

5-2015

A Partial Replication Load Balancing Technique for Distributed Data as a Service on the Cloud

Klaithem Saeed Al Nuaimi

Follow this and additional works at: https://scholarworks.uaeu.ac.ae/all_dissertations

Part of the [Philosophy Commons](#)

Recommended Citation

Al Nuaimi, Klaithem Saeed, "A Partial Replication Load Balancing Technique for Distributed Data as a Service on the Cloud" (2015). *Dissertations*. 37.

https://scholarworks.uaeu.ac.ae/all_dissertations/37

This Thesis is brought to you for free and open access by the Electronic Theses and Dissertations at Scholarworks@UAEU. It has been accepted for inclusion in Dissertations by an authorized administrator of Scholarworks@UAEU. For more information, please contact fadl.musa@uaeu.ac.ae.



جامعة الإمارات العربية المتحدة
United Arab Emirates University

United Arab Emirates University

College of Information Technology

A PARTIAL REPLICATION LOAD BALANCING TECHNIQUE
FOR DISTRIBUTED DATA AS A SERVICE ON THE CLOUD

Klaithem Saeed Al Nuaimi

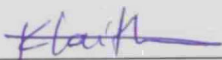
This dissertation is submitted in partial fulfilment of the requirements for the
degree of Doctor of Philosophy

Under the Supervision of Dr. Nader Mohamed

May 2015

Declaration of Original Work

I, Klaithem Saeed Al Nuaimi, the undersigned, a graduate student at the United Arab Emirates University (UAEU), and the author of this dissertation entitled "*A Partial Replication Load Balancing Technique for Distributed Data as a Service on the Cloud*", hereby, solemnly declare that this dissertation is an original research work that has been done and prepared by me under the supervision of Dr. Nader Mohamed, in the College of Information Technology at UAEU. This work has not been previously formed as the basis for the award of any academic degree, diploma, or a similar title at this or any other university. The materials borrowed from other sources and included in my dissertation have been properly cited and acknowledged.

Student's Signature 

Date 1st June - 2015

Copyright © 2015 Klaithem Saeed Al Nuaimi
All Rights Reserved

This Doctorate Dissertation is approved by the following Examining Committee Members:

1) Advisor (Committee Chair): Dr. Nader Mohamed

Title: Associate Professor

Department: Networking

College of Information Technology

Signature  Date 28/5/2015

2) Member: Dr. Imad Jawhar

Title: Associate Professor

Department: Networking

College of Information Technology

Signature  Date 28/5/2015

3) Member: Dr. Nazar Zaki

Title: Associate Professor and Intelligent Sys. & Software Dev. Tracks Coordinators

Department: Intelligent Systems

College of Information Technology

Signature  Date 1/6/2015

4) Member: Dr. Ahmed Al Faresi

Title: Assistant Professor and Assistant Dean for Research & Graduate Studies

Department of Information Security

College of Information Technology

Signature  Date 1/6/2015

5) Member (External Examiner): Prof. Qing Yang

Title: Distinguished Engineering Professor

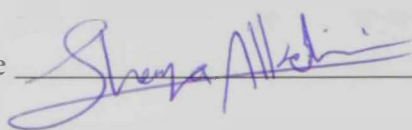
Department of Electrical, Computer, and Biomedical Engineering

Institution: University of Rhode Island

Signature  Date 28/05/2015

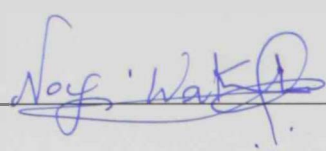
This Doctorate Dissertation is accepted by:

Dean of the College of Information Technology: Dr. Shayma AlKobaisi

Signature 

Date June 23, 2015

Dean of the College of the Graduate Studies: Professor Nagi T. Wakim

Signature 

Date 24/6/2015

Abstract

Data as a service (DaaS) is an important model on the Cloud, as DaaS provides clients with different types of large files and data sets in fields like finance, science, health, geography, astronomy, and many others. This includes all types of files with varying sizes from a few kilobytes to hundreds of terabytes. DaaS can be implemented and provided using multiple data centers located at different locations and usually connected via the Internet. When data is provided using multiple data centers it is referred to as distributed DaaS. DaaS providers must ensure that their services are fast, reliable, and efficient. However, ensuring these requirements needs to be done while considering the cost associated and will be carried by the DaaS provider and most likely by the users as well. One traditional approach to support a large number of clients is to replicate the services on different servers. However, this requires full replication of all stored data sets, which requires a huge amount of storage. The huge storage consumption will result in increased costs. Therefore, the aim of this research is to provide a fast, efficient distributed DaaS for the clients, while reducing the storage consumption on the Cloud servers used by the DaaS providers. The method I utilize in this research for fast distributed DaaS is the collaborative dual-direction download of a file or dataset partitions from multiple servers to the client, which will enhance the speed of the download process significantly. Moreover, I partially replicate the file partitions among Cloud servers using the previous download experiences I obtain for each partition. As a result, I generate partial sections of the data sets that will collectively be smaller than the total size needed if full replicas are stored on each server. My method is self-managed; and operates only when more storage is needed. I evaluated my approach against other existing approaches and demonstrated that it provides an important enhancement to current approaches in both download performance and storage consumption. I also developed and analyzed the mathematical model supporting my approach and validated its accuracy.

Keywords: Cloud Computing, Data-as-a-Service (DaaS), load balancing, storage optimization.

Title and Abstract (in Arabic)

العنوان: تقنية التكرار الجزئي للملفات وتوزيع المهام لخدمة توفير البيانات الموزعة على مسافات كبيرة في السحب الإلكتروني.

المخلص:

توفر هذه الخدمة عملاء مع أنواع مختلفة من الملفات الكبيرة ومجموعات البيانات في مجالات مثل التمويل، والموارد البشرية، وألعاب الفيديو، وغيرها الكثير. وهذا يشمل جميع أنواع الملفات بأحجامها التي تتفاوت من بضعة كيلو بايت إلى مئات تيرابايت. يمكن توفير هذه البيانات باستخدام مركز متعدد تقع في مواقع مختلفة تتصل عادة عبر الإنترنت وتسمى مراكز البيانات. وبما أن هناك الملايين من المستخدمين في جميع أنحاء العالم، يجب على مقدمي الخدمة توفير جودة وفعالية خدماتهم. وضمان هذه المتطلبات التشغيلية من قبل مزود الخدمة وعلى الأرجح من قبل العميل هو تكرار الخدمات على ملفات مختلفة في المخازن التي يتطلب كمية كبيرة من التخزين. يمكن تجنب إدخال مزيد من التحسينات من قبل المستخدمين للخدمة. ولذلك، فإن الهدف من هذا البحث على تحسين جودة الخدمات المقدمة من قبل مقدمي الخدمات هو توفير خدمات التخزين الضخم بوعي إلى زيادة التكاليف، مما يتطلب تكرار الملفات. وبالإضافة إلى ذلك، فإن هناك أيضا زيادة تكاليف التخزين. ومع تعزيز استهلاك التخزين مع طرق سريعة وفعالة للعملاء، مع تعزيز استهلاك التخزين من شأنها تعزيز سرعة عملية التحميل بشكل كبير. يمكن استخدامها لتحميل التجارب المسابقة التي نحصل عليها من البيانات سيكون أصغر من الحجم الإجمالي المطلوب إذا تم تحميلها بشكل تلقائي. وتعمل فقط عندما تكون هناك حاجة مزيد من التخزين. قمنا بتقييم نهجنا ضد النهج القائمة الأخرى وأثبتت أنه يمكن تحقيق توازن بين متطلبات الأداء وتحسين استهلاك التخزين. كما قمنا بتطوير وتحليل نموذج رياضي يدعم نهجنا والتحقق من دقة النتائج. فإذنا نعتقد أنه يوفر نتائج واعدة في مجال موازنة التحميل والتخزين الأمثل للخدمات البيانات على السحابة.

الكلمات ذات الأهمية: السحب الإلكتروني، خدمة توفير البيانات، جودة الخدمات، توزيع المهام، خادم البيانات.

Acknowledgements

First, all thanks be to Allah for providing me with the determination to complete this dissertation. Then, I would like to express my sincere appreciation to my advisor Dr. Nader Mohamed, you have been a great mentor and a very patient person. I would like to thank you for being positive even at hardship. I would like to thank my committee members Dr. Imad Jawhar, Dr. Ahmed Al Faresi, Dr. Nazar Zaki and Dr. Qing Yang for being in this committee, thank you for your bright comments and suggestions which improved my dissertation tremendously. Special thanks to Dr. Jameela Al-Jaroodi for her constant reviews.

I would like to thank UAEU for providing me with the opportunity to join this research and encouraging me to complete it.

A special thanks to my family. Words cannot express how grateful I am to you for all the support and love you provided.

Dedication

To my beloved parents and family

Table of Contents

Title	i
Declaration of Original Work	ii
Copyright	iii
Approval of the Doctorate Dissertation	iv
Abstract	vi
Title and Abstract (in Arabic)	vii
Acknowledgements	viii
Dedication	ix
Table of Contents	x
List of Tables	xii
List of Figures	xiii
List of Abbreviations	xv
Chapter 1: Introduction	1
1.1 Background on Cloud Services	1
1.2 How can the download speed be improved and better utilize cloud resources?	3
1.3 Dissertation Structure	4
Chapter 2: Problem Statement, Contribution, and Research Scope	7
2.1 Problem Statement and Motivation	7
2.2 Research Contribution and Scope	9
Chapter 3: Literature Review	12
3.1 Literature Classification	12
3.2 Research Challenges	13
3.2.1 Spatial Distribution of the Cloud Nodes	13
3.2.2 Storage/ Replication	14
3.2.3 Network Overhead	14
3.2.4 Point of Failure	14
3.3 Load Balancing Approaches	15
3.3.1 Static Load Balancing Algorithms	15
3.3.2 Dynamic Load Balancing Algorithms	17
3.4 Storage Optimization Work	20
3.4.1 Full Replication Storage Work	20
3.4.2 Partial Replication Storage Work	21
3.5 Discussion of Current Approaches	23
3.6 Chapter Conclusion	28
Chapter 4: Collaborative Dual Direction Load Balancing Approach	30

4.1 CDDL.B Methodology.....	30
4.2 Simulation and Analysis of CDDL.B	33
4.3 CDDL.B Benefit and Limitations	35
4.4 Conclusion	35
Chapter 5: Static Partial Replication Technique Using Collaborative Dual Direction Download	37
5.1 SPRT Method.....	37
5.2 Evaluation and Simulation of SPRT	49
5.3 Pros and Cons of SPRT.....	56
5.4 Conclusion	56
Chapter 6: Self-Managed Partial Replication Technique Using Collaborative Dual Direction Download (ssCloud)	58
6.1 Description of ssCloud.....	58
6.2 Example of ssCloud	65
6.3 Analysis and Simulation Results of ssCloud	70
6.4 Enhancements and Limitation of ssCloud.....	82
6.5 Conclusion	83
Chapter 7: Performance Analysis.....	84
7.1 Expected Storage Saved Estimation	84
7.2 Expected Download Time Estimation	90
7.3 Discussion and Observations	94
7.4 Chapter Conclusion.....	98
Chapter 8: Conclusion and Future work	99
8.1 Summary of Research Contribution.....	99
8.2 Future Work	100
Bibliography.....	103
List of Publications	112

List of Tables

Table 2-1: Comparison of Current Industry DaaS Providers in 2015.....	8
Table 3-1: Load Balancing Algorithms, their Pros and Cons	26
Table 3-2: Comparison of Load Balancing Algorithms in Terms of Challenges	27
Table 3-3: Comparison of Storage Optimization Techniques in Terms of Challenges	28
Table 5-1: Example of Block Size Handling MTU	41
Table 5-2: Metadata Size of Different File Sizes.....	55
Table 6-1: Effect of Different Speed of Servers in Three Runs.....	69
Table 6-2: Number of Remaining Blocks Per Server After Removing Unused Blocks.....	70
Table 6-3: Comparison of Storage Optimization Techniques	71
Table 6-4: Experimental Relationship Between NOS, NOC, Block Size and NOB .	80
Table 7-1: Evaluation of the Accuracy of Equation 7.....	90
Table 7-2: Dual Servers Experience in Ten Runs	93
Table 7-3: Equation 9 Accuracy Evaluation	93
Table 7-4: Speed Difference Between the Dual Servers, Affecting Download Time	94

List of Figures

Figure 1-1: Cloud Computing Services Architecture	1
Figure 1-2: DaaS Architecture in the Cloud	2
Figure 2-1: Dissertation Scope.....	11
Figure 3-1: Literature Classification.....	13
Figure 4-1: Partitioning a File in CDDLDB	31
Figure 4-2: Dual Servers Providing One Partition.....	31
Figure 4-3: Simple Example of CDDLDB Mechanism.....	32
Figure 4-4: Comparing CDDLDB Performance to Normal Selection Methods	34
Figure 4-5: Effect of Number of Dual Servers on the Download Time	35
Figure 5-1: SPRT File Download from the Cloud Workflow	38
Figure 5-2: SPRT Replicated Data Removal Process.....	39
Figure 5-3: MTU in the Cloud Network.....	40
Figure 5-4: Cloud Node A File Structure	44
Figure 5-5: Cloud Node B File Structure.....	44
Figure 5-6: Example of File Details in Controller's Database	47
Figure 5-7: Example of Experience Saved in Controller's Database of Each Block .	48
Figure 5-8: SPRT Solution Design	49
Figure 5-9: Storage Needed by SPRT Compared to CDDLDB	50
Figure 5-10: Blocks Downloaded from Server 1	50
Figure 5-11: Blocks Downloaded from Server 2.....	50
Figure 5-12: Storage Consumption in Two Cloud Servers.....	52
Figure 5-13: Storage Consumption in Four Cloud Servers	52
Figure 5-14: Effect of Number of Servers on the Blocks' Replication.....	53
Figure 5-15: Storage of All Blocks After Upload Process	54
Figure 5-16: Storage of the Same Blocks After Running SPRT	54
Figure 5-17: Partial Storage of Four Cloud Servers After Running SPRT.....	55
Figure 6-1: Overall Solution Structure of ssCloud	60
Figure 6-2: Sequence Diagram of File Upload Process.....	61
Figure 6-3: File Structure in the Cloud Servers After Initial Upload	64
Figure 6-4: File Structure in the Cloud Servers After Unused Blocks Removal	65
Figure 6-5: Uploaded File Details in Controller's Database	66

Figure 6-6: Uploaded Blocks Details in Controller's Database	66
Figure 6-7: Uploaded Files Structure in Cloud Servers.....	67
Figure 6-8: Blocks of the Uploaded File Saved as Separate Files in the Servers	67
Figure 6-9: File Download Splitting and Assignment Process.....	68
Figure 6-10: Probability of Removing a Block.....	72
Figure 6-11: Time Difference in Download for Different File Sizes	74
Figure 6-12: Comparison of Full File Upload and Blocks Upload in Terms of Time Taken.....	75
Figure 6-13: Consumed Storage Difference in MB.....	75
Figure 6-14: Download Performance Comparison.....	77
Figure 6-15: Upload Performance Comparison.....	78
Figure 6-16: Error Rate Caused by the Database Server in the Case of an Exceeding Number of Connections	79
Figure 6-17: Block Size Effect on Download Time for 10 MB File Using Two Servers.....	81
Figure 6-18: Block Size Effect on Download Time for 100 MB File Using Two Servers.....	81
Figure 6-19: Block Size Effect on Download Time for 400 MB File Using Two Servers.....	81
Figure 6-20: Block Size Effect on Download Time for 400 MB File Using Four Servers.....	82
Figure 7-1: Number of Replicated Blocks in Two Servers for 1000 Block File	85
Figure 7-2: Number of Replicated Blocks in Four Servers for 1000 Blocks File	86
Figure 7-3: Experimental Relationship Between Min-Max Speed Gap and Maximum Number of Replicated Blocks for 100 MB File Size	96

List of Abbreviations

CLBDM	Central Load-Balancing Decision Model
CDDL B	Collaborative Dual-Direction Load-Balancing
DaaS	Data as a Service
DDFTP	Dual-Direction File Transfer Protocol
ESFWLC	Exponential Smooth Forecast based on Weighted Least Connection
F(R)	Factorization method of R
IDE	Intgrated Development Environment
INS	Index Name Server
LBMM	Load-Balancing Min-Min
M	Number of cloud Nodes
MTU	Maximum Transfer Unit
N	Number of blocks
NOC	Number of Connections
NOS	Number of Servers
OLB	Opportunistic Load Balancing
R	Original File size in bytes
RRNS	Redundant Residue Number System
SPRT	Static Partial Replication Technique
SOF	Single Point of Failure
ssCloud	Smart Storage Cloud
VM	Virtual Machine
WLC	Weigted Least Connection
X	Original File

Chapter 1: Introduction

In this chapter, I provide a background of the cloud structure and services, focusing on data as a service in Section 1.1 then I discuss my research question and a brief summary of the current solutions in Section 1.2. I finally show the dissertation structure in Section 1.3.

1.1. Background on Cloud services

Systems, such as grid, clusters, and cloud computing have been a trend for many users in the last few years. Especially cloud computing which became even of more interest to the users and researchers [1][2][3][4]. One of the main features on the cloud is that it provides flexible and easy methods to store and retrieve data [5][6][7], especially for large data sets and files, such as videos, scientific research, and bioinformatics files [8][9][10] that could be used by an increasing number of users around the world. Since cloud computing has great potential for data storage and data retrieval, it opens the opportunity to conduct research in optimizing the techniques for storing data in the cloud. That is the area of providing data as a service (DaaS) on the cloud, as shown in Figure 1-1.



Figure 1-1: Cloud Computing Services Architecture.

Data as a Service provides the capability to deliver specific and valuable data on demand [11][12]. This data can be business, scientific, medical, or any other useful data required by multiple users. This large data can be replicated on multiple servers located at different sites on the Internet to provide a scalable capability to support a large number of requests. The DaaS is also reviewed in [13] as providing data in different formats for different resources in various geographical locations. The clients would be able to upload, download, and edit the data on the cloud based on their reassigned privileges. Usually, the cloud will have multiple distributed servers, which are able to access the data centers to fetch the required data and provide it to the cloud user. Figure 1-2 shows how the cloud DaaS is usually structured. Distributed DaaS mainly has spatially distributed resources of the cloud and provides the user with access to the data independently from their location. For example, there could be a cluster in one country, some servers in another country, and other clusters in other continent [14].

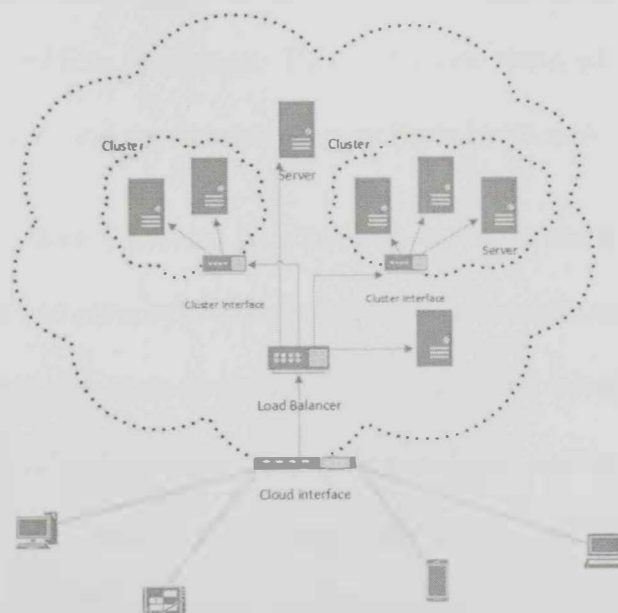


Figure 1-2: DaaS Architecture in the Cloud

Data as a Service provides the capability to deliver specific and valuable data on demand [11][12]. This data can be business, scientific, medical, or any other useful data required by multiple users. This large data can be replicated on multiple servers located at different sites on the Internet to provide a scalable capability to support a large number of requests. The DaaS is also reviewed in [13] as providing data in different formats for different resources in various geographical locations. The clients would be able to upload, download, and edit the data on the cloud based on their reassigned privileges. Usually, the cloud will have multiple distributed servers, which are able to access the data centers to fetch the required data and provide it to the cloud user. Figure 1-2 shows how the cloud DaaS is usually structured. Distributed DaaS mainly has spatially distributed resources of the cloud and provides the user with access to the data independently from their location. For example, there could be a cluster in one country, some servers in another country, and other clusters in other continent [14].

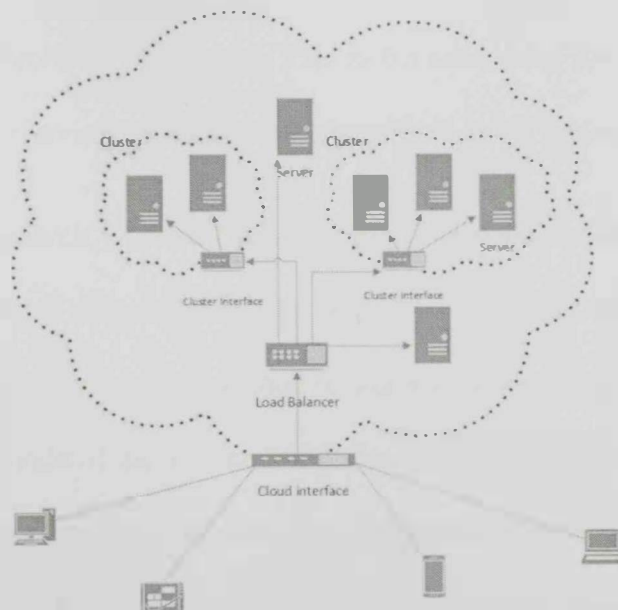


Figure 1-2: DaaS Architecture in the Cloud

1.2. How can the download speed be improved and better utilize cloud resources?

The main focus of this research is to optimize the load balancing and storage interface for cloud computing. The cloud uses multiple servers (usually referred to as cloud nodes) and each node has different performances and load characteristics as well as dynamically varying states of the network links between these servers and the requesting clients; therefore, balancing the load to improve data download is not a trivial task [15][16][17]. There have been some solutions proposed by researchers in cloud DaaS and other distributed systems, such as dual-direction FTP which is concerned with file download among FTP servers, the 'Ant Colony', which assigns an 'ant' to go through a route to pick a free cloud server to perform the task, and many other approaches. However, most of these approaches either focus on improving only the load balancing or improving only the storage consumption. In addition, the mere issue of creating multiple replicas of big data creates another problem of storage. This is because there are huge amounts of storage wasted by saving the same data on multiple cloud nodes [18][19].

In this research, I present an algorithm to reduce the load on each server node of the DaaS and reduce the storage needed for the replicated data sets. This is done using the dual-direction downloading algorithm and based on the experience with each cloud node of those containing data replicas. As a result, I reduced the size of the data files I retain on each node. The main attributes I consider in this research are the number of times each block has been downloaded in earlier requests and the speed of the download. With this information, my smart controller will be able to make all the decisions. Another benefit of this algorithm is that the

client will not have to deal with any complex calculations, which could increase the download time. Therefore, I believe my algorithms speed up the data download process and simultaneously reduce the total amount of storage needed for replications on the cloud servers. I use a special simulator that I built to evaluate the performance of the algorithm and compare it to the other existing ones.

1.3. Dissertation Structure

In the rest of this dissertation, I will introduce the research by reviewing the current problems of load balancing and storage optimization in providing Data as a Service in the cloud in Chapter 2. In addition, the problem statements are presented in Chapter 2.

In Chapter 3, I provide a thorough review of the research area of providing DaaS in the cloud. I classify the research area into multiple levels and review the work done by researchers in the last few years accordingly. I then introduce the challenges faced in this area and the importance of overcoming them in order to provide an efficient method. I also compare the various methods reviewed in the literature according to the challenges and find the limitations of each method. I show that a common limitation between most of the methods used in literature is not being able to provide a method that has a high-speed load-balancing strategy that optimizes the storage used by the cloud provider. I show the importance of having such an approach in order to provide an efficient quality of service for the clients and reduce the cost to providers.

