A Study on Reliability and Data Placement in Distributed Systems

Thesis submitted by
Hindol Bhattacharya

Doctor of Philosophy (Engineering)

2022
School of Education Technology
Faculty Council of Engineering and Technology
Jadavpur University Kolkata, India

List of Publications, Presentations and Patents based on this Thesis

INDEX NO.: 152/17/E

1.	**Title of the thesis:**	A Study on Reliability and Data Placement in Distributed Systems
2.	**Name, Designation & Institution of the Supervisor:**	Prof Matangini Chattopadhyay

3. **List of Publications:**

a. **Journal:**

i. H. Bhattacharya, S. Chattopadhyay, M. Chattopadhyay, and A. Banerjee, "Storage and bandwidth optimized reliable distributed data allocation algorithm," International Journal of Ambient Computing and Intelligence (IJACI), vol. 10, no. 1, pp. 78–95, 2019.

b. **Conferences:**

i. H. Bhattacharya, S. Chattopadhyay, and M. Chattopadhyay, "NS3 based HDFS data placement algorithm evaluation framework," in 2017 International Conference on Computer, Electrical & Communication Engineering (ICCECE), pp. 1–8, IEEE, 2017.

ii. H. Bhattacharya, S. Chattopadhyay, and M. Chattopadhyay, and A. Banerjee, "A novel intelligent modeling of storage and bandwidth constraints in distributed storage allocation," in International Conference on Computational Intelligence, Communications, and Business Analytics, pp. 336–346, Springer, 2017.

Continued...

iii. H. Bhattacharya, S. Chattopadhyay, and M. Chattopadhyay, “Problems with replica placement using data dependency in scientific cloud workflow,” in 2018 Fifth International Conference on Emerging Applications of Information Technology (EAIT), pp. 1–4, IEEE, 2018.

iv. H. Bhattacharya, A. Bhattacharya, S. Chattopadhyay, and M. Chattopadhyay, “Lda topic modeling based dataset dependency matrix prediction,” in International Conference on Computational Intelligence, Communications, and Business Analytics, pp. 54–69, Springer, 2018.

v. H. Bhattacharya, M. Chattopadhyay, and S. Chattopadhay, “A case for splitting a file for data placement in a distributed scientific workflow,” in 2021 IEEE 12th Annual Information Technology, Electronics and Mobile Communication Conference (IEMCON), pp. 1058–1063, IEEE, 2021.

vi. H. Bhattacharya, A. Bhattacharya, M. Chattopadhyay, and S. Chattopadhyay, “Determining threshold for partitioning a dependency graph in replica prefetching in distributed systems,” in 2021 Sixth International Conference on Research in Computational Intelligence and Communication Networks (ICRCICN - 2021), p. Article in Press, Springer, 2021.

4. **List of Patents:** None

5. **List of Presentations in National/International/Conferences /Workshops:**

a. **International Conferences:**

i. 2017 International Conference on Computer, Electrical & Communication Engineering (ICCECE)
Paper- NS3 based HDFS data placement algorithm evaluation framework

ii. 2017 International Conference on Computational Intelligence, Communications, and Business Analytics (CICBA)
Paper- A novel intelligent modeling of storage and bandwidth constraints in distributed storage allocation

Continued...

iii. 2018 Fifth International Conference on Emerging Applications of Information Technology (EAIT)
Paper- Problems with replica placement using data dependency in scientific cloud workflow

iv. 2018 International Conference on Computational Intelligence, Communications, and Business Analytics (CICBA)
Paper- Lda topic modeling based dataset dependency matrix prediction

v. 2021 IEEE 12th Annual Information Technology, Electronics and Mobile Communication Conference (IEMCON)
Paper- A case for splitting a file for data placement in a distributed scientific workflow

vi. 2021 Sixth International Conference on Research in Computational Intelligence and Communication Networks (ICRCICN)
Paper- Determining threshold for partitioning a dependency graph in replica prefetching in distributed systems

PROFORMA – 1
Statement of Originality

I **Hindol Bhattacarya** registered on **July 11, 2016** do hereby declare that this thesis entitled "A Study on Reliability and Data Placement in Distributed Systems" contains literature survey and original research work done by the undersigned candidate as part of Doctoral studies.

All information in this thesis have been obtained and presented in accordance with existing academic rules and ethical conduct. I declare that, as required by these rules and conduct, I have fully cited and referred all materials and results that are not original to this work.

I also declare that I have checked this thesis as per the "Policy on Anti Plagiarism, Jadavpur University, 2019", and the level of similarity as checked by iThenticate software is 6 %.

