Distributed Disk Store

Mohammad Samawat Ullah[†]
Department of Computer Science
American International University-Bangladesh, Bangladesh
samawat@aiub.edu

Dr. Stephen McGough
School of Computing Newcastle University, Newcastle upon Tyne, UK
stephen.mcgough@newcastle.ac.uk

ABSTRACT

Background: In large organizations and most often in universities, very little storage space is utilized of the individual computers. Operating systems and basic functionalities take minimum space with user data being stored on external servers, leaving majority of available storage unoccupied. Therefore, this dilemma leads to a potential research area to investigate the possibility of developing a distributed file-store capable of exploiting this unused disk space.

Aims: The Distributed Disk Store (DDS) system was focused to feasibility analysis and to develop a simulation model for distributed file storage, considering replication to avoid loss of data due to loss of computers/disks allowing the recovery of files and to some extent allow user preference for file allocation in disk storage in a university environment.

Method: Discrete-Event Simulation (DES) model and Object-oriented programming language has been considered to deduce logical implication of Distributed Disk Store (DDS). Besides, the project work uses current and previous research works, supervisory recommendation from similar projects, along with programming models to analyze the effectiveness of the measures taken to address Durham University's idle computers space utilization problem.

Expected Outcomes: The simulation model coordinates for networked computers and stores files/objects in distributed manner with a track record of virtual files container, replication for availabilities and somewhat considers user preference.

Keywords – Discrete-Event Simulation, Distributed Disk Store, Distributed Storage, Object based simulation model, Distributed file replication, DDS, file mirroring Distributed storage for large organization, Network storage.

CCS CONCEPTS

• Distributed storage • Distributed architectures • Distributed computing models

ACM Reference Format:

Mohammad Samawat Ullah and Dr. Stephen McGough 2020. Distributed Disk Store. In Proceedings of ICCA 2020 - International Conference on Computing Advancements (ICCA 2020). ACM, Dhaka, Bangladesh, 10 pages. https://doi.org/10.1145/3377049.3377091

1 Introduction

Distributed Disc Store (DDS) is a fascinating region in distributed computing that demands in-depth research. In large organizations and often in universities, there are hundreds and thousands of computers mainly assigned for use by members. However, very little of the storage space is utilized on these computers. The operating system takes only minimum space of entire available storage and the remainder is mostly unoccupied with user files stored on remote servers. Likewise, these organizations have a need to store significant amounts of data including research samples and results. So, we found that a system requires to be established to combine all the inactive storage to enable disk storage in a distributed environment. For example, based on the Durham University Computing and Information Services (CIS) survey data [7], there is total 7030 number of computers assigned for user (i.e. student's and staffs etc.) and on an average each machine's hard drive size is 320 GB. Moreover, every user is allocated 400MB spaces for usage and all data is stored on the user mapped J drive. As a result, the hard drive space is rarely used except for the space of operating systems and basic applications. Besides, it was also stated by CIS that on an average the approximate usage of per computer is not more than 30% (674.88 TB). Thus, reminder 70% (1574.72 TB) of the total (2249.6 TB) remains unutilized (*Figure-1*). Therefore, the DDS system offers a solution to use these idle spaces (i.e. 1574.72 TB in Durham University) as combined resources to offer a distributed disk store. We strongly feel that, this DDS simulation model will work as a great motivation to implement a real-life application for many large organizations.

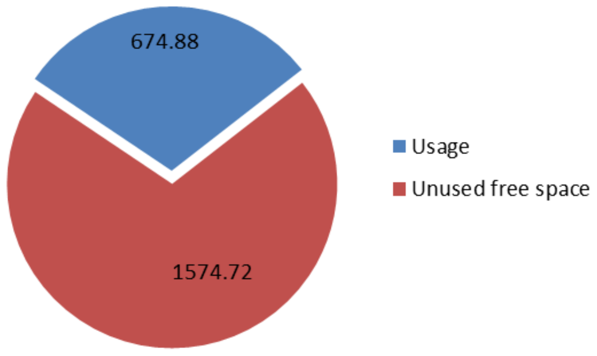

Figure 1: Durham University computer space usage (in TB)

On the other hand, some might argue that, "nowadays storage like physical HDD or online cloud subscription has become much cheaper and therefore we might not need a distributed storage". To answer this, let's consider an example of Newcastle University, UK where the University management bought a 40TB storage disk to store departmental research works. However, the storage was filled unpredictably quick (i.e. within few weeks). So, the need for extra storage is still there while major portion of thousands of storage disks are sitting idle using a little portion for basic computer applications. For instance, referring to the Durham University unutilized disk space (1574.72 TB), it would be more than enough compare to the 40TB storage bought at Newcastle University. Thus, we offer the DDS system as a solution that is focused to utilize the unused spaces which the organization already paid for and will cost nothing like any storage subscription (i.e. Dopbox[1] or Amazon[2] cloud etc.).

The DDS project is segmented in different parts in order to achieve various objectives to accomplish the goal of the project. Firstly, the preliminary question of this research is feasibility analysis of distributed file-store capable of exploiting unused disk spaces along with a simple simulation model for file storing and retrieval from networked computers. To achieve a solution for this, research and analysis of related literature was accomplished and a Discrete Event Simulation model approach with Object based storage [12] was adopted to demonstrate such solution testing before essentially building the real application. Since Object based storage structures follow encapsulation [12], this offers more security compared to block based storage. Also, for object-based storage simulation, object-oriented programming languages like SIMULA, Java were explored and therefore among all we have decided to carry the implementation process using java.

Secondly, file splitting to distribute the file parts in networked machines, merging the distributed file parts from circulated locations on retrieval request and system resiliency were next level demand for DDS project. Splitting and merging were considered for a scenario where one machine's free space is not enough to store a file. Thus, we have established two distinct methods for file saving to single and/or multiple computers together with method for retrieving from single and/or multiple machines. In addition, another functional requirement was to avoid the loss of file due to faulty storing node and potentially recovering files from a subset of powered up computers. This is achieved using mirroring technique [17]. For total N faulty computers in DDS system we have considered to make N+1 copies of file to ensure the system resiliency. For example, during the unit testing of DDS model we have assumed at a time there might be two faulty computers in the system. Thus, three copies of file were generated to ensure the file availability. Consequently, failure in individual component has minimum impact and system recover files from its subsets. Moreover, the DDS simulation model overcomes the challenge of listing objects of a specified disk pointed out in RUSH algorithm [10] and data uniformity after replication indicated in [17].