Chapter 4 describes the base approach of using a collaborative dual-direction download method in the cloud. In chapter 4, I describe the advantages of

the dual- direction technique which enhances the speed of the download process in the cloud using collaborative dual cloud nodes in order to provide different partitions of the files. Then, I show the simulation results of using this method in the cloud and how it has better speed compared to the regular method used for file download in the cloud.

In Chapter 5, I demonstrate my first contribution, which is the static storage optimization technique. I show how I improved the collaborative dual direction by partially replicating the storage using download experience. I then discuss the results of optimizing the storage of the cloud servers and compare the enhancements to the previous approaches. In addition, the limitations and possible enhancements of the static storage optimization are discussed.

Chapter 6 elaborates on how a self-managed method of storage optimization can be added to the collaborative dual-direction download technique. Chapter 6 illustrates how the file can go through different stages in the cloud, starting from the upload stage on which the technique splits the file into multiple blocks and saves them each as a separate file in each cloud node to the download stage in which the dual-direction technique is applied and experience is saved. Finally, a discussion of when and in which cases the file blocks can be removed from a certain cloud node is provided. I display the results I obtained when simulating this method and comparing it to the similar approaches reviewed earlier in the literature.

In Chapter 7, I develop analytical models of the partial replication dual-direction download. I demonstrate the effects of the technique on the time spent downloading the file and the amount of storage that can be saved when using this

approach. I then provide some discussion of the results I attained when validating these models.

Finally, Chapter 8 concludes this dissertation by summarizing the contributions and benefits of this research and the possible future works that can be conducted in order to enhance the current results.

Chapter 2: Problem Statement, Contribution, and Research Scope

In this chapter, I discuss the problem and motivation behind this research and the main contribution of this research. I also clarify the scope of my contribution and the areas in which it is important.

2.1. Problem Statement and Motivation

Cloud services have become a trend in the last decade because of their agility, location independence, and cost effectiveness [20]. There are many organizations and cloud providers that offer DaaS [21][22]. These are very common services among users and are very reliable solutions to keep large files and share them. Examples of the most well-known industry applications are Dropbox, Google Drive, Apple iCloud, Microsoft OneDrive, and Amazon EC2 [23][24]. The services provided by each of the mentioned applications vary from providing the ability to upload and share files to the amount of storage provided to the client. Table 2-1 shows a comparison of the most well-known applications in the industry [25]. It was found that free storage provided to normal users ranges from 2 GB to 15 GB. However, premium storage can reach up to 200 GB. This is why the Dropbox application is the dominant application in the market by 47.9%. Dropbox announced recently that the number of their users reached 270 million users [23]. Imagine having at least 2 GB for 270 Million users. The problem here is that storage consumes most of the cost spent to provide the cloud services. As stated by Greenberg [26] in his analysis of cloud costs, data centers consume 45% of the total costs, infrastructure consumes 25% while network and power draw consume 15% each. Therefore, there is a strong need to reduce the cost of data

centers by optimizing the way data is stored. The storage utilization however, must not negatively affect the download speed at the client side or the reliability of the storage and retrieval [27][28][29]. The main focus of this research is to use an effective load-balancing technique to enhance the download performance and optimize storage usage when providing DaaS in the cloud.

Table 2-1: Comparison of Current Industry DaaS Providers in 2015.

Application	Free Storage	Premium Storage	Market Share
Dropbox	2 GB	Unlimited	47.9%
Google Drive	7 GB	200 GB	16.5%
iCloud	15 GB	50 GB	10.5%
One Drive	5 GB	200 GB	9.3%

Cloud resources in the current systems consume a great deal of cost and time from cloud providers [30]. I noticed that there are two main scenarios usually used when providing DaaS on the Cloud for load balancing and storage optimizing. The scenario for load balancing is to look for one server in the cloud and assign the task to that server. This is of course while taking certain attributes into consideration. For example, considering the number of connections that are created with that server or the speed of the server. The problem with this scenario is that the server will be a bottle-neck if I only consider its speed. Moreover, if I consider only the number of connections, the server might be slow but free which will result in a slow download. Regarding storage of DaaS, the scenario is to replicate full files on all servers. The benefit of full replication is having the ability to distribute the load among the cloud servers if needed. However, to do that I need huge storage space which will result in very high costs. imagine the need to replicate a terabytes files among several servers. Here, comes a

question of how I can decrease the cost of storage in the cloud while still using replication and providing a fast download service? My algorithm has the following benefits to other load balancing and storage optimization techniques:

- It does not incur a high overhead, as less communication is needed to finalize a file download from a cloud service.
- It has a better handling of the resources in terms of saving more storage space in the cloud nodes. This is because only parts of the files are saved and each part is referred to with an ID so that the controller will know which cloud node has which partition of the file. Usually, all download algorithms from the cloud focus only on how to improve the speed of the download process and how to specify which node has the file. However, they do not focus on the storage consumption on the cloud nodes and its effect on speeding up the process of assigning the task to the node. In my algorithm, I treat all cloud nodes as parts of a team. This means that all cloud nodes will be busy downloading partitions of the file.

2.2. Research Contribution and Scope

Based on the studies that were conducted and the various possibilities of load balancing in DaaS, I have defined the scope of this dissertation research to address the storage optimization, load-balancing, performance, and efficiency. The main contributions possible to this area are shown in Figure 2-1. There are three main research areas in enhancing DaaS in the distributed cloud; this includes enhancing the speed of exchanging data through the cloud and its efficiency [31][32][33], optimizing the amount of storage needed to host the files on the cloud, and securing the exchange

process. Both storage optimization and task allocation are also considered under cloud resources management research [34]. The cloud resources management is called green cloud computing by many researchers [28]. A green cloud usually aims to enhance the use of cloud resources and reduce the effort and energy spent to accomplish tasks.

The following are the specific contributions of this dissertation:

1. A static optimization of the storage using the dual direction download technique. This contribution allows the cloud providers to improve the download speed using a dual direction download technique and optimize the storage by removing the redundant replicas manually. The benefit of this contribution over the normal dual direction technique is the storage optimization feature. However, the limitation is the need to perform the task manually at a certain stage. A file and block experience are all saved in a database where decisions about block removal can be made.
2. My second contribution is autonomizing the process of storage optimization. This is done by an analysis carried at the upload phase of the file life-time in the cloud. I propose a technique in which uploading any file requires an analysis of the file size and the collaborative servers' available as well as the previous experiences of the download of each block for the registered files. A block would be removed automatically only if there is a need to do so. That is, if there is not enough space available in one server, and there exists previous blocks with download counter equal to zero while the file was downloaded several times from the cloud. The dual direction has also a minor modification as the files will be stored in the cloud as

multiple blocks. Therefore, instead of downloading from one file only, the process will loop through a number of block files in a folder.

3. My final contribution is an analytical model of the amount of storage used when using my ssCloud technique. I analyzed the expected minimum amount of storage that could be saved by the cloud when using ssCloud, I evaluated the expected results and verified the accuracy of my model. Furthermore, I analyze the expected download time when using ssCloud and evaluated the expected results. I found a high percentage of accuracy in my analytical model.

It is important to note here that I only focus on large file sizes. I do not consider any file size below 1 MB as one server can provide such files in a timely manner. Moreover, files with sizes ranging from 1 MB to 10 MB are also convenient to be provided by 1 server without going through the process of assigning tasks to multiple servers. In this approach my main focus is large files with sizes greater than 10 MB.

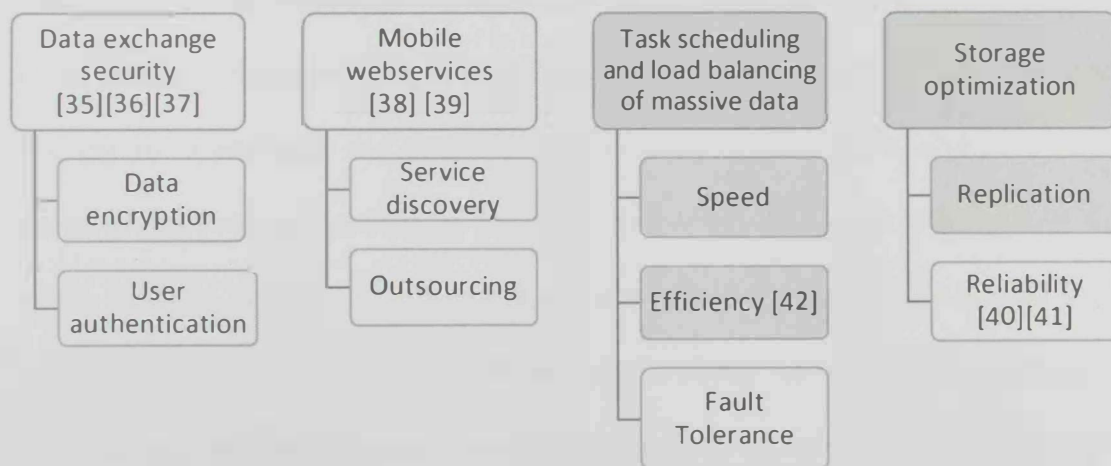


Figure 2-1: Dissertation Scope.

Chapter 3: Literature Review

In this chapter, I provide an analysis of the load balancing and storage optimization research area in Section 3.1. Then, I show the challenges that face most of the techniques reviewed in the literature in Section 3.2. In Section 3.3 I review the load-balancing techniques, while in Section 3.4 I review the storage optimization technique. Then I provide an analysis of the current approaches in Section 3.5. The chapter is finally concluded with Section 3.6.

3.1. Literature Classification

To analyze the state of the art research in DaaS, I thoroughly studied the current approaches in load balancing and storage optimization in the cloud. I noticed some approaches focused on enhancing the load balancing of the file downloads from the cloud [33], while others focused on optimizing storage in the cloud [43][44][45]. Therefore, I classified DaaS research as in Figure 3-1 into two categories: research on load balancing and research on storage optimization. Each category has a sub category based on the common solution provided in the literature. For example, load balancing is categorized into static and dynamic load balancing because some solutions focused on assigning tasks to cloud nodes based on their ability to receive new tasks (static) while dynamic assigns tasks to cloud nodes by taking into consideration the node speed, capacity, and network load. Moreover, the storage optimization is categorized into full and partial replication. This is because some approaches save the same full file on multiple cloud nodes, while others partition the file based on certain characteristics and save different partitions on different servers.

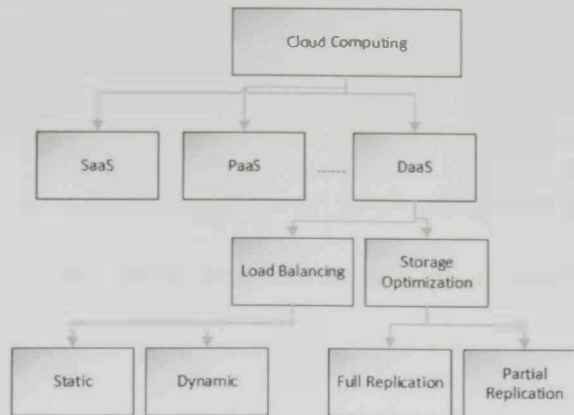


Figure 3-1: Literature Classification.

3.2. Research Challenges

Before I could review the current load-balancing approaches for cloud computing, I must identify the main challenges involved and that could affect how the algorithm would perform. Here I discuss the challenges to be addressed when attempting to propose an optimal solution to the issue of load balancing in cloud computing. These challenges are summarized in the following points.

3.2.1. Spatial Distribution of the Cloud Nodes

Some algorithms are designed to be efficient only for an intranet or closely located nodes where communication delays are negligible. However, it is a challenge to design a load-balancing algorithm that can work for spatially distributed nodes. This is because other factors must be taken into account, such as the speed of the network links among the nodes, the distance between the client and the task processing nodes, and the distances between the nodes involved in providing the service. There is a need to develop a method to control the load-balancing mechanism among all the spatial distributed nodes, while being able to effectively tolerate high delays [46].

3.2.2. Storage/ Replication

A full replication algorithm does not take efficient storage utilization into account. This is because the same data will be stored in all replication nodes. Full replication algorithms impose higher costs since more storage is needed. However, partial replication algorithms could save parts of the data sets in each node (with a certain level of overlap) based on each node's capabilities, such as processing power and capacity [47]. This could lead to better utilization, yet it increases the complexity of the load-balancing algorithms as they attempt to take into account the availability of the data set's parts across the different cloud nodes.

3.2.3. Network Overhead

A network overhead is usually known as straining the network with several connections and messages. Sending and receiving messages through the cloud should be reduced as much as possible so that the network is free to do the tasks assigned more efficiently. Therefore, load-balancing algorithms are preferred have less network overhead [48].

3.2.4. Point of Failure

Controlling the load balancing and data collecting about the different nodes must be designed in a way that avoids having a single point of failure in the algorithm. Some algorithms (centralized algorithms) can provide efficient and effective mechanisms for solving the load balancing in a certain pattern. However, they have the issue of one controller for the whole system. In such cases, if the controller fails, then the whole system fails. Any load-balancing algorithm must be designed in order to overcome this challenge [49]. Distributed load-balancing algorithms seem to provide a better

approach, yet they are much more complex and require more coordination and control to function correctly.

3.3. Load Balancing Approaches

In this section, I discuss the most well-known contributions in the literature of load balancing in cloud computing. I classify the load-balancing algorithms into two types: static algorithms and dynamic algorithms. I first discuss the static load-balancing algorithms that developed for cloud computing. Then, I will discuss the dynamic load-balancing algorithms.

3.3.1. Static Load Balancing Algorithms

Static load-balancing algorithms assign the tasks to the nodes based only on ability of the node to process new requests. Static algorithms do not consider attributes, such as network traffic, nodes CPU speed, node memory size, and other node capabilities.

Radojevic suggested an algorithm called the central load-balancing decision model (CLBDM)[15], which is an improvement of the round robin algorithm, which is based on session switching at the application layer. Round robin [50] is a very famous load-balancing algorithm. However, it sends the requests to the node with the least number of connections. The improvement in CLBDM is that the connection time between the client and the node in the cloud is calculated, and if that connection time exceeds a threshold, then there is an issue. If an issue is found, the connection will be terminated and the task will be forwarded to another node using the regular round robin rules. The CLBDM acts

as an automated administrator. The idea was obtained from a human administrator's point of view.

The proposed algorithm by Nishant [51] is an improvement of the algorithm presented in [52]. Both algorithms use 'ants' behavior to gather information about the cloud nodes in order to assign the task to a specific node. However, the algorithm in [52] has an ant synchronization issue, and this paper is attempting to solve this by adding the feature 'suicide' to the ants. Both algorithms work in the following way, once a request is initiated, the ants and pheromones are initiated and the ants start a forward path from the 'head' node. A forward movement means that the ant is moving from one overloaded node looking to the next node to check if it is overloaded or under-loaded. Moreover, if the ant finds an under-loaded node, it will continue its forward path to check the next node. If the next node is an overloaded node, the ant will use the backward movement to get to the previous node. The addition in algorithm proposed in [51] is that the ant will commit suicide once it finds the target node.

The algorithm proposed in [53] is an addition to the map reduce algorithm [54]. The map reduce algorithm is a model that has two main tasks, map tasks and reduce tasks. Moreover, there are three methods in this model. The three methods are part, comp, and group. The map reduce algorithm first conducts the method by map tasks. At this step, the request entity is partitioned into parts using the map tasks. Then, the key of each part is saved into a hash key table, and the comp method completes a comparison between the parts. After that, the group method groups the parts of similar entities into groups using reduce tasks. Since several map tasks can read entities in parallel and process them, this will cause the reduce tasks to be overloaded. Therefore, it is proposed in this paper to add one more load

balancing between the map task and the reduce task to reduce the overload on these tasks. The load balancing in the middle divides the large blocks into smaller blocks, and then the smaller blocks are sent to the reduce tasks based on their availability.

Ni proposed a load-balancing algorithm [55] for private cloud using virtual machine (VM) mapping to a physical machine. The architecture of the algorithm contains a central scheduling controller and a resource monitor. The scheduling controller does all the work for calculating which resource is able to take the task and assigning it to a specific resource. However, the resource monitor does the job of collecting the details regarding the resources availability. The process of mapping goes through four main phases, which are accepting the VM request, obtaining the resource details using the resource monitor, calculating the resources' ability to handle tasks (the resource with the highest score is the one receiving the task), and accessing the application.

3.3.2. Dynamic Load Balancing Algorithms

Dynamic load-balancing algorithms take into account different attributes of nodes capabilities and network bandwidth. These algorithms assign the tasks dynamically to the nodes based on the attributes calculated. Such algorithms are usually harder to implement but are more efficient.

In [56], they proposed an algorithm to minimize data duplication and redundancy. The algorithm proposed is called an INS (index name server), and it integrates de-duplication and access point selection optimization. There are many parameters involved in the process of calculating the optimum selection point. Some these parameters are hash code of the block of data to be downloaded, the position of the server that has the target block of data, the transition quality, which

is calculated based on the node performance and a weight judgment chart, the maximum bandwidth of downloading from the target server and the path parameter. Another calculation is used to specify whether the connection can handle additional nodes or not (busy level). The authors classified the busy levels into three main categories B(a), B(b), and B(c). The B(a) category means that the connection is very busy, and I cannot add any more connections. The B(c) category means that the connection is not busy, and additional connections can be added. However, B(c) means that the connection is limited, and there is further study needed. The B(b) category is also classified into three further categories; B(b1) means that INS must analyze and establish a backup, B(b2) means the INS must send the requests to the backup nodes, and B(b3), which is the highest level efficiency required, means that INS must reanalyze and establish new backups.

Ren [57] presented a dynamic load-balancing algorithm for cloud computing based on an existing algorithm called weighted least connection (WLC_ [58]). The Weighted Least Connections algorithm assigns tasks to the node based on the number of connections that exist for that node. This is done based on a comparison of the sum of connections of each node in the cloud and then the task is assigned to the node with least connections. However, WLC does not take into consideration the capabilities of each node, such as processing speed, storage capacity, and bandwidth. The proposed algorithm is called exponential smooth forecast based on weighted least connection (ESWLC). The ESWLC improves the WLC by taking into account the time series and trials. The ESWLC builds the decision based on an experience of a node's CPU, memory, number of connections, and load of disk occupation. The ESWLC then predicts which node is to be selected based on exponential smoothing.

The algorithm proposed in [59][60][61] is a dual-direction downloading algorithm from FTP servers (DDFTP). The algorithm presented can be also implemented in cloud computing load balancing. The DDFTP works by splitting an m -long file into $m/2$ partitions. Then, each server node starts processing the assigned task based on a certain pattern. For example, one server will start from block zero and keep downloading incrementally, while another server starts from block m and keeps downloading decrement. Finally, when the two servers download two consecutive blocks, the tasks are considered finished, and other tasks can be assigned to the servers. The algorithm reduces the network communication needed between the client and nodes and therefore reduces the network overhead. Moreover, attributes, such as network load, node load, and network speed, are taken into consideration.

The paper in [62] proposes an algorithm called load balancing min-min (LBMM). The LBMM algorithm has a three-level load-balancing framework. It uses an opportunistic load-balancing algorithm (OLB) [63]. The OLB algorithm is a static load-balancing algorithm that has the goal of keeping each node in the cloud busy. However, the OLB algorithm does not consider the execution time of the node. This might cause the tasks to be processed in a slower manner and could cause some bottlenecks since requests might be pending while waiting for the nodes to be free. The LBMM algorithm improves the OLB algorithm by adding three-layered architecture to the algorithm. The first level of the LBMM architecture is the request manager, which is responsible for receiving the task and assigning it to one service manager in the second level of the LBMM. When the service manager receives the request, it divides it into subtasks in order to speed up processing that request. A service manager would also assign the subtask to a

service node, which is responsible for executing the task. The service manager assigns the tasks to the service node based on different attributes, such as the remaining CPU space (freeness of the node), remaining memory, and the transmission rate.

3.4. Storage Optimization Work

There has been some interesting work on storage optimization in the cloud. I noticed that some of these works focused on either dealing with large file sizes or small size files. Moreover, most of the approaches dealing with small file sizes replicated the full file over all the cloud resources. However, the approaches dealing with large file sizes usually split the file onto multiple cloud servers and had a partial replication only. Here, I show the storage optimization related works.

3.4.1. Full Replication Storage Work

Zhang [64] proposed a full replication solution that targets the download of small files from the cloud. The solution is referred to as BerryStore. The targeted file size is a maximum of 10 MB. The advantage of this solution is to group many small files into one large file for which there is only one directory in the cloud nodes. This will result in minimizing the search and queries of the small files where there will be only one query method for all small files. The main structure of the solution is the client, NameServer, and DataServer. The client requests the file, the NameServer attains the location of that file (in which large file it is located), and the DataServer contains the real file data from which the client can download the actual file. The solution is good, yet not practical for large files. Moreover, the

solution replicates the grouped large files on multiple cloud nodes, which can be enhanced by reducing the replication time.

3.4.2. Partial Replication Storage Work

Srivastava [65] proposed another solution that works for multi-cloud storage and within each cloud. It reduces the migration effort of the client data from one cloud to another. Each cloud contains multiple clusters, Virtual Machines (VMs) and physical servers. Therefore, for each cloud there will be a CloudInterface and for each cluster, there will be a ClusterInterface. The purpose of having interfaces is to organize the interactions between each client and each cluster within the cloud. Moreover, there is a broker that obtains the client's request and processes it to the multi-clouds. The client submits requests to the broker to either upload or download. For an upload request, the client specifies the security level. The 'SecurityLevel' is a parameter used by the 'FileSplittingFunction' to split the file into multiple files based on the percentage of security level provided by the client. For example, if the client specifies the security level to be 50%, then the file will be split into two sub files each saved in a different location. For each cloud, the number of sub files is equal to the number of free VMs. The limitation of this approach is its complexity. Especially when the files are saved in different clouds, the operation will be more complex.

Villari et al. [66][67][68] proposed the redundant residue number system (RRNS). Their main concern was the security of the client files hosted in the cloud. It is similar to Srivastava's solution. However, it is different in terms of keeping the metadata of each partition and its location in the cloud at the client side as an XML file. This is to increase the security of the files because the only one who can

collect all the partitions and create the original file will be the client. The number of file partitions is specified by the client. The solution is also useful for clients dealing with multi-cloud providers. Another parameter specified by the client is the redundancy degree (RD), which refers to the number of replicas of the partitions in each cloud node. The solution has four phases, splitting, dissemination, retrieval, and reconstruction. The problem is that if the client has lost the metadata of the partitions' locations, the client will not be able to download the file. Moreover, each file chunk is saved on the cloud nodes as XML files. Therefore, more processing is needed to convert them to their original formats.

There are approaches to enhance the storage consumption in the cloud of clouds. These approaches consider avoiding vendor lock-in, enhancing the security and privacy, and enhancing the cost of replicating full data across multiple providers in the cloud. These approaches include some popular work such as RACS [69], DEPSKY [70], SafeStore [71], and Hybris [72]. These solutions deal with the service provider architecture as a black box, they integrate their solutions with the storage provider so that there is data gathering by a local server at the client side by requesting data existing in each service provider. The service provider's storage architecture and load balancing technique is not touched and therefore, there is a latency to the download time of the file eventually. The approaches are very useful for avoiding vendor (service provider) lock in issue.

This means that the client will suffer minimal effects if the vendor goes out of business or did not provide sufficient service to satisfy the client. Although replicating even partitions of the data across multiple vendors will increase the cost for the client as discussed in [69][70][71] and have a small latency to the download time, it offers a very suitable solution to prevent the service provider from having

access to the full data of the client and it would help the client to be somehow independent from the service provider.