1. Hindol Bhattacharya 05/07/2022.
Signature of the Candidate
Date

Certified by Supervisor(s):
(Signature with date, seal)

1. Matangini Chattopadhyay 05/07/2022.

Professor
School of Education Technology
Jadavpur University
Kolkata - 700032

PROFORMA – 2
CERTIFICATE FROM THE SUPERVISOR

This is to certify that the thesis entitled "Study on Reliability and Data Placement in Distributed Systems" submitted by Shri**Hindol Bhattacharya**, who got his name registered on **July 11, 2016** for the award of Ph. D. (Engg.) degree of Jadavpur University is absolutely based upon his own work under the supervision of **Prof. Matangini Chattopdhyay** and that neither his thesis nor any part of the thesis has been submitted for any degree/diploma or any other academic award anywhere before.

1. Matangini Chattopadhyay 05/07/2022
Prof. Matangini Chattopadhyay
School of Education Technology
Jadavpur University

Professor
School of Education Technology
Jadavpur University
Kolkata - 700032

Acknowledgements

It is with great pleasure that I would like to express my gratitude and indebtedness to my guide Prof. Matangini Chattopadhyay, School of Education Technology, Jadavpur University, Kolkata, for her continuous guidance, valuable advice and constant encouragement through the research work culminating in this thesis. I would also like to thank her for extending support in regard to my administrative needs as Director, School of Education Technology. This thesis would have remained incomplete, if not for Prof. Chattopadhyay's kind guidance in this regard.

I would also like to take this opportunity to thank Prof. Samiran Chattopadhyay, Department of Information Technology, Jadavpur University, for extending his guidance together with Prof. Matangini Chattopadhyay in the various works of my thesis. My gratitude is also due to Prof. Arnab Bhattacharya, Department of Computer Science and Engineering, Indian Institute of Technology, Kanpur, for providing me with the internship opportunity at Indian Institute of Technology, Kanpur, during my period of Ph.D. research and for his assistance in my research work.

I also thank my collaborators in various works- Dr. Kripabandhu Ghosh of Indian Institute of Science Education and Research, Kalyani and Dr. Avishek Banerjee, School of Computer Science Engineering and Technology, Bennett University, Noida. Being seniors researchers, their advice has been of great assistance in my work.

The ackowledgement is incomplete without expressing my gratitude to the administration, faculty members and staff of the School of Education Technology, Jadavpur University and the officers and staff of Jadavpur University for being ever helpful in assisting me with my administrative needs.

Last but not the least, I would like to express my gratitude towards my parents who supported and motivated me through my education journey from Nursery to Ph.D. The support and assistance provided by my friends, seniors and fellow researchers at Jadavpur University also deserve its due credit; especially that of Dr. Susovan Jana of the

Department of Production Engineering, Jadavpur University, who has been a fellow traveler in this journey of research and education, since my MTech years.

Finally I would like to thank Jadavpur University and Department of Science of Techology, Government of India for financially supporting me throught my PhD research years through State Govt Fellowship and INSPIRE-AORC fellowship scheme respectively.

Jadavpur University
2022

Hindol Bhattacharya
Hindol Bhattacharya

Contents

Contents

Abstract

Cloud computing has been used for business applications for quite some time with considerable success. The rapid adoption of cloud services such as Microsoft Azure, Amazon AWS, Google Cloud platform, etc. is a testament to the rise of cloud services. The rapid rise of cloud computing can be attributed to the conveniences provided by it, such as infinite and dynamic resource availability, managing IT resources, limiting the cost to only the services used, etc. Cloud computing has been utilized by non IT businesses which use computation to enhance their business, while the core business may be in a different domain. Traditional use of IT services would require over or under provisioning of computational and associated resources by these businesses; thereby increasing cost. Also, expert human resource would have to be recruited to maintain these resources; adding to the cost and human resource management problems. In cloud computing, these costs are significantly reduced and the problems get eliminated.

Traditionally, the scientific community has utilized the services of high performance computing (HPC) or Grid computing (HPC built by aggregating power of a distributed system). However, a similar problem that businesses face is also equally applicable to the scientific computing community. Generally, scientists are experts in their own domain of science and are expected not to be proficient in maintaining IT resources. Also, resource requirements are dynamic in nature and varyies between each experiment or project. Considering these factors, a use case for cloud computing in scientific computing can be made. Indeed scientific computation is steadily migrating into the cloud platform.