Thirdly, the advance level functional requirement for the DDS project was to ensure the user preferences to store file in a storing node. Particularly, while storing machine joins to the DDS network, DDS system is supposed to facilitate the environment for this machine to express what files can be stored to this computer and ensure only specified files are stored. Accordingly, the system allows option for the owners of the computers to specify file preference and free space they would like to volunteer. For example, one machine in English department can specify if they want to accept only English department's files and volunteer 200GB free space. As a result, this 200GB storage can be used dedicatedly for English department's files.

The overall research followed the Discrete-Event Simulation (DES) model [2] and utilized object-oriented programming language (Java) to develop the simulation model and to demonstration the feasibility of such solution testing before essentially building them into real application. We are optimistic that this simulation model will also help to draw interpretation about the system and to avoid any adverse effect [2].

Since the large organization usual network speed is 100Mbps fast Ethernet and we are considering a University network as a model for our research, the simulation parameter units are assumed based on similar state. Even though simulation parameters do not really reflect the situation in real life, we have tried to assume parameters to closely relate DDS with a real system. Therefore, we assumed each unit time in seconds (1-unit time = 1 second), file size and computers free space in GB (1-unit file size = 1 Giga Byte).

The reminder of the paper is segmented into sections and section-II briefly describes the literature review of related research works, and section-III describes the solutions to the problems in detail including the design and implementation details. Section-IV provides the results of the solutions including the information on experimental settings together with benefits and disadvantages of the proposed solutions. Finally, section-V presents overall conclusions of the DDS system.

1.1 Related Work

Reliability Mechanism for very large systems (RMVLS) [17]

This study is mainly focused to ensure the availability of large system regardless of individual component failure. Also, two main error (nonrecoverable failure and drive failure) were considered to detect and two-way mirroring technique were suggested as solution. Besides, Object-based storage device (OSD) (Wang et al., 2004) is a network attached storage device that beaks files into objects to ease the distribution of in networked devices. The decentralized OSD was effective but costly due to custom hardware and software manageability. On the other hand, OSD software integration with each drive seemed to be less costly while it needs more redundancy. Therefore, the authors, proposed mirroring and signature method to recognize and correct the bad files. While mirroring denotes keeping multiple copies of a file, signature method states storing signature of an object with the object. The research states that, redundancy and recovery mechanisms work effectively to overcome disk failures. However, data uniformity after replication is still a question to resolve. Also, two-way mirroring cannot ensure long time data retention. Although, three-way mirroring is an anticipated solution for this, it might need detail exploration for that.

The project is related to our DDS model and was supportive in storage strategy. Nowadays Object-Oriented concept is most popular and widely used. Therefore, OSD was thoroughly analyzed to utilize in DDS. Also, we tried to examine and fit the mirroring strategy in DDS. We cross checked whether two-way mirroring is sufficient for us or we need three-way mirroring

[1] www.dropbox.com

[2] www.amazon.com

Bigtable: A Distributed Storage System for Structured Data [5]

Bigtable is a distributed storing structure used by most google products and this research is mainly concentrated to Bigtable data model. It uses scalable distributed Google File System (GFS) that is supported by numerous inexpensive client machines (Ghemawat et al., 2003). Hence, Bigtable has clearly defined and detailed API that allows range of functions for cluster and table controls. Besides, Cluster management system is one of the key components of Bigtable, which is primarily responsible for resource administration, failure response and status monitoring of involved machines. Some real applications like Google Analytics, Google earth, personalized search widely use Bigtable to keep the track of distributed files information. It is dynamic and widely acceptable for ease feature of adding additional nodes with the system to enable high scalability.

The Bigtable data model is a substantial content to relate with our Distributed Disk Storage (DDS) system and used for file locator in DDS environment. As this system is widely accepted by renowned systems (i.e. google applications), it seemed more reliable to relate with DDS file location tracking and flexibilities in disk addition. Although Bigtable offers enormous features in a broader level, we tried to relate and limit our research up to the tracking feature

Object-Based Storage [12]

Due to the advice of industrial and academic research, storage technology was proposed to adopt a change in basic structure. Consequently, Object-Based Storage (OBS) came into the account with variable size, attribute and new security features. The vital contribution of OBS allows the beneficial capabilities of both files and blocks. In particular, the key difference between OBS and Block based storage (BBS) is not physical but accessing methods. Therefore, OBS structures follow encapsulation to offer secure and simple system. This research states that OBS is the necessary change for next generation storage method. However, unsupervised configuration, security, recovery and management in OBS are defined as prospective change by authors in this research.

The detailed investigation regarding OBS supported us to determine file storage mechanism and design the data structure of DDS. Precisely, as described by the authors and recent trends of storage technology, we followed OBS for our file storage in distributed environment

Replication Under Scalable Hashing: A Family of Algorithms for Scalable Decentralized Data Distribution [10]

RUSH (Replication Under Scalable Hashing) is a collection of algorithms that is mainly responsible for mapping duplicate objects to specified nodes or disks. Also, it determines and distributes the files based on user preference. Besides, it checks and ensures that the same files cannot be in the same machine to avoid unnecessary redundancy and physical storage consumption. Moreover, it facilitates concurrent client access to numerous servers. Furthermore, RUSH considered Object-based Storage Devices (OSDs) (Wang et al., 2004) for its own file systems and block allocation. More specifically, OSDs requires key-data pair to store and only key for data recollection.

In addition, RUSH allocates all the replicas to networked OSDs and clients detect the location of the object using server lists. More precisely, while there is any new node added or removed, the algorithm does the necessary adjustments to maintain the balance. Likewise, it does the operation in decentralized fashion. While there are so many nodes in the operation, decentralization is a primary need for faster search without any interference from other nodes. The concept was initiated considering the situation of cluster addition for system growth. It eases to locate the sub-cluster that holds object and then find specific disk with that object. Still, RUSH algorithm cannot list the objects of a specified disk. Deliberately, it must look through object and replicas to locate the disk. Then apply filtration logic of given disk to find he actual objects of that disk.

To summarize, RUSH family algorithms are closely related to the systems where system node modification and object replication are considered. Hence, we compared RUSH to correlate with our DDS for file replication, decentralization and therefore avoid file duplicity on same disk. To sum up, it opens a scope for finding and integrating objects list feature of a specified disk in DDS. In addition, we have also combined system time with event id to create a unique file id and thus to avoid duplicity.