3.5. Discussion of Current Approaches

As discussed earlier, the different approaches offer specific solutions for load balancing that suit some situations but not others. The static algorithms are usually very efficient in terms of overhead, as they do not need to monitor the resources during run-time. Therefore, they would work very well in a stable environment where operational properties do not change over time and loads are generally uniform and constant [73][74]. The dynamic algorithms, on the other hand, offer a much better solution that could adjust the load dynamically at run-time based on the observed properties of the resources at run-time. However, this feature leads to high overhead on the system, as constant monitoring and control will add more traffic and may cause more delays [75]. Some newly proposed dynamic load-balancing algorithms try to avoid this overhead by utilizing novel task distribution models [76][77].

Table 3-1 shows a comparison among the reviewed algorithms. The comparison shows the positive and negative points of each algorithm. For example, the INS algorithm is able to handle the load balancing dynamically. However, the provided algorithm is complicated, which could cause high implementation complexity. I foresee that a close examination of the algorithm and changing the overall structure may result in a less complex algorithm. Furthermore, the CLDBM algorithm solves the problem of requiring a human administrator to control the system all the time. Therefore, it provides a centralized controller. However, if the centralized controller fails at any time, the whole system will not

be able to operate, which will cause a system failure. Having a backup of the central controller could solve the issue for CLDBM in cases of failure.

As for the ant colony approach, I can see that the decentralized approach provides a good solution to the single point of failure issue. However, it could easily cause a network overload due to the large number of dispatched 'ants'. In addition, several operational factors are not being considered, which may result in poor performance. This algorithm can be further improved by introducing better evaluation mechanisms that take into consideration the status of the node and its current available resources. In addition, it may also be possible to limit the number of ants used in the discovery process by introducing search controls that could reduce the branching levels required in the search.

In DDFTP, the control is kept to a minimum and no run-time monitoring is needed to keep up with environment changes, while keeping a very efficient load balancing. As a result, it provides a good approach, yet it still needs some improvements for better utilization of the available resources. One possibility is to find a good model that will reduce the level of replication needed, while maintaining the same level of performance. This may be possible with the consideration of partial replications with a certain level of overlap that will enable more efficient resource utilization and maintain minimum overhead for load balancing.

Table 3-2 illustrates a comparison between the reviewed algorithms in terms of the challenges discussed in Section II. For example, the only algorithm that avoids data redundancy and storage replication is the INS algorithm. However, INS is a centralized algorithm and therefore has a single point of failure. Moreover, it is a complex algorithm.

On the other hand, DDFTP relies on replicated resources and does not reduce the storage size required, but it has a dynamic decentralized approach to balance the loads. It is also a much simpler algorithm to download stored data. By applying partial replication, DDFTP can be improved to use less storage. Generally, each algorithm satisfies a partial set of these challenges, which makes it suitable for specific situations that match the addressed challenges. For example INS, CLBDM, and VM mapping all have a single point of failure, thus they would function very well in a very stable environment where the resource reliability is very high. Moreover, all algorithms except for ant colony and VM mapping can handle a highly distributed environment. Therefore, they are more suitable for the public cloud than the other two. In addition, all but DDFTP introduce high overhead on the network. As a result, if the network conditions worsen, they would all suffer significantly as more delays will be involved, which will delay the overall load-balancing process. However, DDFTP would be more capable in handling such delays, as it does not need to rely on run-time monitoring and controls.

Table 3-1: Load Balancing Algorithms, their Pros and Cons.

Algorithm	Pros	Cons
INS	Initially proven to handle certain sorts of dynamic balancing.	Does not have a forecasting algorithm to identify how the behavior of the nodes will be in the future. Only certain parameters are taken into consideration, such as distance and time.
ESWLC	Reduces the server load issue which exists the original WLC	Complicated. Prediction algorithm requires existing data and takes a lot of time for processing.
CLDBM	Solves the issues of the round robin algorithm. Automated task forwarding eliminates the need for a human administrator at all times.	Inherits round robin issues, such as not taking into consideration node capabilities. Single point of failure (if CLBDM fails, then the whole process would fail). The threshold might not be applied to all cases.
ANT COLONY	Best-case scenario is that the under-loaded node is found at the beginning of the search. Decentralized, not a single point of failure. Ants can collect the info in faster manner.	Network overhead because of the number of ants. Points of initiation of ants and number of ants are not clear. Node's status change after ants visits to them is not taken into account. Only availability of node is being considered, while there are other factors that should be taken into consideration.
Enhanced Map Reduce	Less overhead for the reduce tasks.	More processing time. Reduce tasks capabilities are not taken into consideration.
VM Mapping	Reliable calculation method.	Single point of failure. Does not take into account network load and node capabilities.
DDFTP	Fast. Reliable	A full replication requires full storage in all servers.
LBMM	Reliable tasks assignment to nodes.	Slower than other algorithms because work must pass through three layers to be processed.

Table 3-2: Comparison of Load Balancing Algorithms in Terms of Challenges.

	Replication	Single Point of Failure (SOF)	Network Overhead	Spatial Dist.	Fault Tolerance
INS, 2012	Partial	Yes	Yes	Yes	No
ESWLC, 2011	Full	No	Yes	Yes	Yes
CLBDM, 2011	Full	Yes	Yes	Yes	No
Ant-Colony, 2011	Full	No	Yes	No	Yes
Enhanced Map Reduce, 2012	Full	No	Yes	Yes	Yes
VM Mapping, 2011	Full	Yes	Yes	No	Yes
DDFTP, 2013	Full	No	No	Yes	Yes
LBMM, 2011	Full	No	No	Yes	Yes

As for the storage optimization techniques, since most of the technique architecture rely on having the client containing the metadata of each partition on the service providers, then if the client fails the whole process would fail. On the other hand, most of them are more secure than other load balancing approaches since cloud provider cannot have a full access to the whole data of the client. The latency added to the load balancing download speed cannot be ignored since it is added to the latency of the cloud provider and its architecture is not changed in

all of the techniques. Most of the approaches deal with large file sizes except for berrystore of which goal is collect all small files as one large file and replicate it among several servers which is a full replication of the files. Moreover, CDDL B has a high download speed but a full replication of files over the servers in the cloud. The goals of the storage optimization techniques are different but some of them can be integrated together in order to provide even better performance. For example, since RACS treats the cloud provider architecture as a black box and it solves the issues of security and vendor lock in, it can be integrated with CDDL B in order to provide a faster download and less effects to the client data security.

Table 3-3: Comparison of Storage Optimization Techniques in Terms of Challenges

	SOF	Security	Replication	Client overhead
RRNS, 2014	Yes	High	Partial	High
Berry-Store, 2012	No	Moderate	Full	Moderate
RACS, 2010	Yes	High	Partial	High
Depsky, 2013	Yes	High	Partial	High
Hybris, 2014	Yes	High	Partial	High
SafeStore, 2007	Yes	High	Partial	High
CDDL B, 2013	No	Moderate	Full	Low

3.6. Chapter Conclusion

In this chapter, I have reviewed the state of the art research of providing DaaS in the cloud. From my analysis, I noticed that the current approaches lack the

ability to handle both efficient load balancing and an efficient technique to reduce the storage consumption among the cloud servers. Both of these issues are important in order to provide better services to the client and reduce the cost of hosting millions of files in the cloud. Therefore, I aim in this dissertation to provide a novel technique to solve the issue mentioned earlier.

Chapter 4: Collaborative Dual Direction Load-Balancing Approach

In this chapter, I demonstrate the collaborative dual-direction load-balancing (CDDLB) technique. I show how the technique works in the cloud and explain the basis of CDDL.B. The technique by which files are partitioned and partition tasks are assigned to servers is also illustrated in this chapter. Then, an evaluation of the method is discussed in Section 4.2. Finally, the possible enhancements and strengths of the techniques are demonstrated in Section 4.3.

4.1. CDDL.B Methodology

Here, I describe the collaborative dual-direction download approach. I have applied the technique DDFTP used in FTP file exchange to the cloud in order to allow collaborative servers to provide partitions of the files whenever a client requests that file.

The CDDL.B idea originates from the same approach as DDFTP, which uses a dual-direction download technique in FTP servers [59][60]. The CDDL.B is the dual-direction file retrieval from the cloud servers. The algorithm works by splitting the file into partitions of data as shown in Figure 4-1 and assigning two cloud servers for each partition to download the data from opposite directions. Each of the cloud servers will handle a download of either forward or backward in the partition depending on its assignment. This way, the download process is parallelized across the available replicas, and the overall client download time is improved significantly. In addition, the approach provides an efficient load-balancing scheme with minimum coordination

and zero interaction among the servers being used. However, the CDDL method works well with the existence of full replicas of the data set on each of the cloud server nodes in use.

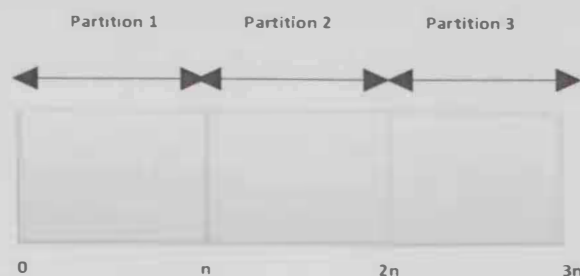


Figure 4-1: Partitioning a File in CDDL.

If I assume that each partition is of length n , then for each set of two cloud servers, the first one will provide the data starting from block index zero and increment its counter to move forward, while the second server will provide the data starting from block index $n-1$ and decrement its counter to move backwards, as shown in Figure 4-2.

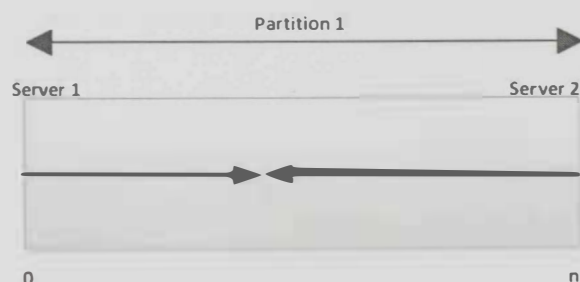


Figure 4-2: Dual Servers Providing One Partition.

Figure 4-3 shows a very simplified example of a download process for a file with four cloud servers in the cloud. When a client requests file X , the request will be forwarded to the load balancer in the cloud. There are several load balancers in the cloud structure; however, requests are generally forwarded to the closest load balancer

in terms of distance. The load balancer will then identify the available cloud nodes to process the task (server 1, 2, 4, and cluster 1); it will partition the file according to the number of available servers into two partitions and then will assign: 1) a forward download task starting from block zero to S1, 2) a backward download task starting from block n to S2, 3) a forward download task starting from block $n+1$ to C1, and 3) a backward download task starting from block $2n$ to node S4. The speed of each of the cloud nodes differs according to its performance, which is the benefit of the dual-direction download process. If a certain server is slow when providing its task, the collaborating server can overcome this limitation by providing the blocks in its direction.

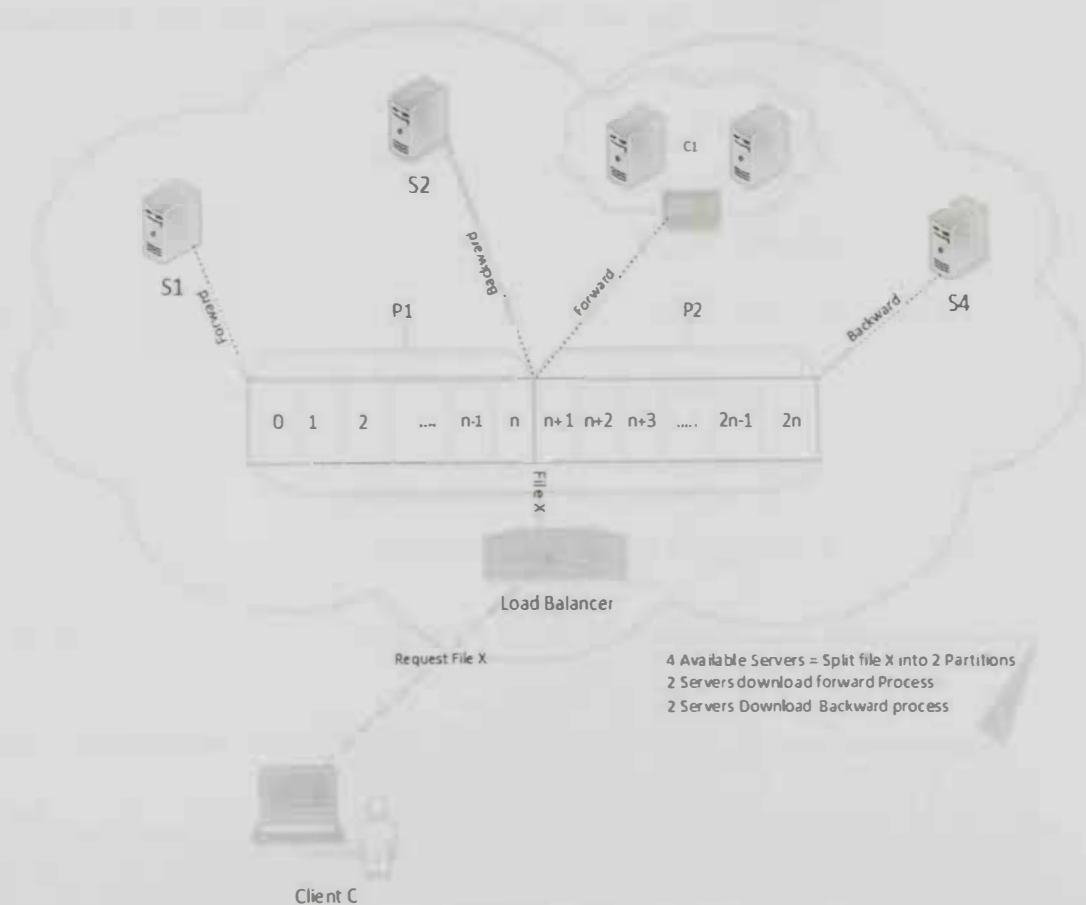


Figure 4-3: Simple Example of CDDLB Mechanism.

Since any file X will be downloaded collaboratively between multiple servers and each set of two servers will collaborate to download one partition, the equation to calculate the number of partitions needed for any file X is by dividing the number of available servers over two. Moreover, a partition size is decided as shown in Equation 1 by the number of blocks (N) and number of available servers (M). For example, if I have a file X with 100 blocks and four available servers, then the partition size for each set of two servers is $(100/4)*2$, which is 50 blocks per partition.

$$\text{Partition size} = \left(\frac{N}{M}\right) \times 2 \quad (1)$$

While the number of blocks N in file X can be found by dividing the file X size (R) by the block size. The equation to find N is as below:

$$N = \frac{R}{\text{BlockSize}} \text{ blocks} \quad (2)$$

It was proven in [61] that the performance of the dual-direction technique is enhanced since the number of control messages (communication) between the client and the cloud servers is decreased to the minimal in reality, using dynamic servers and network loads even when there is a reassignment of the task from one server to another. It is found that the number of start messages would be equal to $k + \frac{k}{2} ((\log_4 n) - 1)$ where k is the number of servers, and n is the number of the last block in the file.

4.2. Simulation and Analysis of CDDL B

To evaluate the proposed algorithm, I consider a data set initially replicated on two cloud servers at different locations that are working according to any normal single node selection algorithm (e.g., ant colony or INS). The size of the data set is 50MB. As this data is replicated on both servers, a total of 100MB are

used. The data set is divided into 5000 blocks of size 10,000 bytes each. I assume that the average download speed from the first server to different clients over the Internet is 20 blocks/second with a minimum speed of 15 blocks/second and a maximum speed of 20 blocks/second. The average download speed from the second server to different clients is 30 blocks/second with a minimum speed of 25 blocks/second and a maximum speed of 30 blocks/second. The average download times using any node selection and assigning technique and CDDL B are shown Figure 4-4. As I can see from the figure, CDDL B provides a good download performance, as it utilizes both servers and provides efficient load balancing regardless of the load on the servers and the networks.

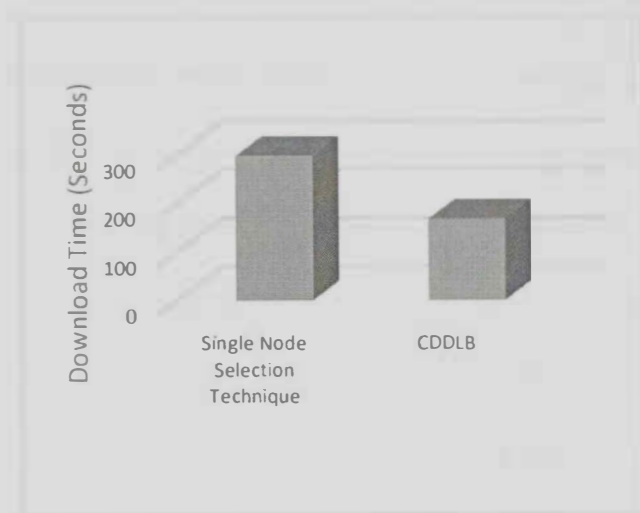


Figure 4-4: Comparing CDDL B Performance to Normal Selection Methods.

In order to check the effect of the processing speed, I simulated the file download speed using various numbers of dual servers for a 100 MB file. I first conducted an experiment using only two servers. Then, I conducted more experiments by increasing the number of servers to four, six, eight, and up to ten servers. The time needed in order to process the request reduced each time I increased the number of servers. Figure 4-5 shows the finishing time of each

processing time done by the number of servers specified. As discussed earlier, in a real cloud the speed and load of cloud servers' change every second.

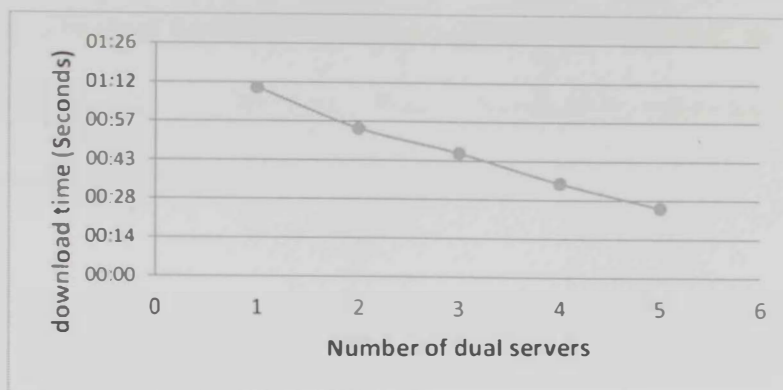


Figure 4-5: Effect of Number of Dual Servers on the Download Time.

4.3. CDDL B Benefits and Limitations

The CDDL technique works well for file downloading and shows some good results as discussed earlier in this chapter. However, the data storage is still consuming a lot of space on each cloud server, and the same data files are saved on each server. Although some parts of these replicas never get used. This means that the storage consumption is more than needed and therefore, my target is to reduce server storage consumption by improving the CDDL algorithm by applying the partial replication of the data files being saved on each cloud server. This means that I will not store the same data file on all cloud servers. I would store different parts of the data files on each cloud server according to the servers' performances throughout the various times download requests were performed on each server.

4.4. Conclusion

In this chapter, I discussed the collaborative dual-direction approach to download files from the cloud. The approach simply partitions files into several partitions depending on the number of available cloud servers and assigns each partition to two

servers so that they can provide it collaboratively. Each server will be providing partition blocks either forward or backward. The importance of this approach is to enhance the download speed of large files in the cloud. However, the limitation here is the need to replicate full files in the cloud. This could be enhanced using the partial replication methods discussed later in this thesis.

Chapter 5: Static Partial Replication Technique Using Collaborative Dual Direction Download

In this chapter, I discuss my static partial replication technique (SPRT), which uses the collaborative dual-direction download in order to make decisions. First, I discuss the technique, its workflows, and needed procedures. Then, I evaluate the performance of this technique and how it proved to have a significant improvement over the other methods, including the CDDL technique, in terms of storage. I finally conclude the chapter by discussing the limitations of the technique and how it can be enhanced further to provide better results.

5.1. SPRT Method

To implement SPRT, I used the workflows shown in Figures 5-1 and 5-2. Figure 5-1 describes the workflow of downloading a file by the cloud client. To download a file, the client initiates a request to the cloud. The cloud controller then checks whether the file was downloaded before, and if so, there will be data regarding the file partitions that were downloaded and which cloud servers provided them. Having this history will help in selecting which cloud server must provide which partition. The controller finds the required data from the database and then assigns the servers, which already have the file partitions to the tasks. After the data is downloaded from all the servers, the client is updated by the required file. However, there must be a first-time download for each file to get its experience. Therefore, the alternative workflow is selected when the file is being downloaded for the first time. The file size in bytes is fetched; the block size is determined by factorizing the file size. Then, servers are assigned based on their availability and processing speeds. When the dual-

direction download is processed from all servers for the first time, the client is updated as well as the database. A database must always be updated with what happens in the servers processing each partition so that the controller can decide later which partitions are to be kept in the cloud server and which are to be removed.

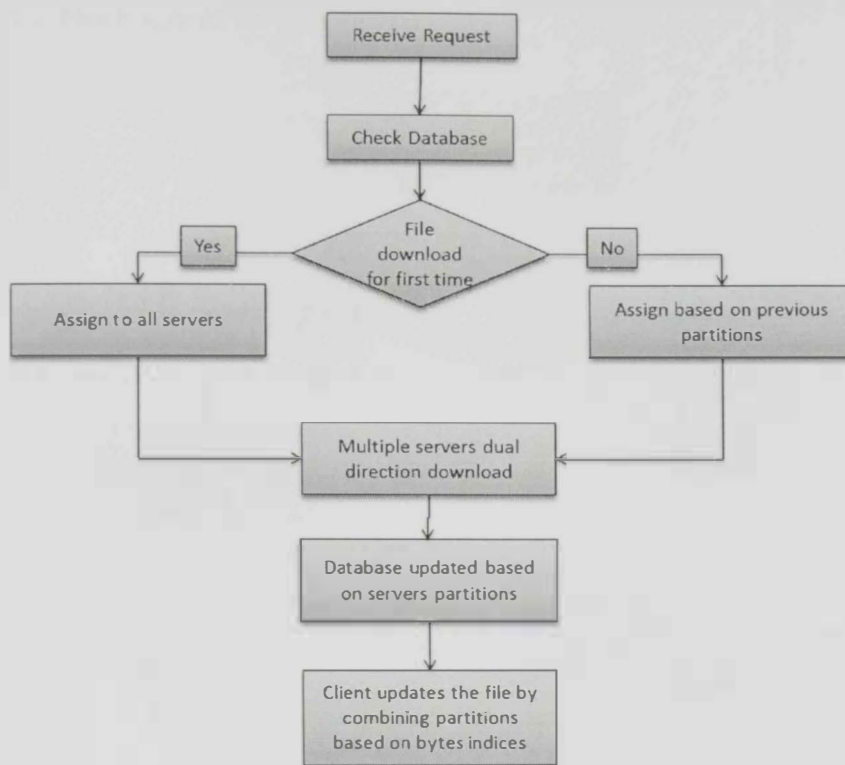


Figure 5-1: SPRT File Download from the Cloud Workflow.

I allow the file partitioning process at the controller side when the controller has enough data to make its decisions. Figure 5-2 illustrates how the controller saves the required partitions on the servers and removes the redundant partitions based on their download rate. To do that, the controller first checks the available data in the database concerning the download from the previous servers' experiences. Then, if blocks downloaded from server S (for example) were found, the controller creates a directory in server S where the directory name is the file X ID. Inside the server folder, the blocks that were downloaded from that server are copied. Each block will be a file by

itself and the name of the file will be the block ID. I tested splitting the original file into the blocks and combining them by the client. The original file was created at the client without any problems. Therefore, this could be the best way to keep partitions of the file in the server without the need for complicated calculations. The file sizes will match the block size in the original file.