Traditional data center based cloud computing had distributed computing embedded in it in terms of aggregating computation powers of multiple devices and creating each virtual device. Recently, this decentralization has been expedited with the use of concepts such as Volunteer cloud, Edge computing, etc. Hence, apart from the typical issues associated with cloud computing like Virtual Machine management, issues arising out of distributed systems also need attention.

In this research work, issues arising out of data placement in such distributed systems and associated reliability factors are studied and some novel solutions to existing problems are proposed. We hope to improve the state of the art in this domain with the studies and proposed solutions described in this work.

List of Figures and Tables

Table 1: List of Figures

Table 2: List of Figures

Table 3: List of Tables

Chapter 1

Introduction

Cloud computing has been a major contributor to the expansion and rapid adoption of e-solutions in a wide domain of business areas, which would have otherwise avoided migrating to Information Technology (IT) based solutions. The illusion of infinite and elastic deployment and retraction of resources on-demand, the convenience to be able to pay only as per use, and encapsulating the underlying complexities in maintaining an IT infrastructure has made it popular even beyond typical business applications. Scientific users of IT services face a similar dilemma as the business users had in the past which is how to gain an advantage that computation has to offer without getting into the nitty-gritty of computation itself. Unsurprisingly, cloud computing has been rising in popularity among the scientific community, to catalyze their research while being blissfully encapsulated from all the maintenance complexities and IT infrastructure planning.

Distributed system in the form of grid computation has been in use in the scientific community for quite sometime. While a cloud computing may necessarily be considered as a distributed system, the actual distribution of physical resources across racks in the data center to the data center itself makes it so. The cloud computing system has added complexity, where the distributed nature of the computing must be transparent to the user, creating an illusion of the users working on a single monolithic system. Further, different efficiency and cost issues creep up for the cloud operator when the distributed system is not being managed efficiently.

Meanwhile, the distributed nature of the cloud has been exacerbated by the emergence of edge computing, fog computing, and mobile cloud computing. These systems have introduced so much heterogeneity in the distributed system, that managing the system has become a genuine concern.

While there are different aspects to infrastructure management in terms of distributed nature of cloud computing, a particularly difficult and important problem is that of data management. Data management, if done properly, can result in the assimilation of the individual powers of individual computing machines and seamlessly make them work together to achieve extremely high-performance computation as required by the scientific community. Traditionally, data transfer has always been the biggest bottleneck in a computer's performance. Hence, a memory hierarchy with advanced caching techniques has been developed to increase data locality in terms of the processor. In a distributed system, data has the potential to be in a different machine connected by a shared and slower network like the internet. Hence, the need for data localization has been exponentially expanded in distributed cloud systems. Indeed, when naive data management techniques are used there is a potential for extremely poor performance and high usage costs.

There has been a lot of work done on data management in distributed systems, especially the cloud. However, there remains much scope for improvement in this domain.

The objectives of this research work are to present different solutions to different problems of data management, specifically data placement in a distributed system such that both reliability and performance of the system may be enhanced. The reliability in a distributed system is handled by two primary methods- erasure coding and replication. While erasure coding incurs less cost in terms of space than the replication technique; its major disadvantage stems from the computational cost incurred to retrieve the lost data. Replication performs two functions- data reliability and data availability. If data is required at multiple locations in a distributed system, it is beneficial to have multiple copies of the data at each site where it is required. In its role in maintaining data reliability, the presence of multiple copies of data ensures that data is available even if one or two copies of the data get corrupted. Unlike, erasure-coding based data reliability, replication involves sufficient storage overhead. However, unlike erasure coding, there is no computational cost in reconstructing the corrupted data. As for data placement, data placement done randomly, without any intelligent data placement, can result in non-localization of data, i.e., data placed in a different site than where it is required. This results in performance degradation due to execution stalls on account of the non-availability of data and waiting for the data to be migrated from a distant site.

It is an interesting study to figure out whether a data placement in a distributed system can be accomplished, where both storage and network costs can be minimized while maintaining high system reliability. This study on optimized data placement is conducted in the context of erasure coding based data reliability management. Generally, the likely use of data in future can be predicted with the use of various machine learning algorithms. Data likely to be required together are placed at a local site to alleviate the data migration problem. It is also an interesting topic to address the cold start problem i.e., how to predict future usage of data when historical information of data usage is not available.

In this research work, we intend to study these issues relating to reliability and data placement in a distributed system. We have proposed novel solutions to some of these problems.