Droplet: A Distributed Solution of Data Deduplication [18]

Deduplication is a method to find the same data portion in a disk and disallow to have more alike copies. Therefore, *Droplet* is the system that facilitates the deduplication and portrayed in this piece of research work. Compressions and deduplications are commonly used to utilize disk storage and well-organized backups. While compression encodes original data by detecting duplicate data sample within segments, deduplication removes repeating data and keeps only one copy in store. The deduplication method considers simple fixed-length block (FLB) that divide data in flat pieces and dynamic variable-length block (VLB) that leaves one indicator in segmented data block to keep the track. Therefore, VLB allows more flexibility for data insertion or modifications.

There was a concern by *[19]* about faster duplicate data identification considering IO operations. So, the ‘fingerprint’ indexing which is closely related to hash function is considered. Therefore, deduplication proved to be better than compression. As a result, d*roplet* was developed to combine with 3 components meta-data, fingerprint and storage servers for monitoring, apply and storing.

Clearly, Droplet research work supported us to deduce the idea of checking duplicity of a file in the same computer and thereby using hash function, we guaranteed one computer can only have one copy of a file

A Case for Redundant Arrays of Inexpensive Disks (RAID) [4]

Redundant Array of Inexpensive Disk (RAID) is shaped based on magnetic disk technology and considered as a better alternative over Single Large Expensive Disk (SLED) in the scale performance, trustworthiness and power consumption. Unreliable system tends to fail more frequently and in an unreliable system, users want to make back up of a single file. Thus, the disks are filled with additional redundant files which could be avoided using RAID. RAID breaks the array of disks into reliability groups with a check sign in each other. As a result, if any disk fails it will be recreated using redundant information.

In our system, bearing the RAID concept in mind, we have created multiple (N+1) copies of files by assuming there might be N faulty computers rather than using considering physical disk amendment. However, based on the DDS Big Table data we are considering replacement of faulty files for future work.

Discrete-Event Simulation of Queues with Spreadsheets: A Teaching Case [7]

The investigation discussed about four simple queuing systems using discrete event simulation model as event driven discrete simulation deals with events to modify system status. In a multi-server queuing model, arrival and departure are two main events. The simple logic is to check which event happened first and the simulation runs until all the requests are processed. Although the arrival process is same as multi-server queuing, the departure process handling is altered in tandem single server queue where server one passes the job to server two. On the other hand, cycling queuing model works with constant work in progress system where a job course serially goes through n working station and after completion it leaves the system.

Considering all three models discussed in this research work, we have found the multi-server queuing model as a close match to construct our initial discrete event simulation model. However, suitable algorithm logic and implementation coding was designed by us for DDS system.

Feasibility of a Serverless Distributed File System Deployed on an Existing Set of Desktop PCs [3]

The serverless system offers high availability and reliability by distributing multiple replicas to networked client machines thereby assigning global namespace and location transparent access for files. The authors claim that the system gives advantage over expensive centralized server that requires special hardware, geographic dependency, RAID disks [4] and system administrators. Unlike previous researches that assume the client machines are mutually trusting, this research considers the realistic scenario where there is no trust between client machines with the motivation that this will discard the need for central administration. Also, to overcome the distrust problem, cryptographic techniques were adopted for data privacy and integrity. Although replicas demand more spaces, the reliability over inexpensive storage cost was measured as fair trade off. Since reliability is one of the top priorities here, number of distributed copies never goes to a minimum level and the free space per machine is decided by the computer owner. The system utilizes the spaces using compression and decompression during file storage and retrieval. Besides, the system can select machine for client service request based on two criteria such as topological closeness and light load on machine. The replicas were sent to the system based on machine's availability and uptimes. Besides, all the related files are kept together. So that either all or none files can be saved or retrieved. Also, if the machine does not respond for lengthy period, it is considered as retired and replaced with replicas to avoid another failure on similar files.

Linking this research with our project, DDS is also mainly focused on reliability and thus we have considered replication factor and additionally incorporated user preference option with an additional scope for file splitting and retrieving.

Briefly, analyzing all the research works in review we deduced that [12,16,17] *have the similar focus on their research and it was about storage strategy where they put strong emphasize on* object-based storage. On the other hand, [5] proved to provide a good start for file tracking methodologies in Distributed disk storage systems. Although, Droplet application [18] is not the same but it offers indexing features which we can presume similar. Alternatively, [14] initiated the proposal for disk scrubbing for early disk failure detection in one of his researches which is a distinct work then all we considered in our DDS extension. Likewise, [15] focused on disk identity verification that can be compared as part of RUSH algorithm family by [10]. In addition, *Freenet* [6] research model offers a way out for publication, replication and retrieval thereby keeping identity secret. This can also be considered to integrate as our future work along with [4] for faulty files replacement. Furthermore, multi-server queue model from [7] and reliability focus on [3] gave us a good lead to relate and model DDS system. Furthermore, although the algebraic signature method along with a hash function [15] was taken into consideration initially, this was beyond the scope of our objectives. Clearly, to some extent, all the efforts above are related to our DDS project and provide a rich research background.

The solution methods are explained using top down approach. Initially, an overview of the DDS is formulated in *section-III.A*. Subsequently, activity flow in all the subsystems are explained in *section III-B* by using activity diagram. Successively, implementation design in *section III-C* then refined in yet greater detail into class design level. Finally, pseudo code in *section III-D* breaks down the overall system flow and methods into specific smaller segments to explain the implementation flow.

Overview of the System

In order to model DDS system, we have followed the divide and conquer strategy. Thus, primarily we have designed the top-level overview for the system. The DDS considers two scenarios for file saving and same for file retrieving. Firstly, a file saving request comes to DDS system, the Manager DDS server checks whether the file size is smaller than one machine's free space. If one machine has enough space, the file is stored in that machine and server gives a file id for future retrieval. Similarly, a user can request for a saved file to the system using the file id that was given during file saving operation. **Figure -2**, gives a complete overview of the described scenario

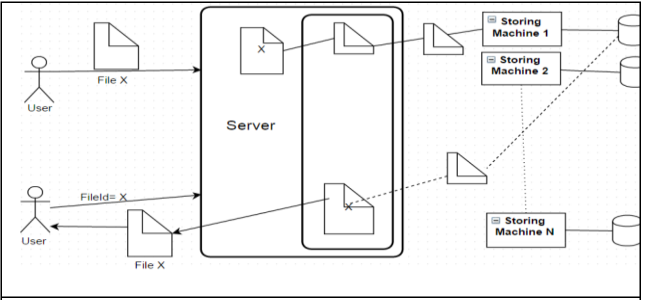

Figure 2: Overview of DDS –File storing to and retrieving from single Machine

On the other hand, considering the scenario in *Figure-3*, if the system does not have enough free space to save a single file, DDS checks how many machines can accommodate this file and based on that the file is fragmented and saved to multiple computer machines. For instance, if the file size is greater than Y number of machine's free size, however, less than X number of machine's free size then the file is fragmented in X numbers. Also, the size of file part is equivalent to available free space on each selected computer and then stored into that computer. Also, all the file parts id has similar id except for identical suffix. Yet server gives only one id thereby omitting file segmentation matter to the user. Likewise, whenever a user request for a the file using the given id, server look for the file ids that starts with similar id and combines all the file parts to provide a single complete file that user saved. Finally a single file is provided that was requested.