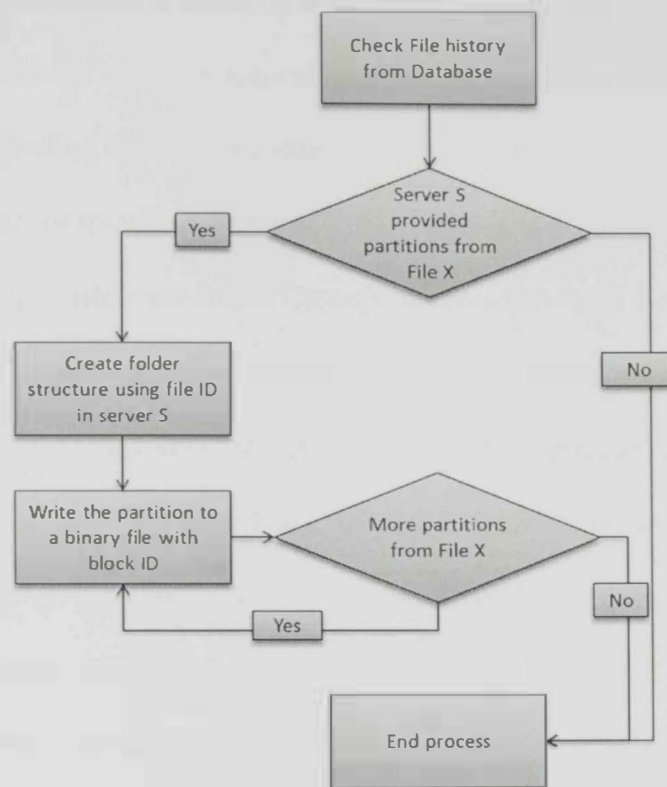


Figure 5-2: SPRT Replicated Data Removal Process.

Moreover, the block size should be selected based on the original problem size (file X size). To do that, I factorize the original file X size and find the biggest factor that belongs to the interval from zero, which is the minimum file size to $\{(\frac{X}{NOC}) \times NOS\}$ that refers to the file size divided by the maximum number of connections allowed by the database server (NOC) multiplied by the number of servers (NOS). This interval will prevent any “exceeding number of connections” errors for the users when

uploading their files to the cloud servers. Since I keep the metadata in the database, it is important to consider the database server's ability to receive the updated connections.

$$\text{BlockSize} = \text{Max}(f(x)) : \text{where } f(x) \in \left\{0, \left(\frac{x}{\text{NOC}}\right) * \text{NOS}\right\} \quad (3)$$

Another problem that could be faced when transmitting a file, even through the cloud, is the maximum transmission unit (MTU). Even if I found a block size that will not face an "exceeding number of connections" error, I could face the MTU error for which a block can be transferred several times because it exceeds the MTU with even one byte. Having a file transferred through several networks will result in having different MTUs for each one. For example, Figure 5-3 shows a file being transferred through a cloud that has an MTU of 1500 bytes, and between the cloud and the client, which has an MTU of 1000 bytes. The 1500-size blocks that passed through the cloud will not be able to go through the cloud-client network because the MTU there is less. Therefore, each block of >1000 will be transferred as two blocks of 1000 and 500. This will consume time from the transfer process.

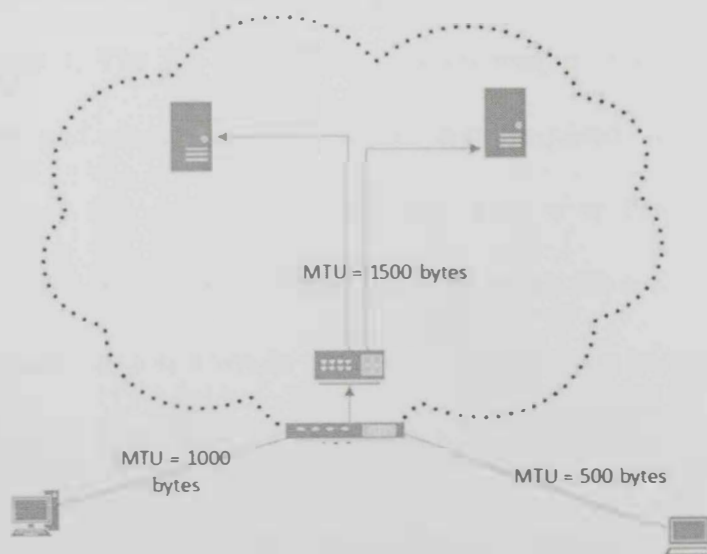


Figure 5-3: MTU in the Cloud Network.

When I say that an MTU is 500, it is really 512 bytes and 1000 is really a 1024 bytes (which is double). The benefit here is that any factorial result from Equation 1 is actually divisible by any of the multiples of 512, 1024, or 1536. Therefore, when a block size of 51,200 of a certain file is defined, this block will be transferred through the network based on the MTU, as in the table. The table shows that after selecting the minimum MTU in a certain route of the file transfer, the block can be split into several packets without any remaining packets.

Table 5-1: Example of Block Size Handling MTU.

MTU	Real packet size	Example Result
500	512	$51200/512 = 100$ packets
1000	1024	$51200/1024 = 50$ packets
1500	1536	Note: Reduce packet size to 1280 $51200/1280 = 40$ packets
2000	2048	$51200/2048=25$ packets

The pseudo code in Algorithm 5-1 shows how the block size is determined based on Equation 3. The file size is first acknowledged. Then, the factorization method is applied, and when the largest number in the required interval is found, it is updated in the block size table in the controller. This is so that the block size is determined for all servers and all download times when the file is first uploaded to the cloud. The file is uploaded as a whole in the cloud without any additional procedures except determining its blocks size for download purposes.

Algorithm 1 upload a file to the cloud

Require: x_i , $FileName_i$, NOC , NOS

Ensure: $x_i > 0$, $NOC > 0$, $NOS > 0$

```

1: set  $k = 0$ 
2: set  $UpperLimit = (x/NOC) * NOS$ 
3: for  $i = 0$  to  $i = UpperLimit$  do
4:   if  $x \mid i$  then
5:      $BlockSize \leftarrow i$ 
6:   end if
7: end for
8: for  $t = 0$  to  $t = x$  do
9:   for  $s = 0$  to  $s = NOS$  do
10:    transfer  $block_t$  to  $Server_s$ 
11:    append to  $TargetFileName$ 
12:   end for
13: end for

```

Ensure: $TargetFileNameSize = x$

During my experience, I found that the number of replicated blocks in more than one cloud node is associated with the number of coordinated nodes in the download process. It is also associated with the load assigned to each server and the speed of the cloud server. For example, if I had only two nodes downloading the file and both nodes have the same load and the same speed, then the number of replicated blocks on the two servers from the file will be two. While when the number of nodes downloading the file is four, the number of replicated blocks will be four, and if one of the dual servers was faster than the second server, then the number of replicated blocks could increase to six. This is because one server processes the request much faster than the other one, and for the other server to reach it, more blocks are replicated.

Therefore, if I have four replicas of a data file on four cloud servers, then I need to divide the file into $4/2 = 2$ partitions. If the data file X has 3000 blocks for example, then each partition will be of size $(3000/4)*2 = 1500$ blocks. Assuming I have the cloud servers A, B, C, and D. The first time the request is initiated, the controller will look for the free servers and assign the partitions to them accordingly. In this example, partition 1 will be assigned to servers A and B. Server A will provide the forward download of partition 1, while server B will provide the backward download of the same partition. As the servers push the blocks, they also update their blocks' download counters as in Tables 1 and 2, where the partition is of size P and server A downloads from zero onwards and server B downloads from $P-1$ downwards until they meet at blocks k and $k+1$.

Similarly, the second partition is assigned to cloud servers C and D, and they both keep similar tables. These tables are updated every time a download request is assigned to the servers for the same file. This will allow the servers to know which blocks are being used and which are not. Over time and with the repetitions of the downloads, the servers can decide to remove the blocks that are never used from storage. This way if I examine servers A and B, after a while I may find that server A has pushed blocks zero to k at least once, while the remaining blocks in the partition were never used. In addition, server B has pushed blocks $P-1$ to block j at least once, while the others were never used. In this case, the controller may decide to instruct server A to delete blocks $k+1$ to $P-1$ and server B to delete blocks zero to $j-1$. Assuming varying performance and loads on the two servers, j will usually be smaller than k , thus there will be some overlap across the servers to ensure proper download in the upcoming requests. For this approach to work correctly, I must ensure that the downloads on particular servers are always done in the same direction. For example,

cloud server A will always be assigned to start from the beginning of a partition, while cloud server B will always start from the end of the partition. The same applies to all servers participating in the overall download process.

As more requests are initiated for downloading a specific file, the controller will be able to remove some blocks from each partition on the cloud servers. Simultaneously, the download process will continue normally for future requests without noticing the partial replications. This will allow us to reduce the storage needed on the cloud servers, while achieving better levels of performance for the client. The partial replication of the load-balancing algorithm performs better as the number of downloads increases. This is because more information about the cloud servers becomes available for the evaluation of their ability to obtain which part of the file. Figures 5-4 and 5-5 demonstrate how the file blocks are stored as file structure in the cloud servers to simplify the search process of the partitions blocks for the client. Moreover, to secure the other files hosted by the cloud server from being accessed by the wrong clients.

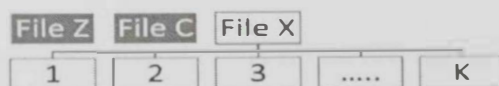


Figure 5-4: Cloud Node A File Structure.

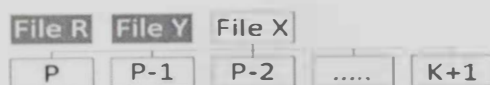


Figure 5-5: Cloud Node B File Structure.

Algorithm 5-2 and 5-3 show the pseudo code of the partition removal at the server level in the cloud. The main idea involves copying the file blocks into other

smaller files based on the block IDs in each server node. After that the original file is removed. To save partitions of the file into the cloud servers, I first check the existing experience saved for that file. This experience is saved in a database that is available with the controller. All the rows saved for that specific file will be retrieved. Then, for each server that provided a partition of the file, a directory will be created in that specific server containing the file ID. This is so that it becomes easier for the server to find the data for that file. When the directory is created, the method will check the database for which blocks were downloaded from that server. As long as there are blocks downloaded from the server by checking the attribute 'DownloadCounter' in the controller table, a small file containing the block IDs will be created in the directory and the binary will be written to the file starting from the first position of the block till the last position. The new file size will match the block size. Therefore, I made sure that there is no additional storage needed when writing the partition of the original to the new small files. Moreover, when downloading a file, each block is read and appended to the resulting file on the client side, and its size is also matched to the original file size and the sum of the blocks sizes, which confirms that there is no additional storage needed when splitting the file into multiple blocks files. Moreover, when removing a block, I ensure that the file was downloaded several times before, while the block was not downloaded from that server at all.

Algorithm 2 Remove blocks for additional storage

 Require: *FileID*

Ensure: none

- 1: Declare *i*
 - 2: *Selection* = Select all from ControllerDB GROUPBY *FileID*
 - 3: *i* ← *Selection*
 - 4: for all *i* do
 - 5: getServerID(*i*)
 - 6: call CreateDirectory Algorithm
 - 7: end for
-

Algorithm 3 CreateDirectory

 Require: *FileID*, *ServerID*

Ensure: none

- 1: Declare *i*, *j*, *path*
 - 2: while *i* ≤ *blocksDownloadedFromSever* do
 - 3: Set *NewPath* ← *directory* + *ServerID* + *FileID*
 - 4: *FileF* = *CreateWriteFile*(*i*)
 - 5: *j* ← *StartPositionOfBlock*
 - 6: while *j* ≤ *EndPositionOfBlock* do
 - 7: WriteBinary(*F*, *j*)
 - 8: end while
 - 9: close File *F*
 - 10: end while
-

Figures 5-6 and 5-7 show an example XML of the data saved in the controller's database. I made sure that the data saved there is minimal so that it does not overload either the retrieval or the storage of the data center. When the file is first uploaded, I add its details, such as the identification number, name, file size, and block size identified based on Equation 3, and I initiate the number of downloads to zero. As there are more requests initiated for that file, the number of downloads will increase. I keep this attribute to compare the block downloads to

the file downloads when attempting to delete any blocks to avoid deleting a block from a file that wasn't downloaded before. As for the 'filesblocksmmap' table, I keep the attributes that will help us in deciding whether or not to delete a certain block from a file. The first three attributes (node ID, file ID, and block ID) will help in determining which block is which and help map it to the cloud node that usually provides it and the file to which it belongs. Then, I add the download counter and the processing type, which is either forward or backward based on the dual-direction approach.

```
<?xml version="1.0" encoding="utf-8" ?>
<!--
- Database: 'controller'
-->
<controller>
  <!-- Table filedownload -->
  <filedownload>
    <FileID>100</FileID>
    <FileName>Important.pdf</FileName>
    <NumOfDownloads>100</NumOfDownloads>
    <FileSize>524</FileSize>
    <BlockSize>256,000</BlockSize>
  </filedownload>
</controller>
```

Figure 5-6: Example of File Details in Controller's Database.


```

<?xml version="1.0" encoding="utf-8" ?>
<!--
- Database: 'controller'
-->
<controller>
  <!-- Table filesblocksmap -->
  <filesblocksmap>
    <rowID>1</rowID>
    <NodeID>0</ThreadID>
    <FileID>1</FileID>
    <BlockID>0</BlockID>
    <DownloadCounter>1</DCounter>
    <ProcessingType>Forward</Processing>
  </filesblocksmap>
  <filesblocksmap>
    <rowID>2</rowID>
    <NodeID>1</ThreadID>
    <FileID>1</FileID>
    <BlockID>2047</BlockID>
    <DownloadCounter>1</DCounter>
    <ProcessingType>Backward</Processing>
  </filesblocksmap>
</controller>

```

Figure 5-7: Example of Experience Saved in Controller's Database of Each Block.

The components of the solution are shown in Figure 5-8. The main components are 1) the clients who initiate the request and send it to the cloud, 2) the load balancer that checks the file download experience from the database and assigns tasks to the cloud servers, 3) the cloud servers that process the requests, and 4) the file controller that does the partitioning on the storage level at the cloud servers after checking the experience of the file downloads.

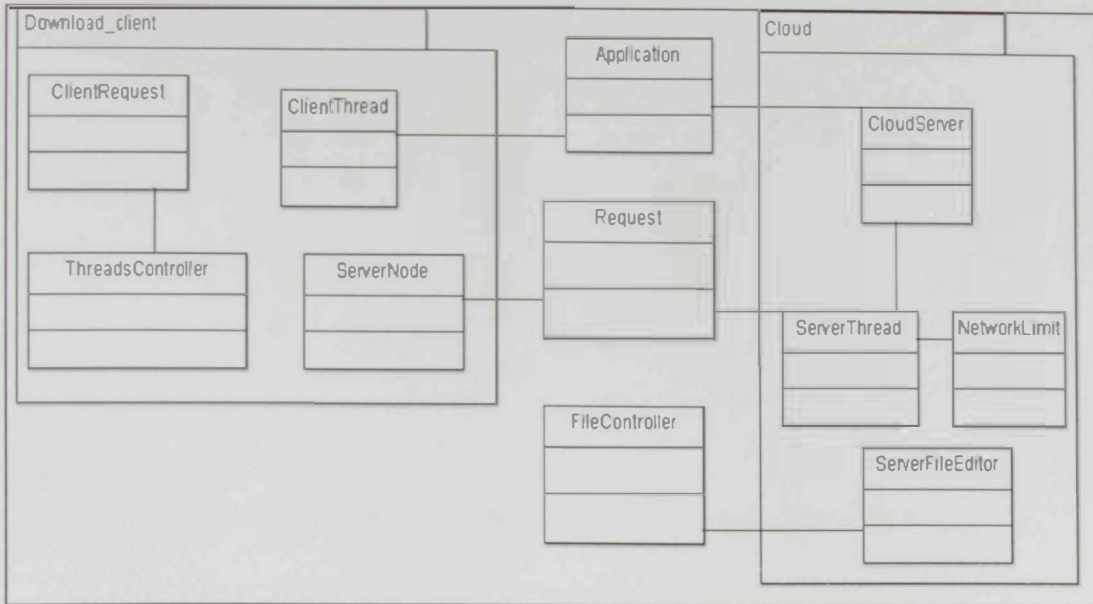


Figure 5-8: SPRT Solution Design.

5.2. Evaluation and Simulation of SPRT

Here, I show an evaluation of the storage enhancements of SPRT over CDDL (discussed in Chapter 4). To evaluate my methods, I implemented my own Cloud simulation environment as shown in the class diagram in Figure 5-8. Servers' speed, network speed, bandwidth, and round trip time are all attributes which I can manipulate to simulate a real cloud network. This simulator follows the same approach used by other models used for other related research [76][77][78].

The first comparison in terms of storage is shown in Figure 5-9. Only 60 MB of space is needed after removing the blocks that have never been downloaded from both servers. The storage space needed by the new approach is reduced from 100 MB to 60 MB (i.e., 40% savings) without increasing the download time compared to DDFTP with full replication.

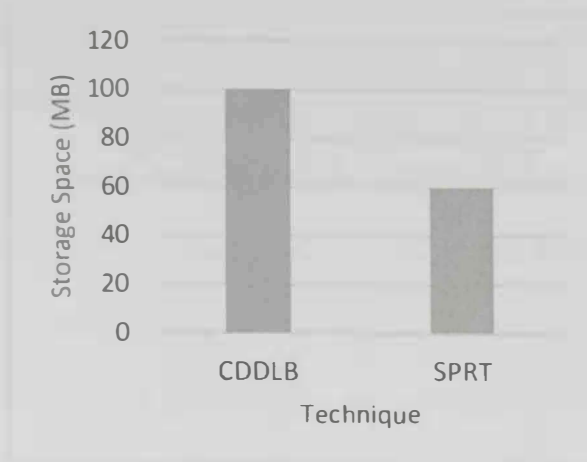


Figure 5-9: Storage Needed by SPRT Compared to CDDL.

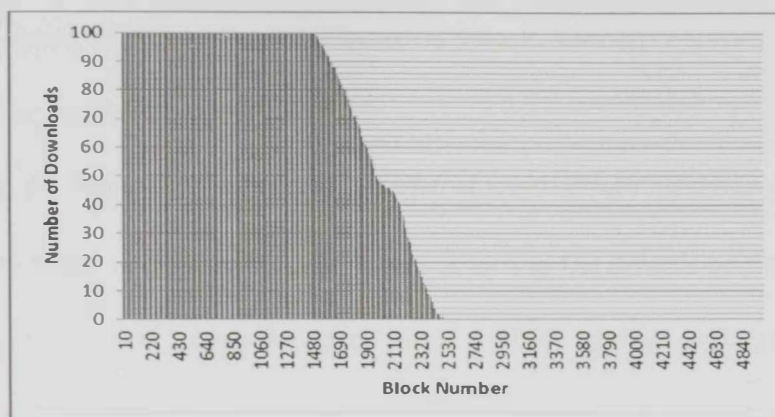


Figure 5-10: Blocks Downloaded from Server 1.

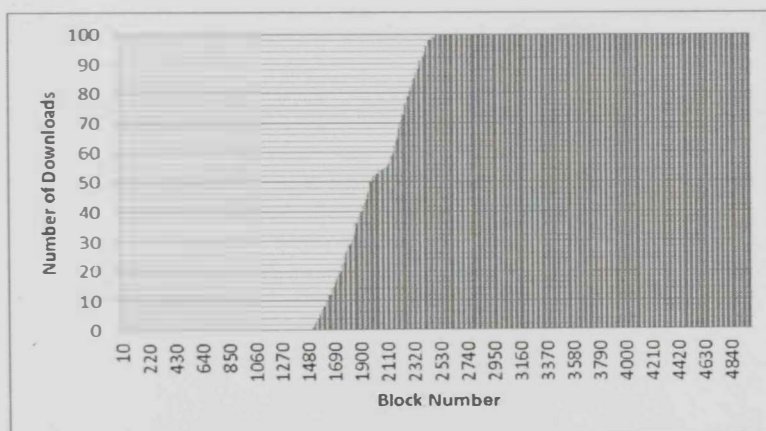


Figure 5-11: Blocks Downloaded from Server 2.

As displayed in Figures 5-10 and 5-11, the first server has never downloaded blocks higher than block number 2500, while the second server has never downloaded any blocks lower than block number 1500. The main reason for

obtaining these numbers is based on two cases. The first case is when the first server was downloading at its maximum speed, while the second server was downloading at its minimum speed. Thus, the maximum block number that the first server downloaded is block number 2500, as both servers will be downloading an average of 25 blocks/second. The second case is when the first server was downloading at its minimum speed, while the second server was downloading at its maximum speed. Thus, the minimum block number that the second server will download is block number 1500, as the speed of the first server is 15 blocks/second, while the speed of the second server is 35 blocks/second. Using the technique developed in this research, it is possible to remove the last 2500 blocks from the first server and the first 1500 blocks from the second server without affecting the parallel download operations and without increasing the download time.

Figure 5-12 compares the space used by SPRT (partial replication) and two of the most used algorithms in load balancing, which are the ant colony load-balancing and map reduce algorithm using different file sizes. I noticed that when the file size increased, my partial replication algorithm improved the storage optimization of the cloud to a greater extent. This is because the difference of the file sizes between my algorithm and other load-balancing algorithms increased.

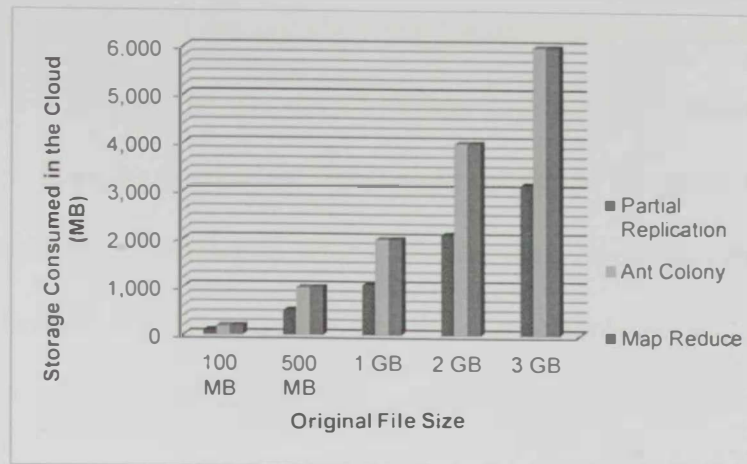


Figure 5-12: Storage Consumption in Two Cloud Servers

When testing the same algorithms using four cloud servers, the difference increased much more, even when the number of replicated blocks in the partial replication algorithm increased. However, the difference between it and the other algorithms was greater because they are based on a full replication of data. Figure 5-13 shows a comparison between the three techniques in terms of storage optimization when using four servers. As more servers are used, I can achieve more savings.

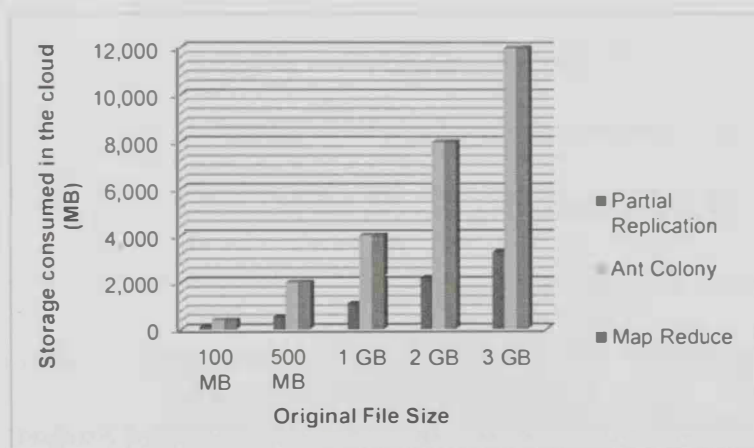


Figure 5-13: Storage Consumption in Four Cloud Servers.