1.1 Distributed Data Management

Distributed data management is concerned with maintaining reliability, consistency, and localization of the data. Consistency is a concern when writing on data is performed. Consistency ensures that the update made to data at one site is reflected at all the sites. Since, we are dealing with read-only data in our problem, consistency is irrelevant to the scope of this work and hence, will not be discussed anymore. In the next two subsections, the other two concerns of data management-localization and reliability are discussed in brief.

1.1.1 Data Localization

It has always been a holy grail of computer architects and system designers to ensure that the execution of a program does not stall due to unavailability of data. One reason for data unavailability is that the data is placed at a distant device from where the execution is being done. In a monolithic system, generally, data is expected to be at the processor's register, which is the closest place to the processor. Unfortunately, due to size limitations, data often has to reside in distant devices such as main memory or secondary memory. In such a case where data is not available close to the processor, data is migrated from the distant devices (main and secondary memory) to cache memory and registers. During this migration, execution cycles of the processor are stalled on account of unavailability of data. To mitigate such a problem, caching protocols have been introduced to pre-fetch the data into the cache before they are required.

In a distributed system, the same problem is present, with an additional issue. While in a monolithic system, the entire storage elements are in the same computational device, but in distributed systems the data may be stored at a storage element on a device connected over a network. This not only adds to the distance of the data from the site of computation where the data is needed but also has to deal with the delay associated with the network- such as congestion. This problem of remote data placement is illustrated in Figure 1.1.

Further, once the data is transferred from a remote device to a local device, the data resides at the lowest level of the device's hierarchy, i.e., the secondary storage device. The local device has to deliver the data to a memory element closer to the processing element, like a cache memory. This is illustrated in Figure 1.2.

Hence data migration over a network in a distributed system adds another layer of inefficiency in terms of execution stalls and performance degradation. Thus, data localization, i.e., data being available at the local node where it is required, becomes a major optimization issue necessary for optimal system performance. It may be noted that data localization in a distributed system concerns the localization of data at the secondary memory of the local device.

1.1.2 Data Reliability Maintenance

Data reliability maintenance is the system's responsibility to recover data from any data corruption event or data being unavailable for any other reason. Primarily, data reliability is maintained in two ways- replication and erasure coding.

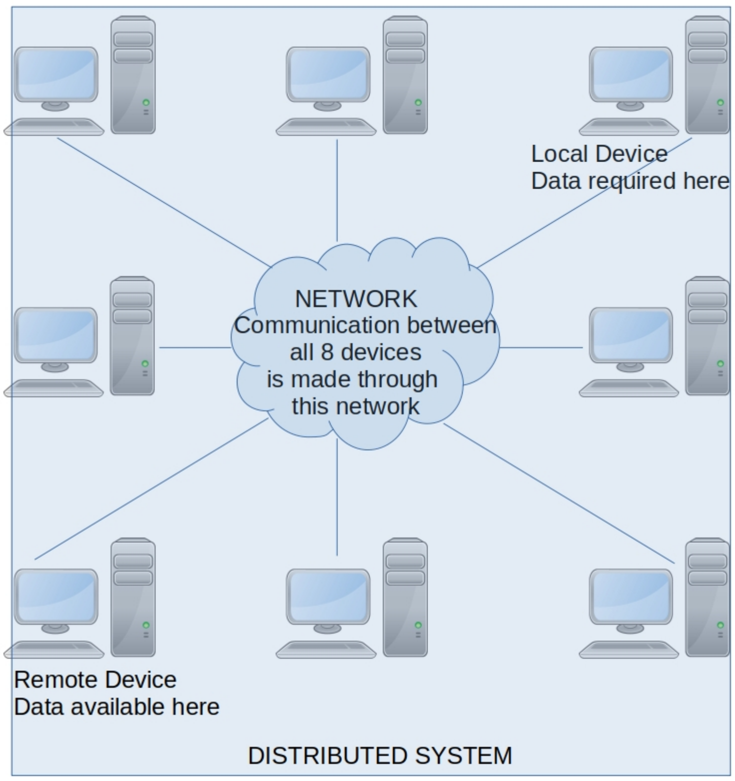


Figure 1.1: An example of a distributed system with remote data placement

Replication

Replication is the system's attempt in maintaining reliability by creating multiple copies of the data unit and placing that data at different sites of the distributed system. For an n factor replication, i.e n replicas are made of data, the system can recover from a loss of $n-1$ loss of data copies.

This is illustrated in Figure 1.3. We can observe that the required data is replicated 3 times and each replica is placed at three different sites (nodes) of the distributed system. If data gets corrupted at one node, the data can be migrated from any one of the other two nodes.