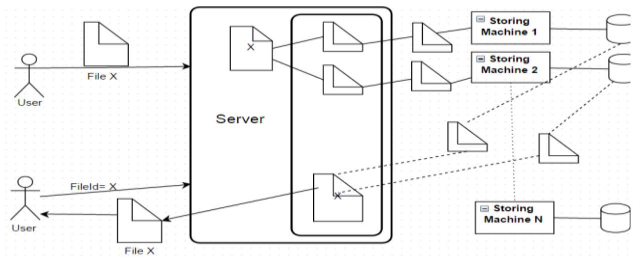

Figure 3: Overview of DDS - File parts storing to and retrieving from Multiple Machines

Activity within the DDS

Building up from model presented in overview section III.A, we then have gone through more detail and designed activity diagram (Figure-4) that shows the overall activity flows for DDS system. At first, system starts with all the necessary variables initialization and then generates first event. Since events change the system status and we are following event driven discrete simulation, whenever an event occurs, we store it to future event list first before processing. During the event processing, the event is retrieved from future event list to check the event type. As stated earlier there might be only two types of event such as fileSave and fileRetrieve.

If the event type is a fileSave type, then the file size is checked with a single machines free size. Based on the available spaces in that machine file is stored, file size is deducted from machine's total space and the file id and machine id is stored in a DDS bigtable for future reference before printing the confirmation to user. Also, as soon as the file saving operation is done, a random number of new file saving events are generated (i.e. between *1 and N*). However, if the file size is bigger than one machine's free space, we calculate how many machines free space is the right fit for this file and make splits(same as number of machines) of that file with a unique id for each part of that file. Also, the size of file part is equivalent to the available free space on each machine.

Let us consider we have ***n*** number of machines in DDS system and machine's free space is represented with x_i. If ***j*** represents the number of machines (of those free space is suitable to store the file) and ***i*** represent the id of machines from ***1** to **j.***

Thus, total free space to save a file in multiple machines is

$$\sum_{i=1}^{j} x_i$$

Also, the number of splits of a file is ***j*** and the size of each split is x_i. Based on the calculation using the method above, we then save each part of the files to selected machines as well as update the free spaces by deducting the size of file's part. Finally, file part id and storing machine if of that part is stored to DDS bigtable for future reference and retrieval operations. However, for a file size is bigger than all machines size, DDS will print a "not enough storage" message.

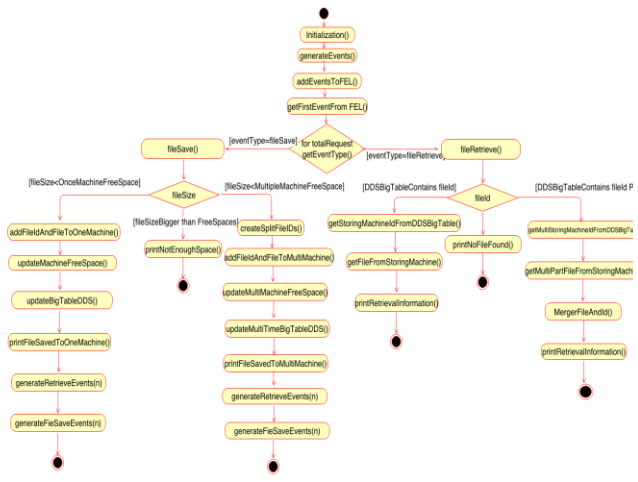
Initialization()
generateEvents()
addEventsToFEL()
getFirstEventFrom FEL()
for totalRequest getEventType()
[eventType=fileSave]
[eventType=fileRetrieve]
fileSave()
fileRetrieve()
[fileSize<OnceMachineFreeSpace]
fileSize
[fileSize<MultipleMachineFreeSpace]
[fileSizeBigger than FreeSpaces]
[DDSBigTableContains fileId]
fileId
[DDSBigTableContains fileId P
addFileIdAndFileToOneMachine()
updateMachineFreeSpace()
updateBigTableDDS()
printFileSavedToOneMachine()
generateRetrieveEvents(n)
generateFieSaveEvents(n)
printNotEnoughSpace()
createSplitFileIDs()
addFileIdAndFileToMultiMachine()
updateMultiMachineFreeSpace()
updateMultiTimeBigTableDDS()
printFileSavedToMultiMachine()
generateRetrieveEvents(n)
generateFieSaveEvents(n)
getStoringMachineIdFromDDSBigTable()
getFileFromStoringMachine()
printRetrievalInformation()
printNoFileFound()
getMultiStoringMachineIdFromDDSBigTa
getMultiPartFileFromStoringMach
MergerFileAndId()
printRetrievalInformation()

Figure 4: Activity in DDS - File storing to and retrieving from DDS system

In contrary, if the event type is a fileRetrieve type then DDS system manager looks for exact file id match in DDS bigtable and based on id match, system gets the file from stored machine. Yet, if the file id matches as a part of DDS bigtable key string, we consider it as a part of the same file and collect all parts to merge it as a single file and provide it back as a successful message.

For example, if file id part is found in ***j*** number of machines, ***i*** represent the id of file part and ***f*** denotes file id and $\boldsymbol{p_i}$ is the part indicator with unique suffix,

$$\text{File id is fragmented to} \quad \sum_{i=1}^{j} f + p_i$$

Finally, a "No File Found" message is printed if the file id does not match with the requested file id.