In addition to testing the performance of the algorithm while increasing the number of servers, I also simulated the storage consumed whenever the number of servers increased. To do that, I simulated a download for a 100 MB file using two, four,

six, and eight servers. Then, I ran my algorithm for optimizing the storage of the servers. Figure 5-14 shows the amount of storage consumed for each group of servers. I noticed from the results that the storage consumed increased whenever the number of cloud servers increased. This is because for each dual server working on a partition, there are blocks replicated, and those are the blocks where the two servers meet in the download process. The percentage of the replicated blocks is very low compared to other full replication techniques. The other full replication techniques double the storage consumed as the number of servers increase.



Figure 5-14: Effect of Number of Servers on Blocks' Replication.

I simulated an experiment with a file containing 12,800 blocks using two servers. At the first upload, servers were storing all the file blocks as different files (see Figure 5-15). Then, I downloaded the file a few times using my collaborative dual-direction download approach. Then, I stored several files in both servers 1 and 2 to crowd the storage space and leave little room for new uploads. Finally, I submitted a request by the client to upload a new file. After running SPRT to optimize the storage for the new file, blocks that were not previously provided by each server were removed. Figure 5-16 shows that blocks from 6500 were removed from the server because they were not previously used. Blocks starting from block

zero to 6460 were removed from server 2. There are some blocks that were used by the two servers for downloads and these blocks are left in both servers.

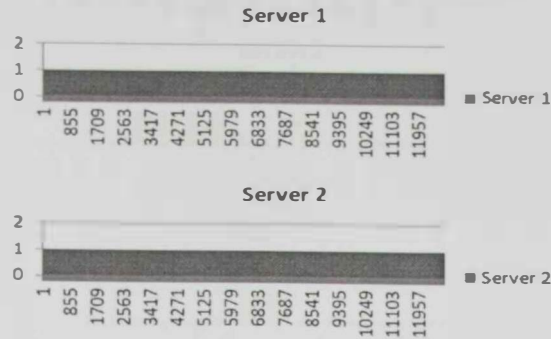


Figure 5-15: Storage of All File Blocks After the Upload Process.

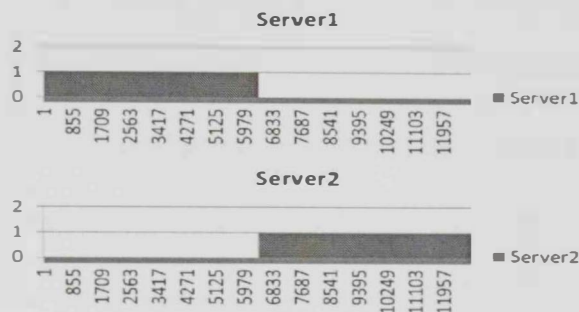


Figure 5-16: Storage of the Same File Blocks After Running SPRT.

When running the same experiment using four servers downloading the same file, the number of partitions increased (see Figure 5-17). If all servers had a full storage and all of them removed the unused blocks, then each server would only save the number of blocks from the original file. The importance of this approach is that although the file does not exist in its entirety on any one of the servers, all the blocks of the original file exist and can be found in the collaborative cloud servers and the file can be reconstructed easily. Moreover, the download process is faster, as there are a number of servers working together to provide the different partitions of the file simultaneously.



Figure 5-17: Partial Storage of Four Cloud Servers After Running SPRT.

An important parameter to evaluate is the amount of additional storage consumed by the metadata of each file. I have checked this parameter and found it does not exceed one megabyte of storage for a one gigabyte file. This amount is minimal compared to the file size. Table 5-2 shows different file sizes that I tried and the amount of storage consumed by their metadata.

Table 5-2: Metadata Size of Different Files

File Size	Metadata size
1 GB	1.3 MB
500 MB	700 KB
100 MB	400 KB
10 MB	200 KB

5.3. Pros and Cons of SPRT

The SPRT method showed promising results in terms of enhancing the performance and storage consumption of the cloud. Therefore, using this technique will reduce the cost of cloud resources used by cloud providers without an effect on performance. The performance of using dual-direction techniques improves the speed and therefore performs better than a regular selection technique as discussed in Chapter 4. In addition, adding a storage enhancement to the dual-direction technique has very good effects on the efficiency of the original CDDL B.

However, using SPRT will result in the need to have manual control over the removal process of partitions; therefore, the basis cannot be determined. Even if the threshold of storage was determined and a removal process was conducted whenever the threshold was reached, it wouldn't be an optimal solution, since the storage resource is not fully utilized. Therefore, I considered the need for a manual control over the SPRT as a limitation of the technique and I have attempted to enhance it, as discussed in Chapter 6.

5.4. Conclusion

In this chapter, I introduced the static optimization technique of cloud storage using the dual-direction download experience. The SPRT saves the experience of each block when downloading the file using a dual-direction technique; therefore, there is a need to store this data in a database in the cloud itself. The technique resulted in a big enhancement of storage compared to the original CDDL B method. The SPRT has a partial replication feature on which there are few blocks that will be saved in multiple cloud servers. Therefore, even if there was a failure of any of the cloud servers, there

are backup blocks in another one. I only remove the previously unused blocks. By this method, I preserve the reliability of the technique and optimize the storage. This is while also enhancing the speed.

Chapter 6: Self-Managed Partial Replication Technique Using Collaborative Dual Direction Download (ssCloud)

In this chapter I discuss the ssCloud (smart storage cloud) technique. The technique is an enhancement to the previously proposed methods. Here, I introduce the automation of the cloud storage concept and discuss the need to have such an automation. I then elaborate on the structure of the ssCloud technique and its implementation. I discuss simulation results, which proved the efficiency of this technique, and I compare it to other existing approaches in the research field and the industry. I finally conclude the chapter with a summary of the ssCloud technique, its benefits, and areas of enhancements.

6.1. Description of ssCloud

Here, I discuss my proposed ssCloud methodology for the cloud. The main goal is to enhance the limitation of the SPRT technique, which is the need to have a manual control over partition removals. Here, I automate the process by controlling the file partitioning starting from the upload phase. For example, when the client needs to upload a new file to the cloud, and some cloud servers do not have sufficient storage to host this file. In this case, I look for the blocks that were not downloaded from that certain server for a certain amount of time and remove them so that I can clear sufficient storage for any new files. These blocks are usually replicated on other cloud servers and can be downloaded from those servers when requested. Therefore, the effect of removing these blocks will be minimal to the download time of the file from the cloud.

To download a file, the client initiates a request to the cloud as in the previous methods. The cloud load-balancing module then checks whether the file was downloaded before, and if so, then there will be data regarding the file partitions that were downloaded and which cloud servers provided them. Having this history will help in selecting which cloud server must provide which partition. The controller finds the required data from the database and then assigns the servers, which already have the file partitions to the tasks. After the data is downloaded from all the servers, the client is updated by the required file. However, there must be a first-time download for each file to get its experience. Therefore, an alternative workflow is selected if the file is being downloaded for the first time. The file size in bytes is fetched; the block size is determined by factorizing the file size. Then, servers are assigned based on their availability and processing speeds. The database is updated at the end of every download.

To implement the storage enhancement technique, I structured my solution as described in Figure 6-1. The figure shows that there are two interfaces for each cloud. One is to manage the download requests from the clients. This includes the cloud load-balancing module. The second interface manages the file uploads and blocking processes, which is the 'FileController' of the cloud. This means that the FileController will reduce the load of client requests on the cloud load-balancing module. This is because such requests go to a different interface rather than going to the cloud load-balancing module at all. Blocking and partitioning will also be done at the FileController side. Both the cloud load-balancing module and the FileController have access to the database to make decisions. Moreover, both update the database with the results of their processes.

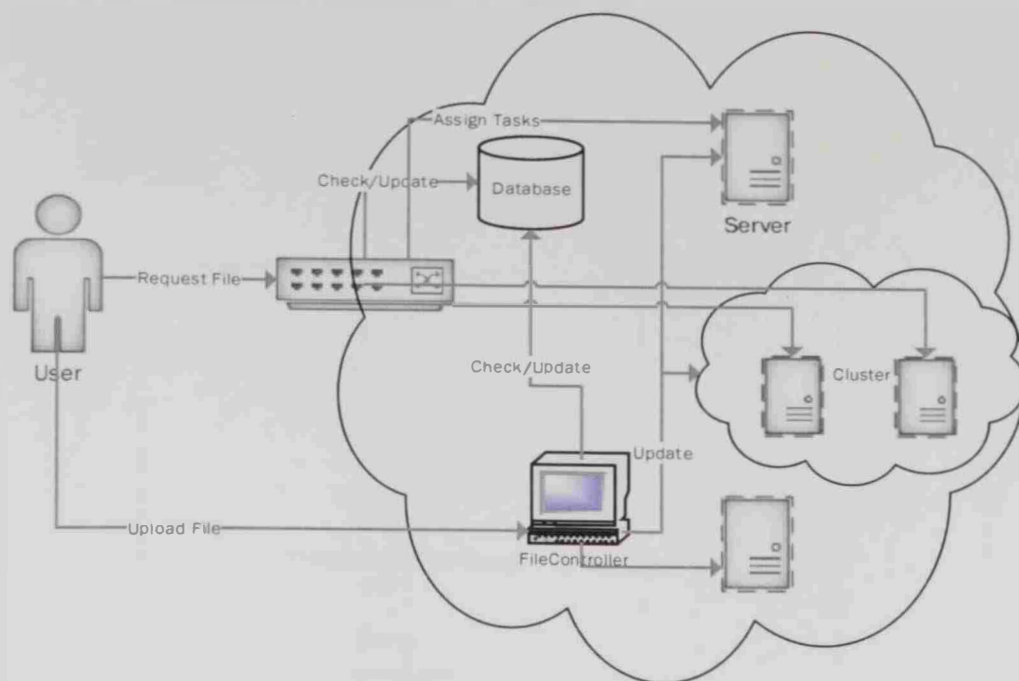


Figure 6-1: Overall Solution Structure of ssCloud.

When the client initially uploads the file, the sequence diagram shown in Figure 6-2 is used. Client, FileController, cloud server, and database are the only entities needed for this process. The client submits an upload file request to the FileController. The FileController obtains the file size from the client. Then, the FileController communicates with the servers on the cloud to identify the current available storage and to compare it to the file size to determine whether it is sufficient to upload the file. If the storage is sufficient, the FileController determines the block size, creates a directory entry with the file name in the servers, and saves the file as blocks of the block size. Finally, the FileController updates the database with each block stored in each server. In the case that the storage was not enough in any of the cloud servers, the FileController will communicate with the database to obtain all non-downloaded blocks that belong to previously downloaded files. Then, it will delete them from the server to clear storage space for the new file.

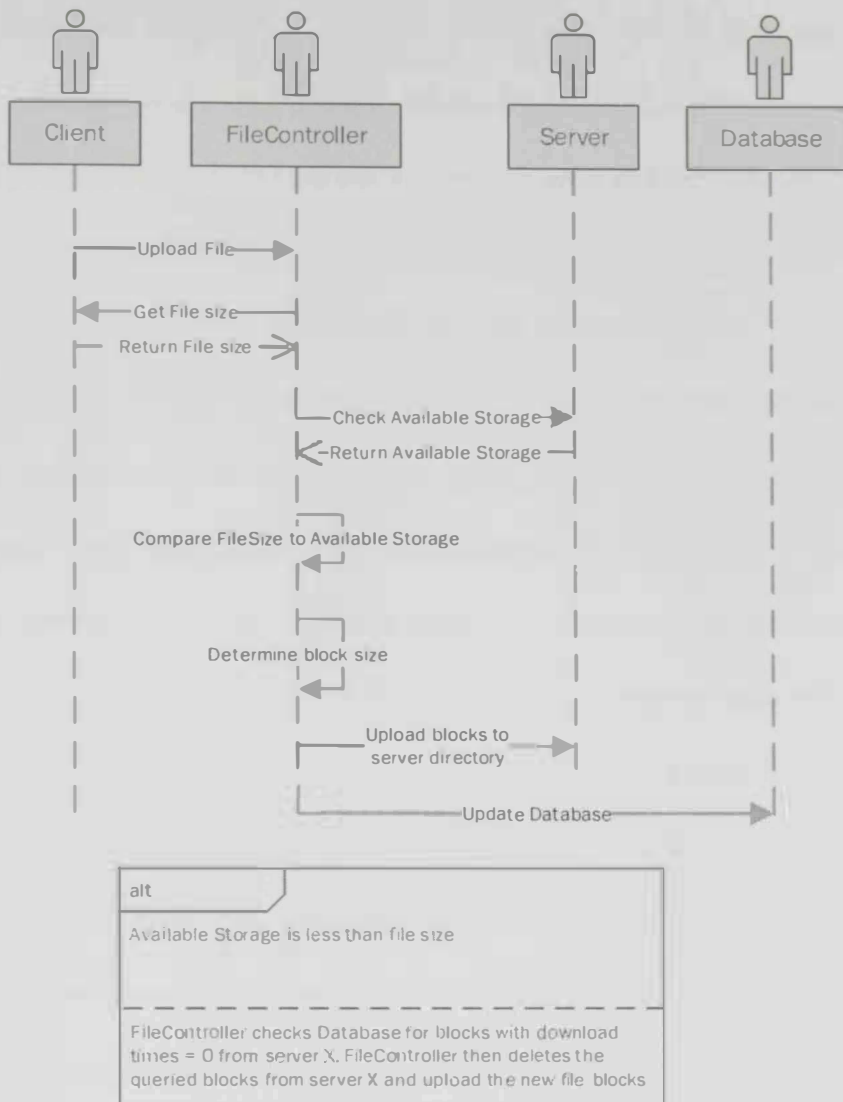


Figure 6-2: Sequence Diagram of File Upload Process.

The dynamic file upload to the cloud pseudo code for the main method of `ssCloud` is shown in Algorithm 6-1. When receiving a file upload request from the client, the method attains the file name and the file size. Then, it runs a loop through each server in the cloud and checks whether the available free storage of that server is sufficient to upload the required file on that server. If the storage is sufficient, then the file is divided into blocks determined by the factorization of the file size. A directory in the targeted cloud server is created and all the blocks of that server are copied to the destination server. In the case when the available storage of a

certain server from the cloud is not sufficient to store the file, then the method determines the required space, and checks the database for all the files that were downloaded from the cloud but the blocks were not provided from the target server. The method then removes the blocks that were never downloaded from that server and recursively uploads the file as blocks into the server directory.

This solution could be implemented in several other ways. For example, I could have implemented a batch process that runs periodically to check for non-downloaded blocks and remove them. The problem with this approach is that if I needed to upload a file before the batch process is executed, the storage may not be sufficient in the targeted server. Another method is to run the batch process after each download process by the load-balancing module. This approach will increase the load on the load-balancing module, which will have a negative effect on the download process. Therefore, I held that the most effective method is to remove blocks when an upload is requested. This allows for finding the unused blocks and removing some only when necessary for more storage. Moreover, all the additional work of storage checking, file splitting, determining block size, and saving will be done by the FileController without the need to include the load-balancing module in the process.

Algorithm 1 ssCloud Dynamic File Upload to the Cloud

Require: x_i , $FileName_i$, NOC , NOS
Ensure: $x_i > 0$, $NOC > 0$, $NOS > 0$

```

1: set  $k = 0$ 
2: set  $UpperLimit = (x/NOC) * NOS$ 
3: for  $i = 0$  to  $i = UpperLimit$  do
4:   if  $x \mid i$  then
5:      $BlockSize \leftarrow i$ 
6:   end if
7: end for
8: for all  $Server_i$  in  $NOS$  do
9:   if  $S_iStorage \geq x$  then
10:    for  $t = 0$  to  $t = x$  do
11:      transfer  $block_t$  to  $Server_i$ 
12:      create directory  $FileName$ 
13:      for  $b=0$  to Number of Block do
14:        New File  $F \leftarrow BlockID$ 
15:        writeBinary to  $F$ 
16:      end for
17:    end for
18:  else
19:    get  $Server_i$  available storage
20:    check none previously downloaded blocks
21:    while  $NotPreviouslyDownloadedBlocks > 0$  do
22:      remove  $NotPreviouslyDownloadedBlocks$ 
23:       $NotPreviouslyDownloadedBlocks = -$ 
24:    end while
25:    CheckLine 9 again
26:  end if
27: end for

```

The file structure after uploading the file to the targeted server is shown in Figure 6-3. The figure shows that the directory of the file in the server contains the file ID that was saved in the main database. Then each block is stored as a separate file using its block ID. This will make it easier for the client and the load balancer to find the blocks of the file even if the history of the file was lost or deleted by mistake. With blocks stored by their incremental ID in the file, if the database was not available, the load balancer can simply calculate the block size using the file size and look for the blocks in the cloud servers to provide them to the client.

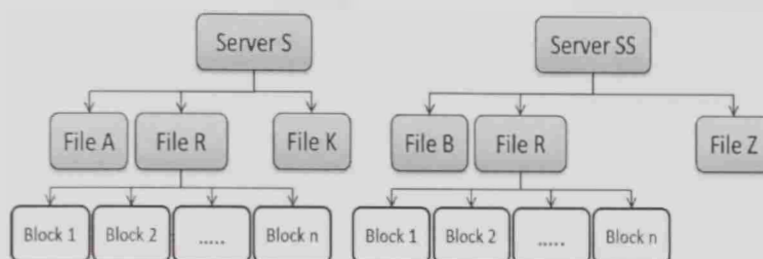


Figure 6-3: File Structure in the Cloud Servers After Initial Upload.

After more file downloads, if there was a request to upload a new file to the cloud server and the server does not have enough storage space, the non-downloaded blocks will be removed from that server. If there were blocks that were downloaded previously, the directory will remain and the previously downloaded blocks will remain in the same directory. Figure 6-4 shows the file structure in case of removing the unused blocks in order to provide more storage. In this case, Server S has a full storage space in which it will not provide blocks 1, 2, 3...100. Therefore, they were removed from its storage, and its storage space was used for the new file C, while server SS has enough storage space for file C therefore, no blocks were removed from server SS.

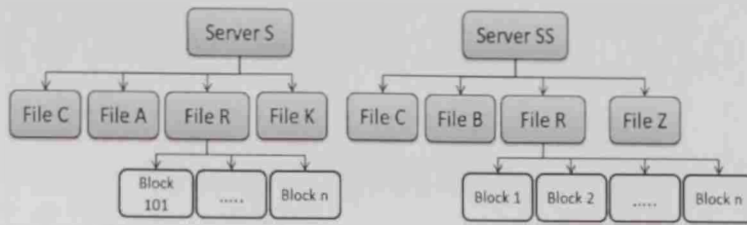


Figure 6-4: File Structure in the Cloud Servers After Unused Blocks Removal.

6.2. Example of ssCloud

In this section, I show the life cycle of a file in ssCloud to clarify how it is handled. I chose a 100 MB file in order to illustrate this example. The life cycle is as follows:

1. The 100 MB file is uploaded to the cloud using the dynamic upload file algorithm discussed in Section 6.1. Since I had eight operating servers, the number of connections allowed by my database server is 16,300 and therefore the block size of this file was found as the below.

$$\begin{aligned} \text{Max}(f(104,857,600)): \text{where } f(104,857,600) \in \left\{ 0, \left(\frac{104,857,600}{16,300} \right) * 8 \right\} \\ = 51,200 \text{ bytes} \end{aligned}$$

The number of blocks for that file will be $\frac{104,857,600}{51,200} = 2,048$ blocks each of size 51,200. Each of these blocks is saved into the controller's database separately as shown in the second row from below Figure 6-5. The tables are exported from the database.

FileID	FileName	NumOfDownloads	FileSize	BlockSize
1	file20.dat	1	20971520	10240
2	file100.dat	0	104857600	51200

Figure 6-5: Uploaded File Details in Controller's Database.

rowID	ThreadID	FileID	BlockID	DCounter	Processing
2050	127.0.0.1	file100.dat	0	0	default
2051	127.0.0.1	file100.dat	1	0	default
2052	127.0.0.1	file100.dat	2	0	default
2053	127.0.0.1	file100.dat	3	0	default
2054	127.0.0.1	file100.dat	4	0	default
2055	127.0.0.1	file100.dat	5	0	default
2056	127.0.0.1	file100.dat	6	0	default
2057	127.0.0.1	file100.dat	7	0	default
2058	127.0.0.1	file100.dat	8	0	default
2059	127.0.0.1	file100.dat	9	0	default
2060	127.0.0.1	file100.dat	10	0	default
2061	127.0.0.1	file100.dat	11	0	default
2062	127.0.0.1	file100.dat	12	0	default

Figure 6-6: Uploaded Blocks Details in Controller's Database.

Moreover, the time taken to upload this file was 20 seconds into all the servers. However, when running this example, I were not using the Internet and therefore, the time might change accordingly.

- Each block is also saved as a separate file in a folder directory with the same name as in the database. Figures 6-7 and 6-8 below show how the blocks were saved.

Name	Date modified	Type
file10.dat	T-10/-1/11-9:38...	File folder
file20.dat	T-11/13/22-9:27...	File folder
file100.dat	T-10/-2/1-1:17...	File folder
file400.dat	T-10/-1/11-9:37...	File folder
file500.dat	T-10/-1/12-9:21...	File folder

Figure 6-7: Uploaded Files Structure in Cloud Servers.

Name	Size
0.dat	50 KB
1.dat	50 KB
2.dat	50 KB
3.dat	50 KB
4.dat	50 KB
5.dat	50 KB
6.dat	50 KB
7.dat	50 KB

Figure 6-8: Blocks of the Uploaded File Saved as Separate Files in the Servers.

- When running the download, the requested file was divided into four partitions; each partition has $(2048/4) = 512$ blocks. Each set of two servers worked on their partition forward and backward till they met at certain block, and depending on the server speed, the partition was received, and they were able to help the other two servers if the other partitions were not finished. Figure 6-9 shows how the file is divided into partitions and which server is assigned to which task. An important note to mention here is that when two servers of a certain partition were very fast and finished their task before any other pair, they can join the pair in downloading their partition.

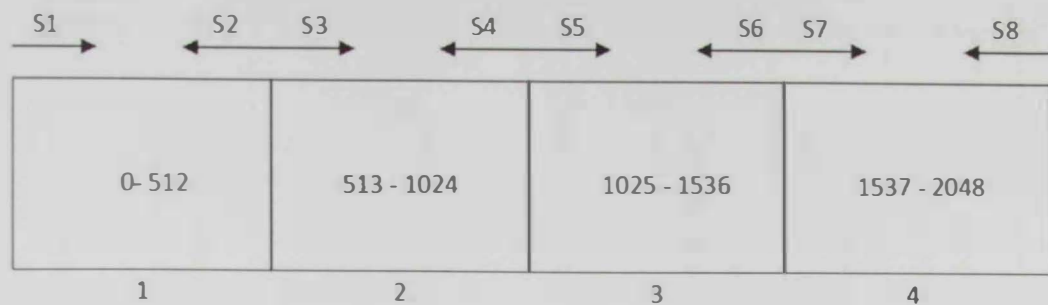


Figure 6-9: File Download Splitting and Assignment Process.

4. After running the download several times, each of the eight servers usually provided some of the blocks, although there were blocks with a download counter of zero. I stored many files on the servers so that when uploading any new file, I could see the blocks that had 'DCounter' of zero removed. I changed the network speed and server speed each time I ran the download in order to simulate a real Internet download and so that the change would affect which blocks were saved in which server. The download time whenever I ran the ssCloud changed since I changed the server speed; therefore, the number of replicated blocks changed. For example, Table 6-1 shows the each server speed and how many blocks it was provided based on its download speed.

Table 6-1: Effect of Different Speeds of Servers in Three Runs.