Apart from maintaining data reliability, replication also ensures data localization. When multiple sites require same data, multiple copies of the data are created, one for each node. Thus each node has a copy of the required data locally.

From Figure 1.3, we can observe that three nodes have a local copy of their required data. Considering that no data is corrupted, each

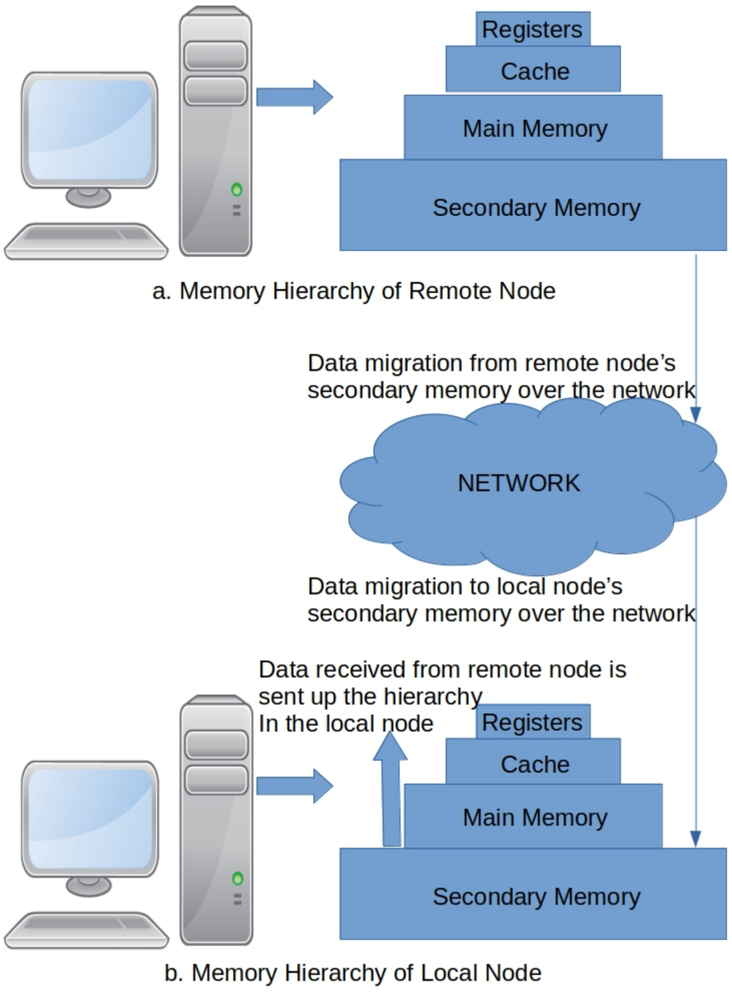

Registers
Cache
Main Memory
Secondary Memory
a. Memory Hierarchy of Remote Node
Data migration from remote node's secondary memory over the network
NETWORK
Data migration to local node's secondary memory over the network
Data received from remote node is sent up the hierarchy In the local node
Registers
Cache
Main Memory
Secondary Memory
b. Memory Hierarchy of Local Node

Figure 1.2: An example of data migration in a distributed system

of the three nodes has a local copy of the data and does not require data migration during run-time.

Erasure Coding

Apart from replication, there is another form of data reliability maintenance-erasure coding. Erasure coding is the technique of creating redundant parity bits along with the actual data. The actual data is striped, or it is divided into equal parts and together with the parity bits, these data are distributed across all the nodes of the network. Erasure coding is not just limited to distributed systems. Systems like Redundant Array of Inexpensive Disks (RAID) in secondary memory management uses the concept of erasure coding to recover from the