Implementation Designs

Moving towards implementation, we have segmented overall implementation in four different classes. Event, StoringMachine, QueueOperaiton and ManagerDDS class. All the classes contain different properties to facilitate the services in DDS system that are explained a bit more detail in the following:

Event

Event class facilitates events related to file saving and retrieval operation. Based on the request, this class generates either saving event or retrieval event. Also, this class holds the events properties and methods to get and modify properties. For example, essential information such as event arrival time, file size, files type is the property of this class along with other properties.

StoringMachine:

StoringMachine class holds the basic structure of all machines to be added in the DDS system. Also, it keeps the information of stored files in any computer, including what type of file is acceptable to any particular machine.

QueueOperation: the most significant class of the DDS system is QueueOperation class. This class established the discrete event simulation feature to our system. Besides, the most distinctive and challenging feature of this class is implementation of priority queue for event queue. In particular, this class ensures events are prioritized based on event total service time (i.e. sum of event arrival and service time) and therefore event with minimal time spent occurs first. However, if both the event occurs at the same time, either of them can occur first similar to the case in real life. For example, let us consider two events e_1 and e_2 where e_1 arrived at t_1 time and service time is s_2 and e_2 arrived at t_2 and service time s_2. Thus

If, $t1+s_1 < t_2+s_2$ then e_1 will occurs before e_2

If, $t1+s_1 > t_2+s_2$ then $e2$ will occurs before e_1

If, $t1+s_1 = t_2+s_2$ then either e_1 or e_2 will occur first

ManagerDDS

The central class for the DDS system is ManagerDDS that enables the event generation, items addition in future event list, file saving operation including file splitting and retrieval operation together with file merging based on event request type. Also, this class holds the method that adds an event to eventQueue and removes events after successful completion of that event request. In addition, the statistical reports and simulation results are also produced from this class

Pseudo code for DDS

In order to implement the DDS system simulation model, we have considered two main events request fileSave and fileRetrieve where events are generated at a random time. The basic logic is to check which event occurs first as stated below. The simulation runs until there is request process reach up to total number of requests.

Pseudo code for entire DDS system

```
Sub ManagerDDS
        Call initialize ()
        Call
generateEvents(fileid,requestType,NumberofEventsToGenerate)
        Add event to Future Event List

        Insertion operation to Queue
        LOOP UNTIL number of requests is not equal to total Requests
                get event from future Event List
                call    setEventserviceTime(eventArrivalTime    +
randomTime)
                IF event type is equal to filesave THEN
                 call fileSave(event) for numberOfCopiesRequired to
save
                        ELSE
                                call fileRetrieve(event)
                        ENDIF
                        Add Event to the eventQueue

        generateNewEvents(fileid,requestType,NumberofEventsToGenerate
)
                ENDLOOP

        Removal operation from Queue
                LOOP UNTIL Queue has event in head
                IF the queue is not null THEN
                        get the head event from the Queue and
remove
                        ENDIF
                        Sleep removal process for 2 to 5 seconds
                ENDLOOP
        Start event inseriton operation to the queue
        Sleep the removal operation for 5 seconds
        Start the removal operation from the queue
End Sub ManagerDDS
```

The initialize method sets all the initial values and generateEvents produces the first event using some input parameters (i.e. table 1). Once the event is created, it is added to the future event list (FEL). After that insertion operation gets the event from FEL and set some service time thereby calling setEventserviceTime method. Subsequently, it checks the eventType either 1 (fileSave) or 2 (fileRetrieve). In case of fileSave, the system calls sub method fileSave and sent event object as parameter. For the same event this fileSave runs for ***N+1*** time to ensure system resiliency as we have considered that at a time there might be ***N*** faulty computers in our DDS system. However, upon on fileRetrive request, sub method fileRetrieve is called with event object as parameter. After that, the event is added to a priority queue named eventQueue that orders events based on total service time (arrival+service). Finally, generateNewEvents generate new fileSave events (i.e. ***1 to n*** number).

Correspondingly, another process performs the remove operation from the eventQueue after some time (i.e. 2 to 5 seconds) delay. And this operation continues until any item remains in the eventQueue.

Pseudo code for a file saving method in DDS system

The fileSave sub method work based on three conditions. Firstly, it compares the fileSize with one machine's free space and fileType with storing node's acceptable type. If this condition satisfies, the fileId and file itself is stored to that machine, update the available free space on that machine by subtracting fileSize and the information is stored in DDSBigtable for retrieval purpose. To finish, it generates some retrieval events as the user might want to retrieve file from DDS system in near future. However, if the first condition does not satisfy, second condition checks whether this file is less than total spaces of DDS system. After that, some machines (of same fileType) free spaces are combined until it becomes bigger than the file size. Subsequently, file split ids and file split (same as machin's free space) are created. Afterwards, filepartId and file added to those selected storing machines at the same time information added to DDSBigtable, machine's free space is updated, and some retrieval events generated. Finally, if none of the first two conditions satisfies, system informs about no nstorage availability.

```
Sub fileSave(Event)
                IF fileSize is LESS than One Storing Machine's Free
space
        AND FileType is equal to Storing machine's acceptable tile type
                Add fileId and File to this Storing Machine
        Subtract this machine's File size and update free space of this machine
        Update central storage-DDSBigtable with FileId and Machine ID
        Generate random file retrieval events

                ELSEIF total free space is greater than file size
LOOP UNTIL space bigger than filesize found in total number of Machines
IF fileType is equal to Storing machine's acceptable file type
        AND machine does not have part of the fileId THEN
        Add machineId and filespace in temporary array subMachineSpace
        IF subTotalFreeSpace is greater than File Size THEN
                Exit from loop
                ELSE
                continue adding machineId and file space in array
                ENFIF
        ENDIF
        ENDLOOP
        LOOP UNTIL subMachineSpace array has item
        Add filepartId and File to each Storing Machine
        Subtract this machine's File size and update free space of this machine
```

```
            Update central storage-DDSBigtable with FileId and Machine ID
            ENDLOOP
            Generate random file retrieval events
            ENDIF
                        ELSE
                        Print System does not have enough Space
            ENDIF
END Sub
```