	First run	Second run	Third run
S1	SP(1000) bytes/ms 38 blocks	SP(1000) bytes/ms 282 blocks	SP(1500) bytes/ms 279 blocks
S2	SP(1000) bytes/ms 374 blocks	SP(1000) bytes/ms 287 blocks	SP(900) bytes/ms 276 blocks
S3	SP(2000) bytes/ms 730 blocks	SP(1000) bytes/ms 269 blocks	SP(900) bytes/ms 260 blocks
S4	SP(500) bytes/ms 187 blocks	SP(1000) bytes/ms 281 blocks	SP(1100) bytes/ms 265 blocks
S5	SP(1000) bytes/ms 374 blocks	SP(1000) bytes/ms 288 blocks	SP(2000) bytes/ms 274 blocks
S6	SP(2000) bytes/ms 295 blocks	SP(1000) bytes/ms 245 blocks	SP(500) bytes/ms 228 blocks
S7	SP(1500) bytes/ms 77 blocks	SP(1000) bytes/ms 260 blocks	SP(1000) bytes/ms 244 blocks
S8	SP(100) 8 blocks	SP(1000) bytes/ms 177 blocks	SP(1200) bytes/ms 258 blocks

5. If all servers had storage issues and needed to remove the unused blocks,

Table 6-2 shows how many blocks each server would carry. There are replicated blocks on multiple servers. However, the file is not fully replicated on the server if it removed the unused blocks based on storage needs. I noticed that of all the remaining blocks, 3107 out of 16384, there

were only 18.9%, which meant that the storage consumption was enhanced by at least 75%. Moreover, since each block was of size 51200, the entire amount of storage saved was $13,277 * 51,200 = 679,782,400$ bytes, which is equal to almost 679 MB of storage. More of the storage saving analysis will be discussed in Chapter 7.

Table 6-2: Number of Remaining Blocks Per Server After Removing Unused Blocks.

Server	Number of Blocks
S1	328
S2	433
S3	741
S4	361
S5	418
S6	303
S7	261
S8	262
Total	Number
removed blocks	$(8 \times 2048) - 3107$ $= 13277$ blocks

6.3. Analysis and Simulation Results of ssCloud

In this section I analyze the differences between ssCloud and other storage optimization approaches for the cloud. I also discuss the evaluation of ssCloud.

When comparing ssCloud to RRNS (discussed in Chapter 3), RRNS retains the file fragmentation details on the client side. This is beneficial as a security measure allowing for higher safety controls for the client. However, if the client loses the file information, a serious issue would occur because no one else has the same information. This is not an issue with ssCloud since there are backups of the database. Even if there is an issue with the database, the blocks and files are stored

in each server by a sequence ID. This means that they are reachable, but the load-balancing module will have to expend more effort to obtain the information. BerryStore, on the other hand, does not take security as a priority. Its target is to provide a fast method to download small files by storing multiple small files as one large file. The problem here is that it cannot be applied to larger files, while ssCloud and RRNS both can handle files of any size. Table 6-3 shows the comparison between the three approaches.

Table 6-3: Comparison of Storage Optimization Techniques.

	SOF	Security	File Types	Replication	Client effort
RRNS	Yes	High	All	Partial	High
Berry- Store	No	Moderate	<10 MB	Full	Moderate
ssCloud	No	Moderate	All	Partial	Low

To know the probability of deleting a certain block from a given server, I use a conditional probability because there are three events that must happen before deleting a block from a server. First, a file upload request must be initiated. Then, the server storage must be insufficient. Finally, the block must not have been previously downloaded from the server for a previous download request. Figure 6-10 illustrates the probability of deleting a block from a server. In the figure, $P(A)$ is the probability of uploading a new file; $P(B)$ is the probability of insufficient storage in server X, and $P(C)$ is the probability that block z was never downloaded before, while file R containing that block was downloaded several times. The highlighted intersection

between $P(A)$, $P(B)$, and $P(C)$ signifies having all these events occurring simultaneously. That is, if $P(A \& B \& C)$ then the block will be deleted. Notice that the probability of removing a certain block using ssCloud is low compared to the normal flow. This means that in most cases, there will be a file upload request but cloud servers will have sufficient storage available or the block will be downloaded previously from the server, and it will not be removed.

$$P\{A \cap B \cap C\} = P\{A\} \cdot P\{B|A\} \cdot P\{C|A \cap B\}$$

$$OR = \frac{1}{n!} = \frac{1}{3!} \quad (4)$$

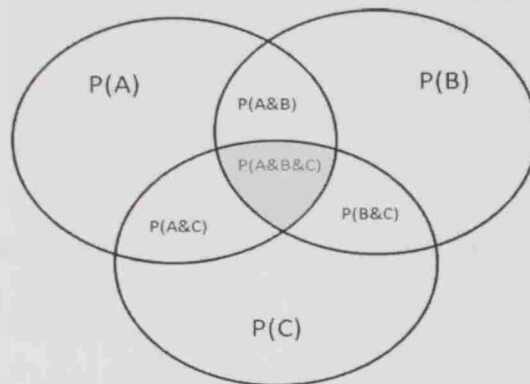


Figure 6-10: Probability of Removing a Block.

On the other hand, the probability that a partition is downloaded from a certain server is a dependent probability. If I have a file of two partitions and four servers will be working on providing these partitions, then the probability that server 1 provides a forward download from the file is as follows:

$$P(S_i \text{ downloads } P_j) = \frac{1}{S} \quad , \quad (5)$$

Where S is the number of available servers and P is the number of partitions. This means that the probability that server 1 provides partition 1 forward is $1/(4/2) = 1/2$. When trying to determine the probability that server 1 provides the download of partition 2, then it will be $1/(3/1) = 1/3$ as the number of servers will decrease because server 1 will be busy providing partition 1, and there is only one partition remaining. This analysis is important to know which blocks will be downloaded by which server. If a block is regularly provided by a server, then it will not be removed.

Storing the file for the first time in my static storage optimization was done by saving the full file then taking copies of the blocks and deleting the original file. However, splitting the file from the beginning as blocks when the file is transmitted from the client to the cloud servers enhances this. This will also be of minimal effect to the client download process. This is because the client will be downloading the blocks within a file (as shown in Figure 6-11). When simulating both cases, I noticed that the download time difference between the two is negligible. Downloading different blocks will increase the download time because databases must be checked for previous experience for that file. Moreover, each block file must be opened and downloaded. I noticed that the maximum difference between downloading the full file without database access and downloading partitions with database access was 8 seconds when the file size was 2 GB. The average overhead of the download process is about 3%. However, using multiple dual-direction servers still improves the performance compared to regular approaches. Furthermore, when optimizing the storage space of the cloud servers, this is a very minimal difference.

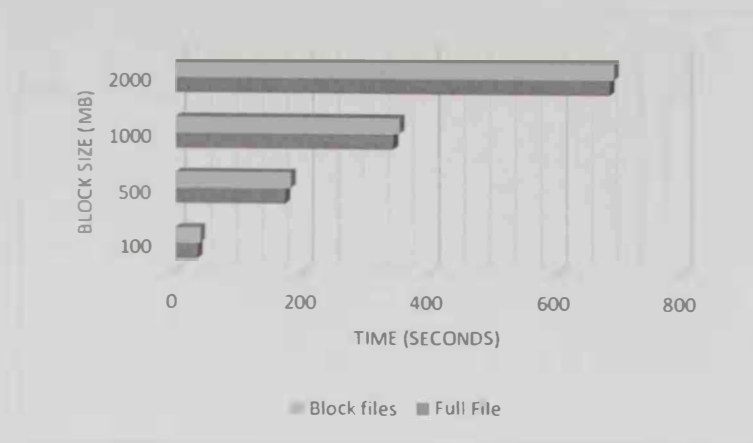


Figure 6-11: Time Difference in Download for Different File Sizes.

Moreover, there is an effect to the need to connect to the database in the upload process too. This is since all the upload requests must go through the controller and be partitioned into blocks as in equation 3, chapter 5. This process increases the time of upload for the files. However, since the file is uploaded once and downloaded several times, my concern was to minimize the database connection effect on the download process. Figure 6 -12 shows a comparison of uploading a file using ssCloud to uploading a full file without partitioning and database connection. I noticed that as the file size increases the difference between the two methods decreases. This is because the number of blocks is determined based on the number of servers, number of database connections and file size. By using equation 3, the number of blocks will decrease as the file size increases and therefore the number of connection requests to the database will decrease too. As a result, this decreases the difference between the process of uploading a file without the need to a database connection and the process used in ssCloud.

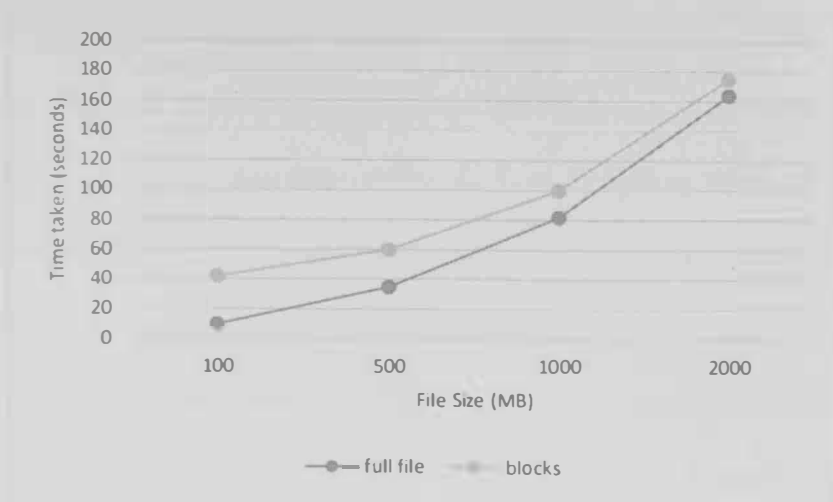


Figure 6 -12: Comparison of Full File Upload and Blocks Upload in Terms of Time Taken

A hypothesis was made that the storage consumption in the server when storing a full file would be less than storing multiple distributed blocks of the same file. That is because most of the researchers assumed that each block of the file would require additional space to store headers and file types. Therefore, I simulated the difference between the two options using my approach and found that the overall size of the original file and the folder containing all the split blocks of the file are exactly the same as shown in Figure 6-13. The result was the same because I stored the file as a number of binary files and the resulting downloaded file is of the same format as the original file.

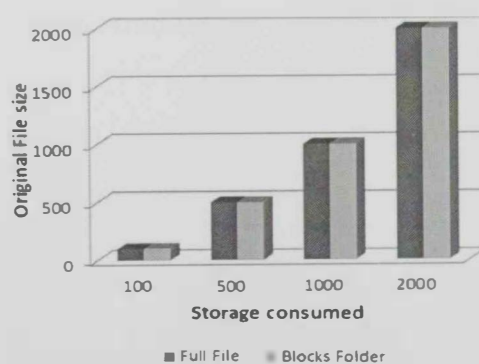


Figure 6-13: Consumed Storage Difference in MB.

Figure 6-14 demonstrates a comparison between ssCloud, RRNS, Dropbox, and Google Drive in terms of the download time for files of different sizes. The sizes of files used for the comparison ranged from 10 MB to 1000 MB (1 GB). I set the download speed as the Internet speed in my network which was 1 Gbps. The figure shows that the 10 MB file was downloaded by RRNS in 30 seconds while the ssCloud downloaded the same file in 11 seconds, Dropbox took 20 seconds, and finally Google Drive took almost one minute. Moreover, a 400 MB file was downloaded by RRNS in 640 seconds, while the ssCloud downloaded it in 525 seconds. This is mainly a result of multiple servers working collaboratively on each partition of the download. Dropbox provided the file in 750 seconds and Google Drive in 660 seconds. The results demonstrate that Google Drive performs better with medium file sizes (100-500); however, when the file size reaches 1 GB both Google Drive and Dropbox need more than 20 minutes to download. The RRNS performs better than Dropbox and Google Drive because it assigns tasks to multiple servers; however, each cloud server is solely responsible for providing its partition; therefore, the delay in any of the servers' performances will affect the entire download process. Although, RRNS performs better than many of the existing load-balancing strategies that assign the full download to one server. However, its performance can be enhanced by the dual-direction approach used by ssCloud.

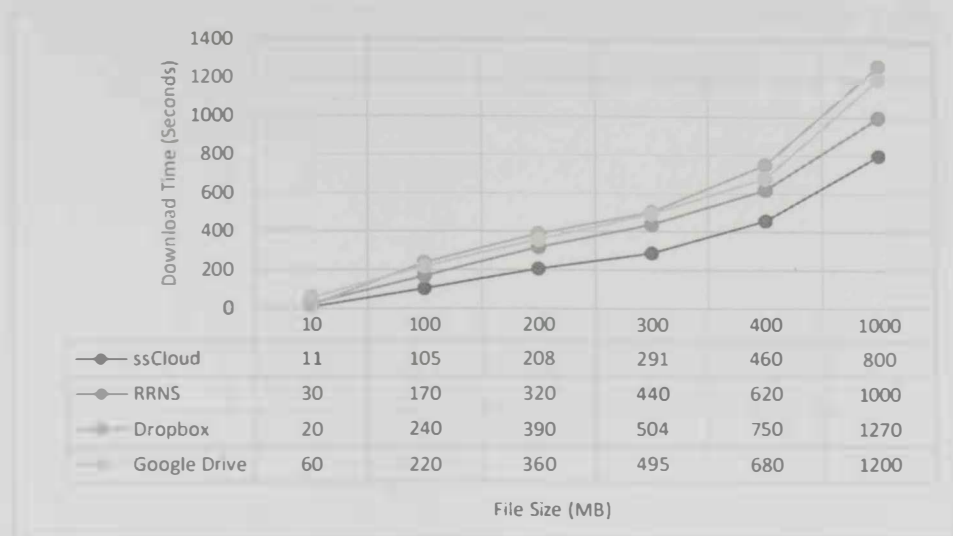


Figure 6-14: Download Performance Comparison.

Dropbox, Google Drive, RRNS, and ssCloud all have an upload phase where the file is uploaded into the cloud [78]. The number of partitions and replicas are then determined. I compared the upload of files of the same sizes using RRNS and ssCloud and compared them to Google Drive and Dropbox using an Internet speed of 1 Gbps for each.

Figure 6-15 illustrates the difference between the approaches. The figure shows that RRNS performed better when the file size was relatively small (10 MB). The file was uploaded in 30 seconds using RRNS, while it was uploaded in 50 seconds using ssCloud. However, as the file size increases, the performance of ssCloud improves and outperforms RRNS in all trials. A file of size 400 MB was uploaded in 120 seconds using RRNS and in 79 seconds using ssCloud. Dropbox usually redirects many of its tasks to Amazon EC2 for processing, and that takes more time to process tasks compared to the other approaches in both the upload and download processes [79].

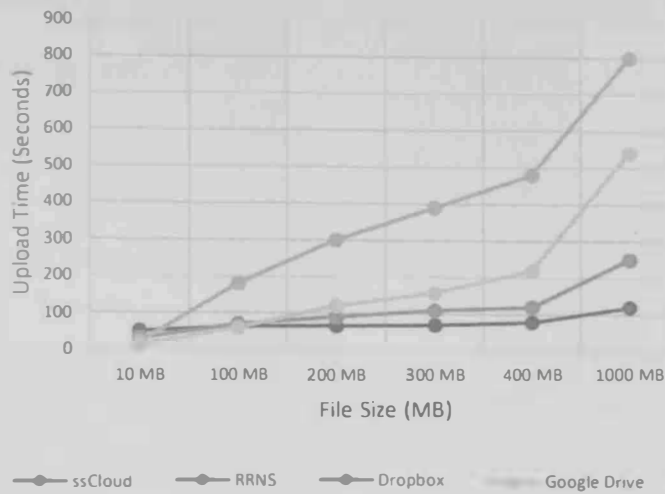


Figure 6-15: Upload Performance Comparison.

The issue of the number of connections will mostly appear at the database server side. This is because a large number of connections could affect any database server, which could result in inefficient performance. This is why I ensure that the blocks are large enough to reduce the number of communications with the database server whenever a block is added to the file. The approach is to split the file into multiple blocks and then save them as separate files in the target folder on the hosting server. It will also save each block record in the database in order to target any action taken regarding the block, such as download or delete. Figure 6-16 shows the error rate in a case where the number of connections of the database server was not considered. In the case where the number of connections was not considered, the database server will generally crash at some point. It usually recovers and saves the rest of the blocks, but I noticed that it has not saved all the correct rows. I also noticed that as the file size increases, the error rate between the actual rows saved and the real value that should have been saved increases. To solve this problem, I considered the number of hosting servers (NOS) and the

number of available database server connections (NOC) when calculating the block size of the target file.

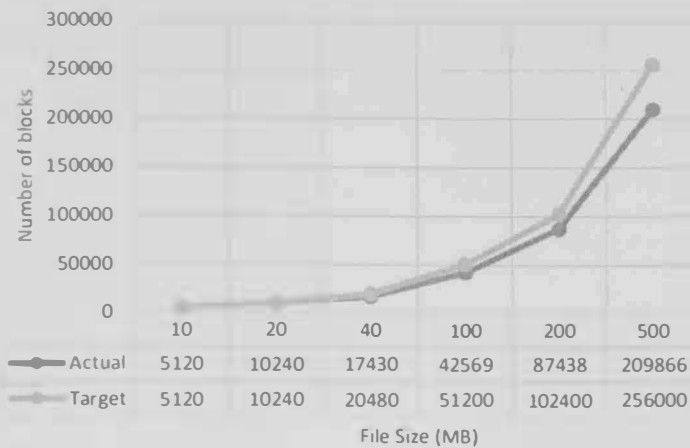


Figure 6-16: Error Rate Caused by the Database Server in the Case of an Exceeding Number of Connections.

When applying my approach for creating a number of blocks that are associated with the DB server connections and the number of hosting servers, I have noticed that as the number of servers increased, the block size also increased and the number of blocks decreased. This is because I want to reduce the number of blocks saved in the database every time there is an upload request. Therefore, clients will not face any failure in the cloud DB server. Table 6-4 displays the results obtained when applying my approach, knowing that when applying this approach the error rate was zero.

Table 6-4: Experimental Relationship Between NOS, NOC, Block Size, and NOB.

File Size	NOS	NOC	Block Size	NOB
524,288,000	4	16,384	128,000	4096
209,715,200	4	16,384	51,200	4096
52,428,800	4	16,384	12,800	4096
524,288,000	6	16,384	256,000	2048
629,145,600	6	16,384	307,200	2048
524,288,000	6	10,000	409,600	1280
629,145,600	6	10,000	491,520	1280

Moreover, I have tested the effect of block size over the download time in the case where the block size was not restricted by the number of connections available with the database server. I have changed the block size among values by 1 KB, which is the minimum size of a block to file size divided by two. As I am using a dual-direction download, the maximum block size without replication should be half of the file size. Results shown in Figures 6-17, 6-18, 6-19, and 6-20 demonstrate that there is an optimal block size for each file, and this optimal block size depends on the file size itself and the number of collaborated servers providing this file. Usually the optimal block size starts from 100 KB–1000 KB for a file provided by two servers, and as the file size increases, the optimal block size changes accordingly. The difference between the optimal block size and any other block size (as I increase) is minimal for small files (10 MB), but as the file size increases, the difference in the performance increases. Therefore, the effect is clear. This emphasizes the importance of choosing the optimal block size when uploading the file to the cloud.

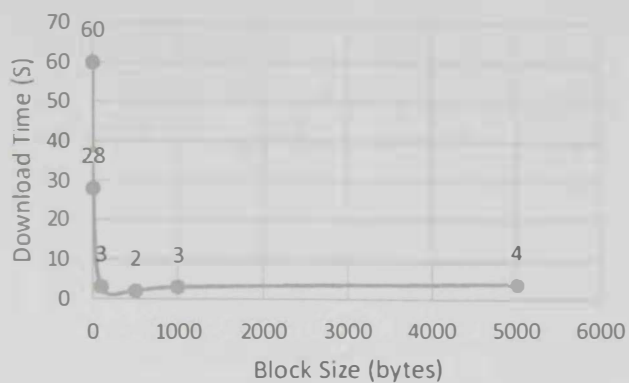


Figure 6-17: Block Size Effect on Download Time for 10 MB File Using Two Servers.

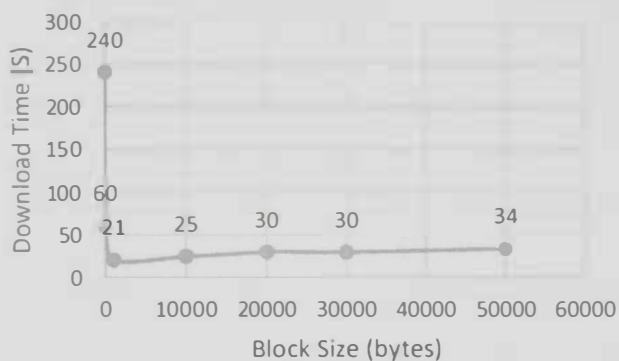


Figure 6-18: Block Size Effect on Download Time for 100 MB File Using Two Servers.

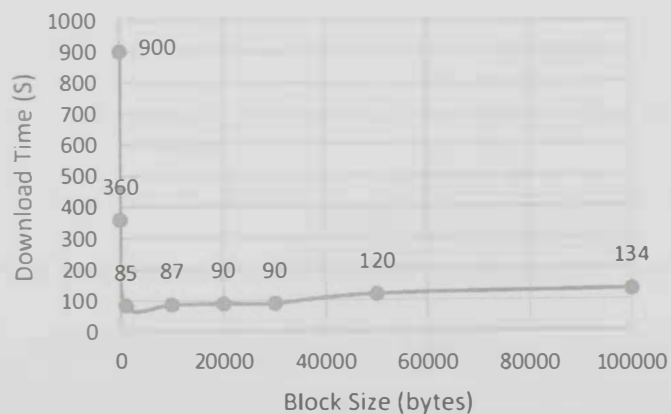


Figure 6-19: Block Size Effect on Download Time for 400 MB File Using Two Servers.

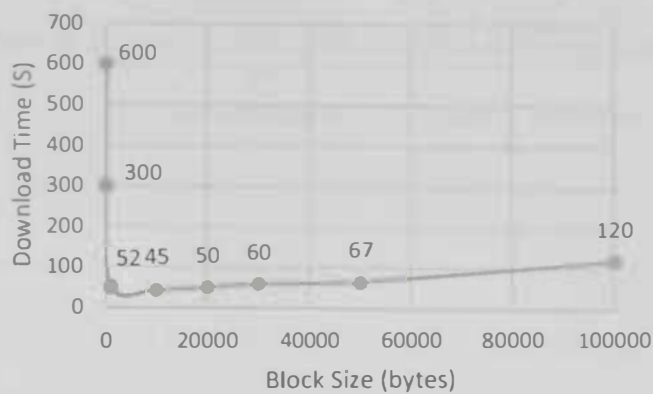


Figure 6-20: Block Size Effect on Download Time for 400 MB File Using Four Servers.

6.4. Enhancements and Limitations of ssCloud

The main importance of ssCloud is the combination of the dual-direction download approach and the autonomic management of the storage resources in the cloud. There is a clear benefit that the download time is tremendously decreased as well as the cost of storing the file whenever necessary. The ssCloud is safe to remove the unused blocks, as this will not affect the download time and therefore will not negatively affect the end users (cloud clients). Moreover, the ssCloud overcomes most of the challenges facing load balancing and storage optimization in the cloud, such as server failure. In the case of a server failure, another contributing server can replace the failing server. As long as this server provides blocks even minimally, then the blocks will not be removed from that server, which confirms the reliability of this method. I think that more analytics on the optimum block size to further enhance the download speed could further improve the ssCloud.

6.5. Conclusion

The design of the ssCloud aims to improve the download time from the cloud and optimize the storage allocation techniques to enhance the cloud DaaS. Load balancing is improved using a collaborative dual-direction download method to partition files and assign partitions to multiple cloud servers. Smart storage allocation is accomplished by automating the file upload process to check for available storage on each server and remove non-downloaded blocks based on previous experiences. The technique's analysis shows that my algorithm has a better opportunity of optimizing cloud storage. In addition, I calculated the probability of removing unused blocks and found it to be very low. However, the choice of deletion is available when needed. Using the ssCloud helps reduce the time needed to download a file and the storage cost needed to host millions of files in the cloud.