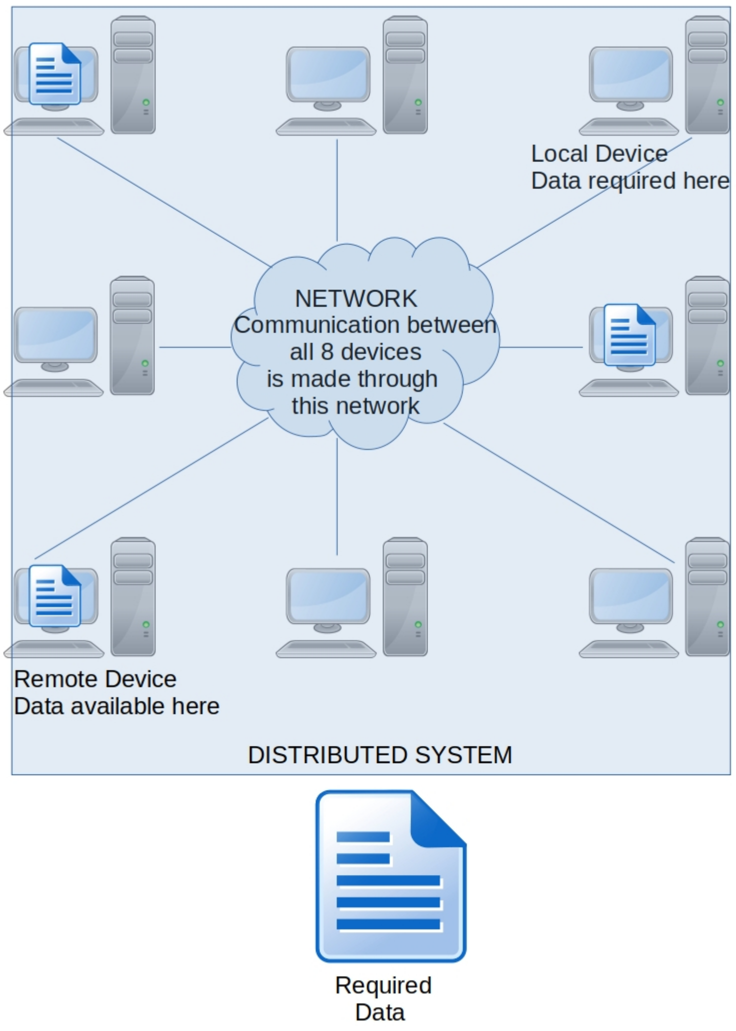


Figure 1.3: An example of data replication in a distributed system

error.

In distributed systems, there are various ways to implement erasure coding. In Figure 1.4, the scheme followed by Hadoop Distributed File System (HDFS) is illustrated.

The difference between replication and erasure coding is illustrated in Figure 1.4. As can be seen in replication that 18 storage units are required for replication, while only 9 storage units are required for erasure coding. This indicates the storage efficiency of erasure coding.

The file is composed of blocks- A1, A2, B1, B2, C1, and C2. A1 and A2 form the A block collection, with a parity block Ap. It may be observed that all three blocks are placed in different storage units in different racks. If the A2 block is corrupted, Ap and A1 can re-