Pseudo code for a file Retrieving method in DDS system

The Sub method fileRetrieve also looks for three situations such as file is in a single machine, multiple machines or no file in the system. At first, bigtableDDS is checked if it has the given file id (specific string) in its key list. If it is found, then that key is used to get the machine and original file location. As soon as the file is retrieved the successful message is printed thereby exiting from this operation. However, based on first condition failure, the file id matching of the same copy was looked in filePartIds form bigTableDDS keys to get the original file location and file from multiple machines. And finally, all "ids" are combined to print the merged file retrieval information. If none of the condition satisfies, a "file not found message" is printed.

```
Sub fileRetrieve(event)
            IF file is in single machine THEN
                        LOOP UNTIL all items in the bigTableDDS is fetched
                        IF the bigTableDDS contains fileId as Key
      AND fileId ends with a specific string THEN
            Get the machine Id from Bigtable DDS Key using file ID
            Get the file from this machine using machine Id from bigTable
                        Print the file retrieved from a single machine
                        Break out of the Loop and return to calling method
                        ENDIF
            ENDLOOP
            ELSEIF check if file is in multiple machine THEN
            LOOP UNTIL all items in the bigTableDDS is fetched
            IF the bigTableDDS key has a string similar to filepartId THEN
            Search for a key that matches to fielPartId in bigTableDDS.
            Get the machine Id from Bigtable DDS Key using filePardId
            Get the file from this machine using machine Id from bigTable
            Combine all part id to merge all file
                        ENDIF
                        ENDLOOP
            Print the file retrieved information from a Multiple machine
            ENDIF
            ELSE
            Print the file Not found information
            ENDIF

END Sub
```

Experimental Setup

All of the investigations were performed on several PCs with similar configuration of 3.30 GHz core i3 CPU and 3GB of RAM, running 32-bit windows-7 operating system(Durham university student computers) and a PC with configuration of 2.30 GHz core i3 CPU and 4GB(3.41 usable) of RAM, running 32-bit windows-7 operating system. To observe the DDS system performance, a thorough unit testing was performed during implementation and the simulation model was tested using several User Acceptance Test[3] (UAT) input parameters. System was tested using data from Durham University CIS. The CIS asset manager (Leggett, 2014), confirmed that there are 7030 computers in Durham university each machine has approximate space of 320GB storage. As the university computer users are given 400MB space from a mapped J: drive, all the user machines take very minimum space for operating system and some basic applications. Based on a rough estimation and survey from the CIS staff we deduce that this basic usage takes maximum 30% of the total space in HDD. Thus, we have set up the environment based the disk size and usage; therefore, each machine disk size is between 96 and 320 in our simulation model. Also, data in *Table-1* and *Table-2* are considered for different level UAT.

[3] User acceptance test (UAT): http://softwaretestingfundamentals.com/acceptance-testing/

Unit Test and UAT Results

As mentioned in the section III (solution part), there are two types of requests (fileSave and fileRetrieve). While the simulation model starts, a file saving request is generated and at the end of file saving operation another file saving event will be generated. During the operation file saving is performed for 3 times (considering at a time there might be 2 faulty computers in the system) and each file saving operation produces few arbitrary (between 1 and 5) retrieval events bearing in mind that user might want to retrieve the same file in near future (i.e. after some casual simulation time in DDS model).

While we tested the system using the data from *Table-1* and *Table-2*, we found some adequate results that reflect the accomplishments of DDS objectives. Table-1 values are generic as input parameters for all the tests. However, each row shows different result based on different test methods. For example, first row represents system with 20 machines and each machine with 1GB free spaces where user wants to save a file of size minimum 25GB. As the accumulated free spaces of the whole system is 20GB, no file saving or retrieve operation was successful. However, the system checks all the necessary condition which required total wall clock time of 6.3 sec to process one request.

Table 1: Input parameters for Unit Test and UAT

Request arrival time Range	1-15
Service Time Range	5-15
Simulation Time	Integer value
No of Copies for replication	3
Request Type	1 for File arrival 2 for File Retrieval
For each File retrieval request generated	1-5

Table 2: DDS System Response Based after Unit Test and UAT Result

Sl	Total Requests	No of Machines	Free Disk space range	File Size Range	Total file copies (n=3) Saved	Total file copies Retrieved	Total Simulation Time	System Time in seconds
	1	20	1-1	25-50	0	0	17	6.3

	10	20	96-320	20-50	18	4	104	32.29
	200	100	50-100	1-50	329	89	1536	802.32
	2000	1000	96-320	20-50	3860	712	16061	16308.47

Moving to a different scenario, we then tested for 10 requests in 20 machines (each of them with free space range between 96GB and 320GB) for file size in the range of 20GB to 50GB (second row of *Table-2*). Due to space availability in all the machines to accommodate files, all the individual machine stores requested whole files system. As we have considered the replication factor to ensure system resilience and establish a fault tolerant system, each file was copied three times and there for total 18 files (i.e 6 files each for 3 times) saved and four retrieval operations performed within 32.23 seconds.

On the other hand, while we ran the test using third row as input of *Table-2*, result demonstrates that some files are saved in single machine and rest are fragmented to fit machines size and saved in multiple machines with a preceding strings $\mathbf{p_0 \ldots p_n}$ for unique identification**.** Likewise, during the retrieval operation single files are retrieved as soon as it was found regardless of which copy it is. For example, we have found some files were retrieved from copy-2(c1), some of them are from copy-1 (c0) and rest copy-3(c2). However, in case of fragmented files, fragmented parts are collected from their original stored machines location and ids are merged for combined output. The output matches with the saved split file id and therefore demonstrates that the DDS system can identify the single and segmented file's location. The same file selection policy applies here like single whole file.

Changing the complexity, we then tested DDS model with Durham University CIS (Leggett, 2014) data where input parameters are same as 4th row of *Table-2*. Thus, DDS simulation model successfully processed 3860 file save 712 file retrieve operations in thereby requiring 16308.47 seconds of system time. Compare to data in 3rd row of *Table-2*, 4th row has 10 times more requests which might generally lead us to think it will take 10 times more time to process the whole request lot. However, as more requests are processed, more files are saved and thus system free spaces are filled up. As a result, far ahead file saving requests splits the files to store a single file in multiple machines. Similarly, file retrieving request takes additional time to merge those split files. Therefore, this causes the gradual increase in required system time (16308.45 seconds) to process 2000 requests.

In addition, for all the saving operations, system checks the given file's type with storing machine's acceptable file types, therefore, saves only if it satisfies the condition. For instance, the system output from 4th row of *Table-2* data demonstrated some file are saved only if the file type and storing node's acceptable file type are same. More specifically, some file of "GEO" file types are store to those machine that only accepts "GEO" types file. Similarly, output was generated for other types like MAT, ECS, PHY, ANT etc. Therefore, the DDS system imposes the user policy to allow specific files in storing node.