Chapter 7: Performance Analysis

In this chapter I develop an analytical model in order to estimate storage saved using my partial replication approach and the amount of time needed to download the files using this technique. I validate the estimations by simulation and provide the results. I then discuss my observations and provide methods of enhancing the ssCloud even more. Finally, I discuss the conclusions.

7.1. Expected Storage Saved Estimation

In this section, I develop an analytical model to estimate the storage saved through a mathematical analysis. In order to explain the storage saved by ssCloud, I investigate an example of two collaborative servers working on a 1000-blocks file. I review the case where the maximum number of blocks downloaded by server 1 was 700 blocks, as shown in Figure 7-1, and the maximum number of blocks downloaded by server 2 was 500 blocks. This means that server 1 (even at its best performance) never downloaded the 300 remaining blocks. Moreover, server 2 never downloaded the 500 remaining blocks. These blocks will be removed by my approach. On the other hand, there are 200 blocks that are commonly downloaded by one of the two servers at different download times. These blocks are the only blocks that will be replicated in both servers at the end.

In order to estimate the number of replicated blocks, I summed the maximum blocks downloaded by both servers and took the file total number of blocks out.

$$Est(Rep) = (700 + 500) - 1000$$

$$= 200 \text{ Blocks}$$

This indicates that the total number of saved blocks is 1200 blocks with a partial replication. However, a full replication technique would need to store 1000 blocks on both of the servers, which would be 2000 blocks. By removing the unused blocks, I saved 800 blocks of storage.

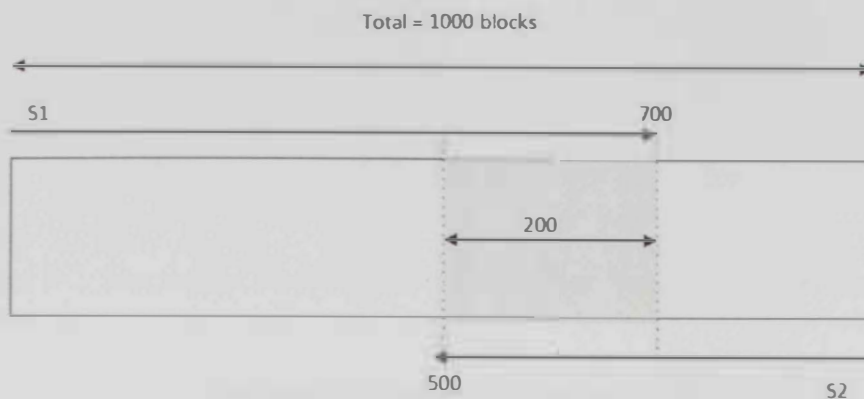


Figure 7-1: Number of Replicated Blocks in Two Servers for 1000 Block File.

In the same way, I can find the replicated blocks among four servers. An example of the case where the same file of 1000 blocks were downloaded by four servers and the maximum number of blocks for each server is below:

- Server 1: 300 blocks
- Server 2: 300 blocks
- Server 3: 400 blocks
- Server 4: 200 blocks

Figure 7-2 shows the replication among the four servers. I can see that there are 100 blocks replicated between server 1 and 2, and 100 other blocks replicated between server 3 and 4. The sum of all the replicated blocks among all partitions is as follows:

$$Est(Rep) = (300 + 300 + 400 + 200) - 1000$$

= 200 Blocks

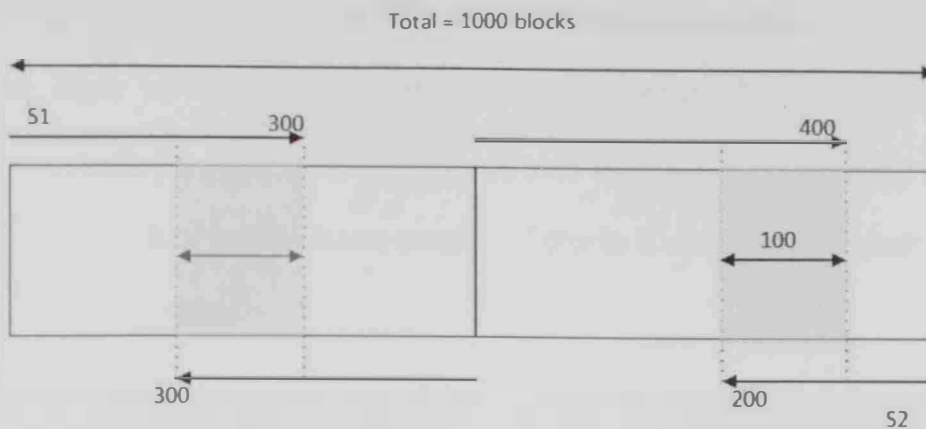


Figure 7-2: Number of Replicated Blocks in Four Servers for 1000 Blocks File.

Therefore, the equation to calculate the number of replicated blocks of file i in any collaborative servers S after an experience is as in Equation 6.

$$Exp(Rep)_i = \left\{ \sum_{j=1}^S Max(BlocksProvided)_j \right\} - TotalBlocks_i \quad (6)$$

Where $\sum_{j=1}^S Max(BlocksProvided)_j$ is the sum of all the maximum number of blocks provided by each server of the collaborative servers and $TotalBlocks_i$ is the total number of blocks in file i .

In order to know the maximum number of blocks that will be provided by a certain server, I need to know the maximum and minimum speeds of each of the collaborative servers. For this example I have four servers of minimum and maximum speeds as follows below:

- Server 1: Min = 15 blocks/s, Max: 20 blocks/s
- Server 2: Min = 5 blocks/s, Max: 12 blocks/s
- Server 3: Min = 6 blocks/s, Max: 10 blocks/s

- Server 4: Min = 8 blocks/s, Max: 15 blocks/s

To estimate the maximum number of blocks that will be downloaded by server 1, I allow it to download at its maximum speed (20 blocks/second) by setting all the other servers' speed to the minimum speeds (five, six, and eight blocks/second). By doing this, server 1 will have to download most of the blocks in the file, which is the maximum number of blocks it can provide. Equation 7 is used for the estimation.

$$\begin{aligned}
 &MaxBlocks(S_i) \\
 &= \frac{Max(\lambda(S_i))}{Max(\lambda(S_i)) + \sum_{j=2}^X Min(\lambda(S_j))} \times TotalBlocks(File) \quad (7)
 \end{aligned}$$

Where $MaxBlocks(S_i)$ is the maximum number of blocks provided by Server i . $Max(\lambda(S_i))$ is the maximum speed of Server i . and $\sum_{j=2}^X Min(\lambda(S_j))$ is the minimum speed of all other collaborative servers downloading the file.

When applying Equation 7 to my example, the maximum number of blocks provided by server 1 is $\frac{20}{20+(5+6+8)} \times 1000 = 512$ blocks. It is important to mention here that I either use speed units of bytes/ms or blocks/s since each block in my approach is found by Equation 3 in terms of bytes. Therefore, any of the two units can be used to estimate the maximum number of blocks provided and the number of replicated blocks. Equation 8 is used to convert the speed from bytes/ms unit to blocks/s unit.

$$\lambda(blocks/s) \approx \frac{\lambda(bytes/ms)}{BlockSize_i} \times 1000 \quad (8)$$

Where $\lambda(\text{blocks}/s)$ the speed in blocks/second is. $\lambda(\text{bytes}/ms)$ is the speed in terms of bytes/milliseconds and $BlockSize_i$ is the block size of file i .

The below experience demonstrates how Equations 6 and 7 are useful for estimating the number of replicated blocks and the saved storage. In order to validate this, I used a file of size 20 MB, and two servers to download it. The file has 2048 blocks of size 10,240 bytes each. Below are the minimum and maximum speeds of both servers.

- Server 1: Min = 600 bytes/ms, Max = 1500 bytes/ms (Min = 58 blocks/s, Max = 146 blocks/s).
- Server 2: Min: 100 bytes/ms, Max = 1000 bytes/ms (Min = 10 blocks/s, Max = 97 blocks/s).

I ran the download for the 20 MB file using the above two servers after setting the speed for server 2 to nine blocks/s. Server 1 automatically performed at its max speed (146 blocks/s). Using Equation 10, $MaxBlocks(Server_1) = \frac{146}{146+10} \times 2048 = 1917$ blocks. The result I obtained from running the experience was that server 1 provided 1921 blocks and server 2 provided 128 blocks, which is very close to the estimated number by using Equation 10.

I conducted another experiment by setting the speed of server 1 to 48 blocks/s so that server 2 was forced to download the maximum number of blocks it could afford. Using Equation 7 $MaxBlocks(Server_2) = \frac{97}{97+58} \times 2048 = 1281$ blocks. The results I obtained from running my method was that Server 1 provided 768 blocks and server 2 provided 1281 blocks which confirms that the equation is correct when using two servers to provide one file.

Moreover, when using Equation 6, the expected number of replications for this run is found by the following equation:

$$\begin{aligned} Est(Rep) &= (1921 + 1281) - 2048 \\ &= 1154 \text{ Blocks} \end{aligned}$$

When testing this using my approach, it is exactly equal to the result above.

The replicated blocks IDs belonged to the real numbers in $\in \{767, 768, \dots, 1921\}$.

I applied this equation to a situation of four servers too. The servers' speed were are follows:

- Server 1: Min = 59 blocks/s, Max = 146 blocks/s.
- Server 2: Min = 10 blocks/s, Max = 97 blocks/s.
- Server 3: Min = 20 blocks/s, Max = 100 blocks/s
- Server 4: Min = 10 blocks/s, Max = 80 blocks/s.

I ran my method using the four above servers four times so that each server could perform at its maximum for one iteration. The results of the maximum blocks for each server against the one expected using Equation 7 are found in Table 7-1. The equation was at least 98.7% correct, and the difference between the expected and actual was at most only 19 blocks, which is the server 1 result.

Table 7-1: Evaluation of the Accuracy Equation 7.

	Expected Max blocks	Real Max blocks	Correctness
Server 1	$= \frac{146}{146 + (10 + 20 + 10)} \times 2048$ $= 1607$	1595	99.2%
Server 2	$= \frac{97}{97 + (59 + 20 + 10)} \times 2048$ $= 1068$	1061	99.3%
Server 3	$= \frac{100}{100 + (59 + 10 + 10)} \times 2048$ $= 1144$	1130	98.7%
Server 4	$= \frac{80}{80 + (59 + 10 + 20)} \times 2048$ $= 969$	969	100%

As for the number of replicated blocks over the four servers for the 20 MB file tested above, it is equal to $(1588 + 1061 + 1130 + 969) - 2048 = 2700$ blocks.

7.2. Expected Download Time Estimation

In order to estimate the expected download time ($Exp(DT)$) of block i using the ssCloud, I must know the attributes, such as the percentage that block i was downloaded from each server and the time taken by each Server in order to download that block. Then, the expected download time of block i is the sum of the percentage that block i was downloaded from server k multiplied by the time taken by server k to provide block i . A simple example to explain this equation is

the time taken to travel from one place to another several times. If a person, who usually uses two methods of travel between two cities, such as by car and plane, would like to estimate the expected travel time between cities, and it previously took one hour to travel by plane (percentage of using plane is 90%) and three hours to travel by car (percentage of using the car is 10%), then the estimated travel time is $(0.10 \times 3 + 0.90 \times 1)$. The same applies to ssCloud since there are different possibilities that a block could be downloaded from any server.

$$Exp(DT_i) = \sum_{k=1}^S (P_i^k \times \lambda_i^k) \quad (9)$$

Where $Exp(DT_i)$ is the expected download time spent to provide block i using ssCloud. S is the number of Servers providing block i . P_i^k is the percentage server k provides block i , and λ_i^k is the speed by which server k provided block i .

The total download time of file F is equal to the sum of the expected download time of all the blocks h in File F as shown in Equation 10.

$$Exp(DT) = \sum_{i=1}^b Exp(DT_i) \quad (10)$$

Where $Exp(DT)$ is the overall expected download time, and h is the number of blocks in the file.

This will also result in summing all the download time for each block (sum of percentage a block was downloaded from a certain server multiplied by the download time of that server) as in Equation 11.

$$Exp(DT) = \sum_{i=1}^b \sum_{k=1}^S (P_i^k \times \lambda_i^k) \quad (11)$$

When using my partial replication dual-direction technique to verify the above-mentioned equations, first, I set a very simple experiment to begin. I uploaded a 20 MB file size, with 2048 blocks, each of size 10,240 bytes. I set this download test to operate using only two servers. I ran the experiment ten iterations and changed the servers' speed each time so that the download percentage of the block from a server is affected (as shown in Table 7-2). For the eleventh time, the speed of server 1 was at 50 blocks/s and the speed of server 2 was 70 blocks/s. I estimated the download time for each block by Equation 6 and Table 7-3 depicts the estimated download time versus the actual download time. I selected blocks 1, 2048, 1024, 500 and 1500 to be the blocks on which I compare the accuracy of Equation 8 because they represent the edges and elements of the groups. For example, block 1 will always be downloaded from server 1 and block 2048 will always be downloaded from server 2. Therefore, it is easy to predict the expected download time for such blocks, and it will be accurate, as they have the same experience every time a download is completed. However, this is not the case for blocks similar to block 500 and block 1500. This is because there is a small percentage of time that they will be provided by a different server than the regular server that usually provides them. For example, server 1 usually provides block 500, but there are two times when server 1 was slow or loaded when server 2 had to provide this block. The Equation 6 prediction in these cases was very efficient since the accuracy percentage was not below 90. The worst case is the point where the two servers usually meet. An example of this case is block 1024. When downloading block 1024, it could be downloaded by any of the servers each time. This will have the least accuracy in my case, but the error rate was 12%. I consider a maximum difference of 12% to be within the acceptable rate because the

download times of each server differs according to the network speed, and this is very unpredictable behavior.

Table 7-2: Dual Server Experience in Ten Runs.

Run	S1 Speed (blocks/s)	S2 Speed (blocks/s)	S1 blocks	S2 blocks
1	100	100	1-1026	2048-1024
2	100	20	1-1709	2048-1708
3	20	100	1-340	2048-339
4	120	100	1-1118	2048-1116
5	80	10	1-1823	2048-1822
6	10	80	1-227	2048-225
7	40	120	1-512	2048-511
8	90	70	1-1153	2048-1150
9	60	50	1-1118	2048-1117
10	120	130	1-984	2048-982

Table 7-3: Equation 9 Accuracy Evaluation.

	<i>Exp</i> (DT) in Seconds	<i>Actual</i> (DT) in Seconds	Correctness
Block 1	$= \left\{ \left(100\% \times \frac{1}{50} \right) + \left(0\% \times \frac{1}{70} \right) \right\} = 0.02$	0.02	100%
Block 2048	$= \left\{ \left(0\% \times \frac{1}{50} \right) + \left(100\% \times \frac{1}{70} \right) \right\} = 0.014$	0.014	100%
Block 1024	$= \left\{ \left(54.54\% \times \frac{1}{50} \right) + \left(45.45\% \times \frac{1}{70} \right) \right\} = 0.016$	0.014	88%
Block 500	$= \left\{ \left(80\% \times \frac{1}{50} \right) + \left(20\% \times \frac{1}{70} \right) \right\} = 0.018$	0.02	90%
Block 1500	$= \left\{ \left(30\% \times \frac{1}{50} \right) + \left(70\% \times \frac{1}{70} \right) \right\} = 0.015$	0.014	93%

7.3. Discussion and Observations

In this section, I discuss different performance and storage observations obtained during the evaluation of my approach. One observation I made was that as the sum of speeds of the dual servers increases, the overall performance increases as well and therefore the download time decreases. Table 7-4 shows the experiment I ran to validate this assumption. I carried 5 runs each with different speeds of each servers and different sums of speeds. The best performance of this run was 8 seconds download when both servers were fast and the sum of speeds was 3000 bytes per second. The difference between the speeds of the servers does not have much effect to the download time because the dual servers work in opposite directions and they meet at a certain point.

Table 7-4: Speed Difference Between the Dual Servers, Affecting Download Time.

	S1 Speed (bytes/ms)	S2 Speed (bytes/ms)	Download time (S)	Speed Difference (bytes/ms)	Sum of speeds (bytes/ms)
1	500	500	21 s	0	1000
2	500	600	19 s	100	1100
3	500	1000	17 s	500	1500
4	1000	2000	8 s	1000	3000
5	1000	1000	11 s	0	2000

Another observation was also the effect of the difference between the minimum and maximum speeds of any server on the number of replicated blocks between the two servers. This depends on the file size. Therefore, I tried a file of

size 400 MB to validate this assumption. For example, if I have two servers as follows:

- Server 1: Min = 20 blocks/s, Max = 100 blocks/s. The difference between the Min and Max is 80 blocks/s.
- Server 2: Min: 50 blocks/s, Max = 150 blocks/s. The difference between the Min and Max is 100 blocks/s.

When I use Equation 7 to discover the maximum number of blocks that can be provided by any of the above-mentioned servers, I found the results below:

- Server 1 Maximum blocks = $\left(\frac{100}{100+50}\right) \times 4098 = 2732 \text{ blocks}$
- Server 2 Maximum blocks = $\left(\frac{150}{150+20}\right) \times 4098 = 3615 \text{ blocks}$

From these results, the maximum number of replicated blocks would be $(2732 + 3615) - 4098 = 2249$. If using the other two servers, there would be less difference between the minimum and maximum speeds and the results would change, for example:

- Server 1: Min = 50 blocks/s, Max = 70 blocks/s. The difference between the Min and Max is 20 blocks/s.
- Server 2: Min: 60 blocks/s, Max = 80 blocks/s. The difference between the Min and Max is 20 blocks/s.

The maximum number of blocks that could be provided by any of the two servers is shown below.

- Server 1 maximum blocks = $\left(\frac{70}{70+60}\right) \times 4098 = 2026 \text{ blocks}$.
- Server 2 maximum blocks = $\left(\frac{80}{80+50}\right) \times 4098 = 2185 \text{ blocks}$.

The number of replicated blocks would be only equal to $(2026 + 2185) - 4098 = 113 \text{ blocks}$. This means that, as the difference between the maximum and

minimum of the dual servers decreases, the number of replicated blocks will also decrease. This would be very useful in terms of saving the storage used for the replicated blocks, since this storage can be used for other large files.

Figure 7-3 shows relationship between the maximum number of replicated blocks with the min-max gap in servers' performances tested in my validation of the previously mentioned observation. The validation was completed for a 400 MB file size of 4098 blocks. The relationship is extrusive, as the gap increases, the number of replicated blocks also increases.

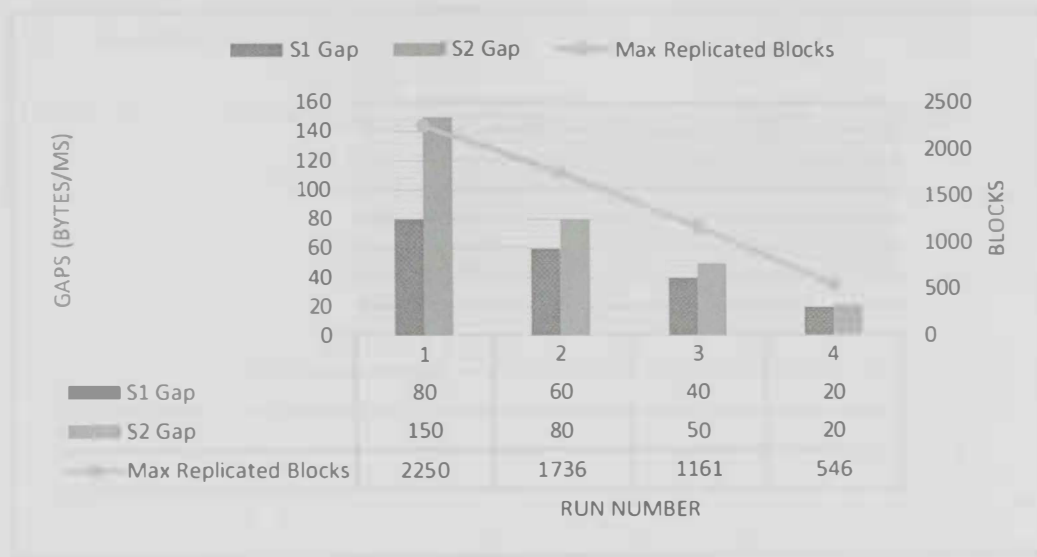


Figure 7-3: Experimental Relationship Between Min-Max Speed Gap and Maximum Number of Replicated Blocks for 100 MB File Size.

To evaluate the storage enhancement of ssCloud compared to the original CDDL technique [80][81][82], I estimate the storage enhancement of a 524,288,000 bytes (524 MB) file replicated on 4 servers. If I use the original CDDL technique, then a full replication of the file is needed across the 4 servers despite the maximum and minimum number of blocks that can be provided by any of the servers. Therefore, the final storage consumption of the file would be as equation 12.

$$Storage_{CDDL B} = M \times R \quad (12)$$

Where M is the number of servers and R is the file size. This means that the storage required by CDDLB for the above example would be $4 \times 524,288,000 = 2,097,152,000$ bytes (around 2 GB). On the other hand, if I use the ssCloud technique, and servers maximum and minimum blocks were as follows:

- Server 1: Min = 59 blocks/s, Max = 146 blocks/s.
- Server 2: Min = 10 blocks/s, Max = 97 blocks/s.
- Server 3: Min = 20 blocks/s, Max = 100 blocks/s
- Server 4: Min = 10 blocks/s, Max = 80 blocks/s.

If I have NOC of 16,300 then the block size would be 128,656 bytes and number of blocks would be 4096 according to equations 1, 2 and 3. The maximum number of blocks for each server according to equation 7 would be as follows:

- Server 1: $\left(\frac{146}{146+(10+20+10)}\right) \times 4096 = 3215 \text{ blocks}$
- Server 2: $\left(\frac{97}{97+(59+20+10)}\right) \times 4096 = 2135 \text{ blocks}$
- Server 3: $\left(\frac{100}{100+(59+10+10)}\right) \times 4096 = 2288 \text{ blocks}$
- Server 4: $\left(\frac{80}{80+(59+10+20)}\right) \times 4096 = 1938 \text{ blocks}$

Now, the overall storage used by ssCloud would be the sum of all the maximum blocks of the above four servers which is 9576 blocks each of size 128,656 bytes. This means that the overall storage consumed would be 1,232,009,856 bytes (around 1.2 GB). This means that the least saved storage if I only removed the zero downloaded blocks would be 865,142,144 bytes compared to the original CDDLB or DDFTP.

7.4. Chapter Conclusion

In this chapter, I have discussed the mathematics behind my partial replication load-balancing approach for providing DaaS in the cloud. I provided an estimation of the storage that could be saved using the ssCloud and the estimated download time after removing the redundant data from storage. I validated the estimates by running the experiments and found a satisfying percentage of accuracy. Finally, I noted some observations and best service optimization methods and validated those as well.

Chapter 8: Conclusion and Future Work

In this chapter, I conclude this dissertation by summarizing the research contributions and goals of this work in Section 8.1. Then, I summarize the possible future work that could be of a significance to the areas of load balancing and storage optimization in the cloud.

8.1. Summary of Research Contribution

Combining an efficient load balancing and storage consumption utilization in the cloud provides the ability to offer better services and less cost for the cloud providers. My solution focuses on enhancing both aspects, as it improves load balancing by collaborative server downloads and improves storage by reducing the amount of replicated blocks among the cloud servers.

The research contributions of this dissertation follow.

8.1.1. Static Removal of Replicated Blocks

I enhanced the collaborative dual-direction download method by removing the previously unused blocks. The first enhancement was to manually have a static removal of unused blocks from each cloud server. I have implemented this technique on top of the previous dual-direction method. The benefit was to reduce the amount of storage consumed. However, the process had to be done manually on occasion. The problem was that the storage consumption could reach its peak before any removal could be conducted.