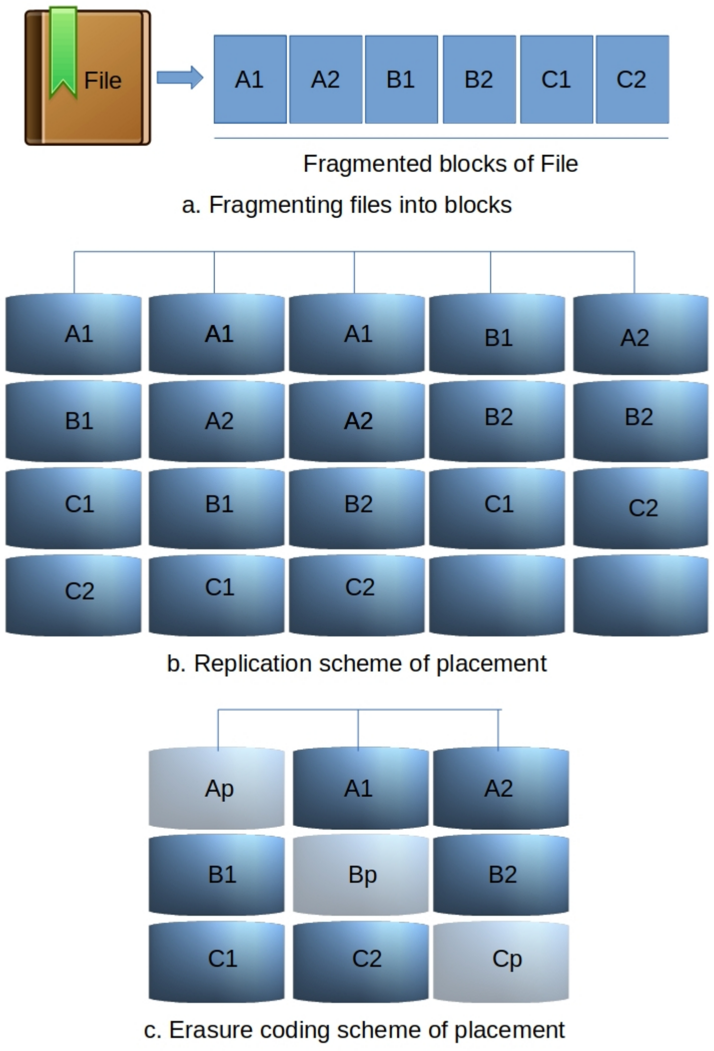

File
A1
A2
B1
B2
C1
C2
Fragmented blocks of File
a. Fragmenting files into blocks
A1
A1
A1
B1
A2
B1
A2
A2
B2
B2
C1
B1
B2
C1
C2
C2
C1
C2
b. Replication scheme of placement
Ap
A1
A2
B1
Bp
B2
C1
C2
Cp
c. Erasure coding scheme of placement

Figure 1.4: Fragmentation of file into blocks in HDFS and placement of blocks in repication and erasure coding scheme

construct block A2. Similarly, any blocks of B and C's collection can be constructed from their respective parity block. Clearly, the system can handle only one block of corruption per A block's collection. Hence, the placement of three blocks (2 data and 1 parity) in different racks and storage element ensures that only one block is lost in the event of a rack or storage failure.

1.1.3 Problems

The problem being attempted to be solved in the works presented in the thesis are the following:

- Given file usage pattern, how do we predict the future requirement of the files at each site of the distributed system? One such future requirement can be accurately predicted, data can be pre-fetched at the appropriate sites where they are likely to be required; thus improving system performance. We are more interested in the question of how to determine which historical usage patterns are useful for prediction and which are not, among the entire data usage pattern.

- When multiple objectives are to be optimized- lower storage cost, lower network usage (localization), and high reliability, a state-space search technique is often the best course of action to be taken. All the possible data placement is considered, with the state space search algorithm trying to find the best placement which optimizes the multiple optimization criteria. The problem is how to implement the proper state-space search algorithm for this purpose.

- When past file usage information is not available, we are faced with a cold start problem. In such a case, content analysis remains the only option. It needs to be investigated whether content analysis of files can be a working substitute for actual file usage patterns for prediction or not.

- To develop a simulation environment, such that the network and storage performance of proposed data placement algorithms on a distributed system can be determined without having to use an actual system.

1.1.4 Our Attempts at Solving the Problems

We discuss some of the techniques that we have leveraged to find solutions to the problems discussed above. In the subsequent chapters, the details on how these techniques are used to solve the problems have been discussed.

Recommendation System

A recommendation system is a type of information filtering system that predicts how a user is likely to prefer a given item. Such insight helps the system to determine whether an item would be useful for a user and if so, such an item is recommended to the user.

Finding an association between items is an important step in recommendation systems. If item A is frequently associated with item B, then if a user N selects item A, likely, he/she will also select item B.

Frequently, such a relationship between items as described above for items A and B can be graphically described in terms of a dependency graph. In a dependency graph, the items are represented as nodes, while an edge exists between the nodes if some association exists between the node pair. The edges are weighted such that amount of weight indicates the amount of association between the items.

The analogy of the recommendation system can be extended to the problem of data placement in distributed systems. Here, the files (data) are the items, while the task using the files corresponds to a user. A dependency graph can be created from the knowledge of the past usage of the file such that the file's historical usage pattern may be determined. The edges with high weight are considered to be the node pairs that have historically been frequently requested together by tasks and hence are likely to be requested together in the future as well. As such files which are used together can be determined and they can be placed together at a site in the distributed system.

The problem is to determine the value over which an edge's weight is to be considered relevant, such that the file pairs are to be considered as having been requested together. This value, known as a threshold, is critical. A lower value would include files that are likely not to be requested together; while a higher value is going to partition the graph so much that relevant file pairs are ignored.

In this dissertation, a social network-based technique has been used to determine a threshold value with considerable accuracy. The solution is aimed at answering the first problem enumerated above; how to determine which historical patterns are useful for prediction.

State Space Search- Evolutionary Algorithms

Recommendation systems are useful when only association relations between files are to be determined to improve data localization. However, when multiple objectives are involved, the recommendation system falls short. Beyond data localization, storage in a distributed system should also be cost-effective while also ensuring reliability. When such multiple optimization objectives are involved, the best course of action is to search through all the possible solutions and find out the solution which optimizes all the criteria.

Searching through all the possible solutions is an intractable problem. Hence, frequently heuristics are used to guide the algorithm toward the proper solution. In cases where heuristics are not possible or do not give good results, metaheuristic algorithms are utilized. Metaheuristic algorithms often inspire solutions from the natural processes' ability to find an optimal solution.