Performance Analysis for Splitting and Merging

In order to ensure the smooth event processing, we had to put some time delay between event addition and removal. Thus, although the methods perform tasks in milliseconds, the actual wall clock time for a complete event processing is in seconds

While the DDS system was tested using the input parameters from *Table-3*, and the results are compiled to deduce a conclusion regarding the system wall clock's response for file saving and retrieving operation. Also, the environment was designed mainly to manipulate the system in a way (allocating minimum free space to a storing machine) so that it has to split files to save and merge to retrieve after few operations and thus the system goes through entire process (i.e. whole file save, file splitting and merging along with storing user preferred file type.)

Table 3: Input parameter unit in GB	
Machine Free Space Range	10-50
File Size Range	10-25
No of Requests	50
No of Machines	30

Based on few sample outputs, the result summary is displayed in the *Figure-5*. Clearly, stroring and retrieval time scale together with file size and storing machine's free space rather than depending on only one factor. Considering the first request with file id-1, a big difference can be seen between file saving (28 milliseconds) and retrieving (3 milliseconds) operaion .

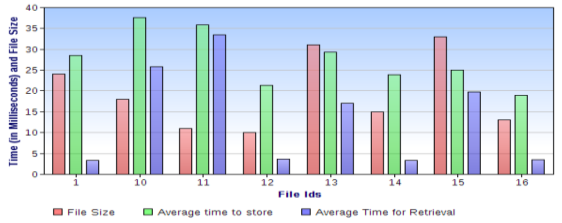

Figure 5: Performance analysis for file Splitting and Merging operation

As the system saved three copies of each file for a file saving request, one copy is saved in a single machine and rest saved in multiple machines after splitting which required extra time. However, during the retrieval operation whole copy in single machine was found. As a result, retrieval time for file id-1 is much faster. In contrary, time consumption seems similar for file saving and retrieval to and from multiple machines (e.g. file id-11 and file id- 15 in the *Figure -5***)**. This tendency similar in case of single machine's file storing and retrieving.

Also, as the the system chceks the file types with storing machine's acceptable type, in some cases files have to be fragmented to store in multiple machines of similar file type thereby omitting single machine with available free spaces. The situation is same for a machine holding a copy of same file, thus, influences to find a machine that does not have a copy of the file is trying to be stored. As a result, depending on the situation, the time requirement for the same file storing or retrieving might vary in the DDS system, however, the difference is not that significant (in seconds).

The DDS project was proposed with clear goal and objectives and section-III and section-IV justifies the overall achievement by stating design model and results. Firstly, a thorough feasibility analysis was performed through the content analysis of related literature review.

Also, a basic discrete event simulation model was developed which then lead us to develop simple file storing and retrieval method to execute in networked nodes. We felt that a proper understanding of discrete event simulation will ease our simulation development process. Thus, the problem was segmented into two parts. Initially, a simple discrete event simulation model structure was designed and developed to understand the discrete events flow within the structure. Then, the file saving and storing methodologies were developed to facilitate the file saving and retrieving operation and thus both parts were integrated together to perform DDS system activities. Therefore, we feel that we have gained true benefit of divide and conquer method here.

Secondly, the complexity in simulation model design was changed (*Figure-2*) to achieve the intermediate level objective. As a result, the file splitting feature is implemented to facilitate the file storage in multiple machines bearing in mind that one file size might be bigger than one machine's available free space. Besides, the simulation model can identify the segmented files location to provide the combined file for retrieval operation. Moreover, to ensure the DDS is a fault tolerant system, we have created ***N*** copies of each files considering ***N-1*** faulty computer in the system. Thus, DDS can retrieve files if at least one storing node with file copy is alive and that ensures more resiliencies over having just only one copy in the system.

Furthermore, there is an option for all the storing computers in the DDS system to choose what kind of file it is going to accept for storing. While a new storing node joins to the DDS network, it can specify its acceptable file. Based on that the DDS system manager cross-checks the file type matching and then stores upon condition verification. Thus, DDS system simulation achieved all objectives by feasibility analysis, file splitting-retrieving, replication policy and allowing user preferences for file storage.

DDS Request Processing Performance Analysis

To check how the DDS system performs as the request number increases in the same environment, we set up the system with the parameter on *Table-4*.

Table 4: DDS request processing input parameters	
Machine free spaces	96-320
File Size	5-70
Total no of Computer	7030

Clearly, *Figure-6* states that the system wall clock time rises as the number of request increases. Although the wall clock timing trend seems similar for the first 1 to 200 event requests (average processing time 4.1 seconds), a tremendous change was noticed while 2000 requests were processed (average processing time 8.15 seconds) in DDS system.

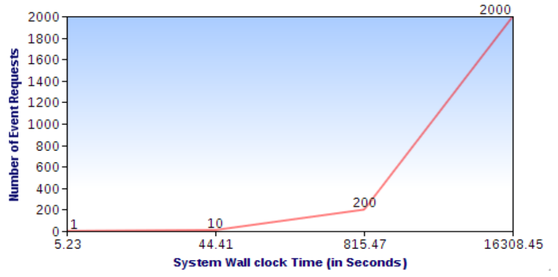

Figure 6: DDS Request processing performance

Based on this inclination in results we can conclude that, as number of request increases, the far ahead file saving (i.e. save after splitting) and retrieval (i.e. collect and merge) requests need multiple computer's invocation (since free spaces are filled up by previous file requests). As a result, this leads to extra time consumption to process well ahead requests in the DDS system. In summary, to evaluate the DDS system performance, primarily the complexity level increased in a chronological order and performance observed. Also, the system was tested by creating random virtual scenarios to verify the results. Finally, system responses accomplished all proposed objectives.

The DDS Verdict

File processing time scales with the file and storing machine size. If the system needs to invoke multiple machines, the required time is greater. Although the time difference in DDS varies between file saving and retrieval, the difference is not that significant (i.e. in seconds).

DDS follows either "all-or-none" strategy for merging and splitting operations. For example, if DDS lose any storing node in the middle of file splitting or merging operation then none of the file part will be saved.

Although the simulation model runs successfully and justifies the feasibility and distributed disk store, during implementation of actual application, some external factors might have certain impact on system performance. Network reliability, file transfer rate and storing machines speed etc.

As mentioned in file replication model (section-III), we assume that there might be ***N*** faulty computers in the DDS system and thus we consider ***N+1*** copies of file for system resiliency. However, the system might be vulnerable for an extreme situation like earthquake or power surge that depends on a geographical or uncertain factor and almost all the systems exist are in danger to this condition.