8.1.2. Autonomic Removal of Replicated Blocks

In this contribution, I have added steps to the cloud environment where uploading files will go through a workflow of 1) determining the block size, 2) splitting the file into blocks according to the block size, 3) and uploading the file onto each server of the cloud environment. When there is a need to remove blocks, the controller will complete an analysis of the unused blocks, and those blocks will be removed. The process is automated through the upload process and the use of controller.

8.1.3. Analytical Model of Performance

My final contribution in this dissertation was to provide an analytical model of how to estimate the amount of storage saved depending on the collaborative server speeds. Moreover, I validated my expected equations against experiments conducted using the simulator. I found a high percentage of accuracy through running the experiments.

8.2. Future Work

As a future addition to this research, I considered some enhancements that could be of significant contribution to the area of load balancing and storage optimization. Below are some of the possible future works of this dissertation.

8.2.1. Auto-Recovery of Blocks

In case there was a need to restore the removed blocks, the process is easy because all blocks exist in the cloud with their unique identifier. An analysis of the

need to restore any block into server X could be a useful enhancement to the current approach.

8.2.2. Partial Editing of the File

Moreover, I discussed throughout this thesis the uploading and downloading of data in the cloud, which is the scope of my research. However, when there is a need to edit or modify a portion of a large file, there should be an improvement to the partial replication load-balancing technique that I provided. This by itself is a huge research effort, which could provide a significant contribution to the topic.

8.2.3. Fault Tolerance Handling

As the cloud is known for its elasticity and cloud servers can join and leave the cloud at different times, an analysis of how the ssCloud can handle fault tolerance in the case when a server fails or leaves the cloud would be needed. The backup of the removed blocks and the amount of replication needed in such cases would be very useful.

8.2.4. Enhancing the Security of ssCloud

Another future work is enhancing the security of the partial files. I mentioned previously that security is an important research area in distributed DaaS. There are many research studies conducted on enhancing the security of the data exchanged in the cloud, as I have seen earlier in this thesis. Using a partial replication could be a solution used by ssCloud as well as other approaches like

RRNS. Therefore, enhancing the security of the ssCloud by adding new features to the partial replication would be an interesting solution.

8.2.5. Implementation and Evaluation of a Compression Method

Since file compression is a popular solution for reducing the storage used, I think that it could further enhance the storage consumption of ssCloud. This could be done by compressing the never downloaded blocks instead of removing them permanently. This may create additional tradeoffs between download speed, storage saving, and reliability. As a result, I plan to evaluate the effects of compressing files at the servers' side in terms of storage and performance to verify that it will not significantly increase the overall download time.

8.2.6. Implementation of the Full Idea on Top of Simulation

To better evaluate the full idea of the compression and additional other features that could be added in the future to the main idea, I need to implement a simulation environment where the full cloud is simulated and different attributes could be changed on large scale environment. This would help in evaluating most of the points in the future work.

Bibliography

- [1] Rimal, B.P., E. Choi, and I. Lumb, "A taxonomy and survey of cloud computing systems," In *proc. 5th International Joint Conference on INC, IMS and IDC*, IEEE. 2009.
- [2] Armbrust, M., Fox, O., Griffith, R., Joseph, A. D., Katz, Y., Konwinski, A., and Zaharia, M, "M.: Above the clouds: a Berkeley view of cloud computing" in *Citeseer*. 2009.
- [3] Muniswamy-Reddy, K. K., Macko, P., and Seltzer, M. I, "Provenance for the Cloud", In *FAST* (Vol. 10, pp. 15-14). 2010.
- [4] Goyal, A., and Dadizadeh, S., "A survey on cloud computing". *University of British Columbia Technical Report for CS, 508*, 55-58. 2009.
- [5] Buyya, R., Ranjan, R., and Calheiros, R. N., "Intercloud: Utility-oriented federation of cloud computing environments for scaling of application services". In *Algorithms and architectures for parallel processing* (pp. 13-31). Springer Berlin Heidelberg. 2010.
- [6] Sakr, S., Liu, A., Batista, D. M., and Alomari, M. "A survey of large scale data management approaches in cloud environments", *Communications Surveys & Tutorials. IEEE, 13*(3), 311-336. 2011.
- [7] Schaffer, H. E., "X as a service, cloud computing, and the need for good judgment", *IT professional, 11*(5), 4-5. 2009.
- [8] Karlsson, J., *et al.*, "Enabling large-scale bioinformatics data analysis with cloud computing.", In *Parallel and Distributed Processing with Applications (ISPA), 2012 IEEE 10th International Symposium on*. IEEE. 2012.
- [9] Cuzzocrea, A., Song, I. Y., and Davis, K. C., "Analytics over large-scale multidimensional data: the big data revolution!", In *Proceedings of the ACM 14th international workshop on Data Warehousing and OLAP* (pp. 101-104). ACM. 2011.
- [10] Wang, L., Kunze, M., Tao, J., and von Laszewski, G., "Towards building a cloud for scientific applications". In *Advances in Engineering software, 42*(9), 714-722. 2011.

- [11] Olson, J. A., "Data as a service: are I in the clouds?". *Journal of Map & Geography Libraries*, 6(1), 76-78. 2009.
- [12] Dikaiakos, M. D., Katsaros, D., Mehra, P., Pallis, G., and Vakali, A. "Cloud computing: Distributed internet computing for IT and scientific research". *Internet Computing, IEEE*, 13(5), 10-13. 2009.
- [13] Youseff, L., M. Butrico, and D. Da Silva, "Toward a unified ontology of cloud computing," *Grid Computing Environments Workshop, IEEE GC'E'08.*, 2008.
- [14] Agarwal, S., Dunagan, J., Jain, N., Saroiu, S., Wolman, A., and Bhogan, H. (2010, April). Volley: Automated Data Placement for Geo-Distributed Cloud Services. In *NSDI* (pp. 17-32).
- [15] Radojevic, B., and Mario, Z. "Analysis of issues with load balancing algorithms in hosted (cloud) environments." *MIPRO, 2011 Proceedings of the 34th International Convention*. IEEE. 2011.
- [16] Letaifa, A. B., *et al.* "State of the Art and Research Challenges of new services architecture technologies: Virtualization, SOA and Cloud Computing." *International Journal of Grid and Distributed Computing* 3.4. 69-88. 2010.
- [17] Huang, Y., *et al.* "Cloud download: using cloud utilities to achieve high-quality content distribution for unpopular videos." *Proceedings of the 19th ACM international conference on Multimedia*. ACM. 2011.
- [18] Reichman, A. "File storage costs less in the cloud than in-house." *Forrester*, 25th August, available at: <http://www.forrester.com/FileStorage+Costs+Less+In+The+Cloud+Than+InHouse/fulltext/-/E-RES57696>. 2011.
- [19] Dillon, T., Wu, C., and Chang, E. "Cloud computing: issues and challenges". In *Advanced Information Networking and Applications (AINA), 2010 24th IEEE International Conference on* (pp. 27-33). IEEE. 2010.
- [20] Dinh, Hoang T., *et al.* "A survey of mobile cloud computing: architecture, applications, and approaches." *Wireless communications and mobile computing* 13.18: 1587-1611. 2013.
- [21] de Oliveira, D., Baião, F. A., and Mattoso, M. "Towards a taxonomy for cloud computing from an e-science perspective". In *Cloud Computing* (pp. 47-62). Springer, London. 2010.

- [22] Abadi, D. J. "Data Management in the Cloud: Limitations and Opportunities". *IEEE Data Eng. Bull.*, 32(1), 3-12. 2009.
- [23] TechCrunch. "Dropbox Hits 275M Users And Launches New Business Product To All" | TechCrunch." *TechCrunch*. N.p., n.d., Web. 30 Apr. 2014.
- [24] Sevier, M., Fifield, T., and Katayama, N. "Belle Monte-Carlo production on the Amazon EC2 cloud". In *Journal of Physics: Conference Series* (Vol. 219, No. 1, p. 012003). IOP Publishing. 2010.
- [25] Hamburger, Ellis. "Google Drive vs. Dropbox, SkyDrive, SugarSync, and others: a cloud sync storage face-off." *The Verge*, 2012.
- [26] Greenberg, Albert, *et al.* "The cost of a cloud: research problems in data center networks." *ACM SIGCOMM Computer Communication Review* 39.1, 2008.
- [27] Casas, P., Fischer, H. R., Suette, S., and Schatz, R. "A first look at quality of experience in personal cloud storage services". In *Communications Workshops (ICC), 2013 IEEE International Conference on* (pp. 733-737). IEEE. 2013.
- [28] Baliga, J., Ayre, R. W., Hinton, K., and Tucker, R. "Green cloud computing: Balancing energy in processing, storage, and transport". *Proceedings of the IEEE*, 99(1), 149-167. 2011.
- [29] Dory, T., Mejías, B., Van Roy, P., and Tran, N. L. "Measuring elasticity for cloud databases". In *CLOUD COMPUTING 2011, The Second International Conference on Cloud Computing, GRIDs, and Virtualization* (pp. 154-160). 2011.
- [30] Benson, K., Dowsley, R., and Shacham, H. "Do you know where ymy cloud files are?". In *Proceedings of the 3rd ACM workshop on Cloud computing security workshop* (pp. 73-82). ACM. 2011
- [31] Wee, S., and Liu, H. "Client-side load balancer using cloud". In *Proceedings of the 2010 ACM Symposium on Applied Computing* (pp. 399-405). ACM. 2010.
- [32] Zhang, C., De Sterck, H., Aboulmaga, A., Djambazian, H., and Sladek, R. "Case study of scientific data processing on a cloud using hadoop". In *High performance computing systems and applications* (pp. 400-415). Springer Berlin Heidelberg. 2010.

- [33] Wang, S. C., Yan, K. Q., Liao, W. P., and Wang, S. S. "Towards a load balancing in a three-level cloud computing network". In *Computer Science and Information Technology (ICCSIT), 2010 3rd IEEE International Conference on* (Vol. 1, pp. 108-113). IEEE. 2010.
- [34] Beloglazov, A., and Buyya, R. "Energy efficient resource management in virtualized cloud data centers". In *Proceedings of the 2010 10th IEEE/ACM International Conference on Cluster, Cloud and Grid Computing* (pp. 826-831). IEEE Computer Society. 2010.
- [35] Tseng, F. H., Chen, C. Y., Chou, L. D., and Chao, H. C. "Implement a reliable and secure cloud distributed file system". In *Intelligent Signal Processing and Communications Systems (ISPACS), 2012 International Symposium on* (pp. 227-232). IEEE. 2012.
- [36] Modi, C., Patel, D., Borisaniya, B., Patel, A., and Rajarajan, M. "A survey on security issues and solutions at different layers of Cloud computing". *The Journal of Supercomputing*, 63(2), 561-592. 2013.
- [37] Parekh, Disha H., and R. Sridaran. "An Analysis of Security Challenges in Cloud Computing." *IJACSA) International Journal of Advanced Computer Science and Applications* 4.1. 2013.
- [38] Mao, H., Xiao, N., Shi, W., and Lu, Y. "Wukong: Toward a Cloud-Oriented File Service for Mobile Devices". In *Services Computing (SCC), 2010 IEEE International Conference on* (pp. 498-505). IEEE. 2010.
- [39] Saranya, S. Mohana, and M. Vijayalakshmi. "Interactive mobile live video learning system in cloud environment." *Recent Trends in Information Technology (ICRTIT), 2011 International Conference on*. IEEE. 2011.
- [40] Broberg, J., Buyya, R., and Tari, Z. "MetaCDN: Harnessing 'Storage Clouds' for high performance content delivery". *Journal of Network and Computer Applications*, 32(5), 1012-1022. 2009.
- [41] Grossman, R. L., Gu, Y., Sabala, M., and Zhang, W. "Compute and storage clouds using wide area high performance networks". *Future Generation Computer Systems*, 25(2), 179-183. 2009.

- [42] Thakar, A., and Szalay, A. "Migrating a (large) science database to the cloud". In *Proceedings of the 19th ACM International Symposium on High Performance Distributed Computing* (pp. 430-434). ACM. 2010.
- [43] Sun, Z., Shen, J., and Yong, J. "DeDu: Building a deduplication storage system over cloud computing". In *Computer Supported Cooperative Work in Design (CSCWD), 2011 15th International Conference on* (pp. 348-355). IEEE. 2011.
- [44] Dong, B., Zheng, Q., Tian, F., Chao, K. M., Ma, R., and Anane, R. "An optimized approach for storing and accessing small files on cloud storage". *Journal of Network and Computer Applications*, 35(6), 1847-1862. 2012.
- [45] Fesehaye, D., Malik, R., and Nahrstedt, K. "Scalable Distributed File System for Cloud Computing". Technical report, University of Illinois at Urbana-Champaign (UIUC), 2010.
- [46] Randles, M., D. Lamb, D., and A. Taleb-Bendiab, "A Comparative Study into Distributed Load Balancing Algorithms for Cloud Computing," in *Proc. IEEE 24th International Conference on Advanced Information Networking and Applications Workshops (WAINA)*, Perth, Australia, 2010.
- [47] Ananthanarayanan, R., Gupta, K., Pandey, P., Pucha, H., Sarkar, P., Shah, M., and Tewari, R. "Cloud analytics: Do I really need to reinvent the storage stack". In *Proceedings of the 2009 Workshop on Hot Topics in Cloud Computing (HotCloud 09)*, San Diego, California. 2009.
- [48] Biran, O., Corradi, A., Fanelli, M., Foschini, L., Nus, A., Raz, D., and Silvera, E. "A stable network-aware vm placement for cloud systems". In *Proceedings of the 2012 12th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (ccgrid 2012)* (pp. 498-506). IEEE Computer Society. 2012.
- [49] Armbrust, M., Fox, A., Griffith, R., Joseph, A. D., Katz, R., Konwinski, A., ... and Zaharia, M. "A view of cloud computing". *Communications of the ACM*, 53(4), 50-58. 2010.
- [50] Sotomayor, B., Montero, R. S., Llorente, I. M., and Foster, I. "Virtual infrastructure management in private and hybrid clouds". *Internet computing, IEEE*, 13(5), 14-22. 2009.
- [51] Nishant, K., P. Sharma, V. Krishna, C. Gupta, KP. Singh, N. Nitin, and R. Rastogi. "Load Balancing of Nodes in Cloud Using Ant Colony Optimization."

- In *proc. 14th International Conference on Computer Modelling and Simulation (UKSim)*, IEEE, pp: 3-8, 2012.
- [52] Zhang, Z. and X. Zhang, "A load balancing mechanism based on ant colony and complex network theory in open cloud computing federation," In *proc. 2nd International Conference on Industrial Mechatronics and Automation (ICIMA)*, IEEE, Vol. 2, pp:240-243, 2010.
- [53] Kolb, L., A. Thor, and E. Rahm, "Load Balancing for MapReduce based Entity Resolution," in *proc. 28th International Conference on Data Engineering (ICDE)*, IEEE, pp: 618-629, 2012.
- [54] Gunarathne, T., T-L. Wu, J. Qiu, and G. Fox, "MapReduce in the Clouds for Science," in *proc. 2nd International Conference on Cloud Computing Technology and Science (CloudCom)*, IEEE, pp. 565-572, November/December 2010.
- [55] Ni, J., Y. Huang, Z. Luan, J. Zhang, and D. Qian, "Virtual machine mapping policy based on load balancing in private cloud environment," in *proc. International Conference on Cloud and Service Computing (CSC)*, IEEE, pp. 292-295, December 2011.
- [56] T-Y., W-T. Lee, Y-S. Lin, Y-S. Lin, H-L. Chan, and J-S. Huang, "Dynamic load balancing mechanism based on cloud storage" in *proc. Computing, Communications and Applications Conference (ComComAp)*, IEEE, pp. 102-106, January 2012.
- [57] Ren, X., R. Lin, and H. Zou, "A dynamic load balancing strategy for cloud computing platform based on exponential smoothing forecast," in *proc. International Conference on Cloud Computing and Intelligent Systems (CCIS)*, IEEE, pp. 220-224, September 2011.
- [58] Lee, R. and B. Jeng, "Load-balancing tactics in cloud," in *proc. International Conference on Cyber-Enabled Distributed Computing and Knowledge Discovery (CyberC)*, IEEE, pp. 447-454, October 2011.
- [59] Al-Jaroodi, J. and N. Mohamed, "DDFTP: Dual-Direction FTP," in *proc. 11th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGrid)*, IEEE, pp. 504-503, May 2011.

- [60] Mohamed, N. and J. Al-Jaroodi, "Delay-tolerant dynamic load balancing," in *proc. 13th International Conference on High Performance Computing and Communications (HPCC)*, pp:237-245, September, 2011.
- [61] Mohamed, N., J. Al-Jaroodi, and A. Eid "A Dual-Direction Technique for Fast File Downloads with Dynamic Load Balancing in the Cloud," in *The Journal of Network and Computer Applications*, Elsevier, Vol. 36, No. 4, pp. 1116-1130, July 2013.
- [62] Wang, S-C., K-Q. Yan, W-P. Liao, and S-S. Wang, "Towards a load balancing in a three-level cloud computing network," in *proc. 3rd International Conference on Computer Science and Information Technology (ICCSIT)*, IEEE, Vol. 1, pp:108-113, July 2010.
- [63] Sang, A., X. Wang, M. Madihian, and RD. Gitlin, "Coordinated load balancing, handoff/cell site selection, and scheduling in multi-cell packet data systems," in *Wireless Networks*, Vol. 14, No. 1, pp: 103-120, January 2008.
- [64] Zhang, Y., Liu, W., and Song, J. "A novel solution of distributed file storage for cloud service". In *Computer Software and Applications Conference Workshops (COMPSACW), 2012 IEEE 36th Annual* (pp. 26-31). IEEE. July 2012.
- [65] Srivastava, S., Gupta, V., Yadav, R., and Kant, K. "Enhanced distributed storage on the cloud". In *Computer and Communication Technology (ICCCT), 2012 Third International Conference on* (pp. 321-325). IEEE. November 2012.
- [66] Celesti, A., Fazio, M., Villari, M., and Puliafito, A. "Adding long-term availability, obfuscation, and encryption to multi-cloud storage systems". *Journal of Network and Computer Applications*. 2014.
- [67] Villari, M., Celesti, A., Tusa, F., and Puliafito, A. "Data reliability in multi-provider cloud storage service with rrms". In *Advances in Service-Oriented and Cloud Computing* (pp. 83-93). Springer Berlin Heidelberg. 2013.
- [68] Villari, M., Celesti, A., Fazio, M., and Puliafito, A. "Evaluating a file fragmentation system for multi-provider cloud storage". *Scalable Computing: Practice and Experience*, 14(4). 2014.

- [69] Abu-Libdeh, H., Princehouse, L., & Weatherspoon, H. "RACS: a case for cloud storage diversity." In *Proceedings of the 1st ACM symposium on Cloud computing*, pp. 229-240. ACM. 2010.
- [70] Bessani, A., Correia, M., Quaresma, B., André, F., & Sousa, P. "DepSky: dependable and secure storage in a cloud-of-clouds". *ACM Transactions on Storage (TOS)*, 9(4), 12. 2013.
- [71] Kotla, R., Alvisi, L., & Dahlin, M. SafeStore: a durable and practical storage system. In *USENIX Annual Technical Conference*, pp. 129-142. 2007.
- [72] Dobre, D., Viotti, P., & Vukolić, M. Hybris: Robust Hybrid Cloud Storage. In *Proceedings of the ACM Symposium on Cloud Computing*, pp. 1-14. ACM. 2014.
- [73] Grosu, D., A.T. Chronopoulos, and M. Leung, "Cooperative load balancing in distributed systems," in *Concurrency and Computation: Practice and Experience*, Vol. 20, No. 16, pp. 1953-1976, 2008.
- [74] Ranjan, R., L. Zhao, X. Wu, A. Liu, A. Quiroz, and M. Parashar, "Peerto- peer cloud provisioning: Service discovery and load-balancing," in *Cloud Computing - Principles, Systems and Applications*, pp: 195-217, 2010.
- [75] Al-Jaroodi, J., Mohamed, N., and Al Nuaimi, K. "An Efficient Fault-Tolerant Algorithm for Distributed Cloud Services," *IEEE NCCA*, December 2012.
- [76] Wu, J., et al. "Cloud storage as the infrastructure of cloud computing," *Intelligent Computing and Cognitive Informatics (ICICCI)*, 2010 International Conference on. IEEE, 2010.
- [77] Zeng, W., et al. "Research on cloud storage architecture and key technologies." *Proceedings of the 2nd International Conference on Interaction Sciences: Information Technology, Culture and Human*, ACM, 2009.
- [78] Li, Z., Wilson, C., Jiang, Z., Liu, Y., Zhao, B. Y., Jin, C., ... and Dai, Y. "Efficient batched synchronization in dropbox-like cloud storage services". In *Middleware 2013* (pp. 307-327). Springer Berlin Heidelberg. 2013.
- [79] Drago, I., Mellia, M., M Munafo, M., Sperotto, A., Sadre, R., and Pras, A. "Inside dropbox: understanding personal cloud storage services". In *Proceedings of the 2012 ACM conference on Internet measurement conference* (pp. 481-494). ACM. November 2012.

- [80] Mohamed, N. and J. Al-Jaroodi, "MidCloud: An Agent-Based Middleware for Effective Utilization of Replicated Cloud Services," in *Software: Practice and Experience, Wiley*, 45(3): 343-363, March 2015.
- [81] Mohamed, N., J. Al-Jaroodi, and H. Jiang, "DDOps: Dual-Direction Operations for Load Balancing on Non-Dedicated Heterogeneous Distributed Systems," in *Cluster Computing, Springer, Vol. 17, No. 2, pp. 503-528, June 2014. DOI: 10.1007/s10586-013-0294-3*, 2014.
- [82] Al-Jaroodi, J., N. Mohamed, and A. Eid "Dual Direction Big Data Download and Analysis," *ACM SIGMETRICS Performance Evaluation Review (PER), ACM, Vol. 41, Issue 4, pp. 98-101*, March 2014.

List of Publications

- [1] Al Nuaimi, Klaithem, et al. "A survey of load balancing in cloud computing: Challenges and algorithms." *Network Cloud Computing and Applications (NCCA), 2012 Second Symposium on*. IEEE, 2012.
- [2] Al Nuaimi, Klaithem, et al. "A partial replication load balancing algorithm for distributed Data as a Service (DaaS)." *High Performance Computing and Simulation (HPCS), 2013 International Conference on*. IEEE, 2013. [ERA Ranking: B]
- [3] Al Nuaimi, Klaithem, et al. "A Novel Approach for Dual-Direction Load Balancing and Storage Optimization in Cloud Services." *Network Computing and Applications (NCA), 2014 IEEE 13th International Symposium on*. IEEE, 2014. [ERA Ranking: A]
- [4] Al Nuaimi, Klaithem, et al. "Dual direction load balancing and partial replication storage of cloud DaaS." *Cloud Networking (CloudNet), 2014 IEEE 3rd International Conference on*. IEEE, 2014.
- [5] Al Nuaimi, Klaithem, et al. "Partial Storage Optimization and Load Control Strategy of Cloud Data Centers." *The Scientific World Journal, communication section, 2015*. In Press. [Impact Factor: 1.219]
- [6] Al Nuaimi, Klaithem, et al. "A Self-Optimized Storage for Distributed Data as a Service". *Convergence of Distributed Clouds, Grids and their Management Track, The 24th IEEE International Conference on Enabling Technologies: Infrastructure for Collaborative Enterprises*. [ERA Ranking: B]

- [7] Al Nuaimi, Klaithem, et al. "ssCloud: A Smart Storage for Distributed DaaS on the Cloud". *Cloud Computing (CLOUD), 2015 IEEE 8th International Conference on IEEE*, 2015. [ERA Ranking: B]