DDS will require participating computers to be powered up to ensure the file availability. Thus, this will require very durable power source to ensure the system accessibility.

This research paper has presented an overview of distributed disk storage architecture and simulation model that we have developed to store file (i.e. single or fragmented) and retrieve the same in distributed networked environment. It also critically analyzed the related literatures.

Our experiments show that, the DDS simulation model can manage networked computers. Also, the system demonstrates successful file storing with replications to ensure system resiliency along with a track record for retrieval in distributed computers. Thus, integrating feature for user preference, the model substantiates that the DDS system is feasible for a large organization.

We found that, by implementing DDS system in a large organization like University, around 70% (i.e. approximately 1575TB in Durham University) of total disk spaces can be used as storage that are sitting idle. Therefore, we believe the DDS system is most cost-effective solution (i.e. only system maintenance cost) to meet the exponential need for storage of research data as university already has these storages installed. Also, every new node addition for the university will add space to the DDS environment

Our performance analysis results based on UAT data shows that, multiple computers invocation requires more time to save or retrieve a file over a single computer. In addition, the average processing time increases with the rise in number of requests.

During the implementation, the idea of intruder's intervention also arose. If an intruder tries to access the system and corrupt any specific file, how the system might respond? An initial thought was given in the system design to incorporate a method similar to signature scheme (Xin et al., 2003) that resembles a hash function to detect any interference in stored file. Although we have considered the fact to integrate in our system, as it was beyond our project scope and due to time limitation, we could not explore all issues, and this is considered as one of our potential extension.

Since reliability is the main concern and considering early disk failure detection (Schwarz et al., 2004) factor, a corrupted file detection method is chosen as a planned future work to replace corrupted files with replicated copy from another subset computer thereby using DDSBigtable reference. Although we thought about a central back up of at least one copy of each file, this this turned out to be a costly process which conflicts with DDS system's interest.

To ensure the system availability all the storing machines in DDS system required to be switched on at all time. However, this requires uninterrupted power supply and therefore, the maintenance cost will become higher. To overcome this problem we are have a plan to model a "Reliability Group of Storing Machines (RGSM)" where the system will make sure system with one specific group (i.e.) of storing machines are filled first and at least one copy of each file is always available. As a result, other groups can take turn to be in hibernation mode. Though this is beyond the scope of this paper this is classified as another potential extension of DDS system.

The key motivation for DDS simulation model was to test the feasibility such system before going for actual implementation. Based on the experimental results we believe that distributed disk store is a realistic project and can be established in networked computers.

In conclusion, DDS can be a very powerful resource for any large organizations like universities as this utilizes the unutilized resources and as a final point, DDS offers rebate from your old machines!

ACKNOWLEDGMENTS

Dr Stephen McGough, who undertook to act as my supervisor despite his many other academic and professional commitments. His wisdom, knowledge and commitment to the highest standards inspired and motivated me. Most importantly this worth mentioning whenever I was stuck in any problem, he had the answer to find a way out. *Tom Friedetzky*, my associate supervisor for his insightful comments on DDS project and the time he has given in the DDS project evaluation. *Durham University CIS team*, for being so supportive thereby providing the necessary data to design the DDS system model. Durham University and Commonwealth scholarship commission for believing in my potential and providing me scholarship to pursue my degree at Durham University. Without the support from both it would be extremely difficult for me to complete my degree

REFERENCES

[1]. Barnes, D. J. and Kölling, M. 2012. Objects first with Java. Boston: Pearson.

[2]. Birtwistle, G. M. 1979. DEMOS, a system for discrete event modelling on Simula. London: Macmillan.

[3]. Bolosky, W., Douceur, J., Ely, D. and Theimer, M. (2000). Feasibility of a serverless distributed file system deployed on an existing set of desktop PCs. 28(1), pp.34--43.

[4]. Buyya, R., Cortes, T. and Jin, H. (2009). A Case for Redundant Arrays of Inexpensive Disks (RAID). Wiley-IEEE Press.

[5]. Chang, F., Dean, J., Ghemawat, S., Hsieh, W. C., Wallach, D. A. and Burrows, M. 2010. Tushar Chandra, Andrew Fikes, and Robert E. Gruber. Bigtable: a distributed storage system for structured data. In Proc. USENIX Symposium on Operating System Design and Implementation (OSDI'06), 2006. DATE: 1 February 2010.

[6]. Clarke, I., S, Berg, O., Wiley, B. and Hong, T. W. 2001. Freenet: A distributed anonymous information storage and retrieval system. pp. 46--66.

[7]. de Mesquita, M. and Hernandez, A. (2006). Discrete-event simulation of queues with spreadsheets: a teaching case. pp.2277--2283

[8]. Fujimoto, R. M. 2000. Parallel and distribution simulation systems. New York: Wiley.

[9]. Ghemawat, S., Gobioff, H. and Leung, S. 2003. The Google file system. 37 (5), pp. 29--43.

[10]. Honicky, R. and Miller, E. L. 2004. Replication under scalable hashing: A family of algorithms for scalable decentralized data distribution. p. 96.

[11]. Leggett, D. (2014). INC000000415835. [email].

[12]. Mesnier, M., Ganger, G. and Riedel, E. 2005. Object-based storage: pushing more functionality into storage. Potentials, IEEE, 24 (2), pp. 31--34.

[13]. Pooley, R. J. 1987. An introduction to programming in SIMULA. Oxford: Blackwell Scientific Publications.

[14]. Schwarz, T. J., Xin, Q., Miller, E. L., Long, D. D., Hospodor, A. and Ng, S. 2004. Disk scrubbing in large archival storage systems. pp. 409--418.

[15]. Schwarz, T. S. and Miller, E. L. 2006. Store, forget, and check: Using algebraic signatures to check remotely administered storage. pp. 12--12.

[16]. Wang, F., Br, T, S. A., Miller, E. L. and Long, D. D. 2004. OBFS: A File System for Object-Based Storage Devices. 4 pp. 283--300.

[17]. Xin, Q., Miller, E. L., Schwarz, T., Long, D. D., Br, T, S. A. and Litwin, W. 2003. Reliability mechanisms for very large storage systems. pp. 146--156.

[18]. Zhang, Y., Wu, Y. and Yang, G. 2012. Droplet: a distributed solution of data deduplication. pp. 114--121.

[19]. Zhu B., Li K. and Patterson H. 2008. Avoiding the Disk Bottleneck in the Data Domain Deduplication File System.