A famous metaheuristic algorithm is a Genetic Algorithm (GA), which inspires optimization process from the natural selection based evolution of species. The idea is that each solution is encoded as chromosomes (containing genes) and each chromosome is compared for its fitness. The chromosomes with the highest fitness are allowed to create offspring and the process continues until a satisfactory result is obtained. The inclusion of mutation operation- random change in the gene values, ensures that the entire search space is explored. The chromosome with the highest fitness value is the solution and the genes in the chromosome are the individual configurations that yield the best solution.

In our problem, the individual placement solutions are encoded as a chromosome and their best placement solution is found through the application of GA. The details will be explored in the subsequent chapter.

Content Analysis: Text Analysis

Often prior information on file usage is not available and this makes predictions impossible. This type of problem is known as the cold start problem. In such cases, the content analysis may come in handy. The premise is that related items are frequently requested together. If someone prefers Rabindra-sangeet and collects an album of it, he/she will likely collect another album on Rabindra-sangeet. Even in Rabindra-sangeet, there are six major genres- Puja (worship), Prem(love), Swadesh (patriotism), Aanushthanik (occasion-specific), Bichitro (miscellaneous), and Nrityonatya (dance dramas and lyrical plays). A person who might be interested in the Prem genre may collect albums related to it; similarly, a person who prefers the Puja genre is likely to collect albums specific to the preferred genre. In this sense, an association is formed between albums in each genre. It is interesting to note that identifying such genres require analyzing the content of the song (listening to the song) and then assigning the specific genre.

In the same way text files can be analyzed using an automated system by making a mathematical representation of text as in vector format in Word2Vec or topic distribution as in Latent Dirichlet Allocation. Like the example above, the text files can be categorized and each user (or task) would be interested in one or few categories of the files. Hence, in a way association relations can be established to some extent. In this work, we attempt to study whether such a content analysis-based method can yield good data placement results.

Also, for any set of files when association information is available, it is generally the norm to place either the entire file on the node or none at all. It is possible to fragment the files into multiple pieces and place them in the node for greater space utilization and efficiency. Note that file association information is available at the whole-file level, but the placement is made at the fragment level.

System Simulation

Distributed systems are essentially a group of computers connected over a network. Thus, the underlying simulation of any distributed system must be that of a network simulator. In the chapter 3, NS3 would be suitably modified to include the additional functionalities of a distributed system. The developed system is envisaged to help researchers acquire necessary performance metrics, without requiring the experiments to be conducted on actual machines.

1.2 Outline of the Thesis

The literature survey is arranged in terms of the concepts necessary to understand the novel works presented in the thesis. Hence, the thesis is composed of two logical parts.

The first part consists of chapter 2, where we present the literature survey. In this chapter, we discuss all the relevant related works associated with the works presented in later chapters, as well as the distributed system as a whole. Two types of papers have been discussed: one type of paper concerns the fundamental aspects of technology for better conceptual understanding; while the second type of paper discusses the state of the art related to each relevant work.

In the second part, consisting of chapters 3-6, the problems and our novel works done to address the problems have been discussed. In chapter 3, we present three works; the first work covers the necessity for handling replication differently than single copy data placement. In the second work of chapter 3, we present the merits and techniques of fragmenting a file in a distributed system, even though association information for the entire file is only known and not that of the fragments. We end chapter 3 by presenting a solution to the problem of developing a simulation system for a distributed system with NS-3 as a base. In chapter 4, we strive to solve the cold start problem, i.e. how to find association information between files from the content of the file itself, when no such information is available. Based on this calculated association data, a data placement algorithm has been proposed for distributed systems. In chapter 5, we consider the problem of prefetching data in the nodes of the distributed system. Here, file usage and ssociation data are available. However; the trick is to determine which of these file associations are incidental and which are relevant enough to be used for predicting future file usage. Generally, a threshold based approach is proposed in the literature. Association values below the threshold are considered incidental and discarded. In this chapter, we discuss an algorithmic approach to finding such a threshold. Finally, we present our last work in chapter 6, where we have develop a placement method that not just tries to localize data and hence improving network performance and reducing network related costs, but also tries to minimize storage cost as well as maintain high reliability of the system. A GA based approach has been discussed in this context.

We provide our concluding remarks in chapter 7 of this thesis with an outline of the future works that can be performed from the works presented in this thesis.

Chapter 2

Literature Survey

2.1 Distributed storage system

For any kind of computation, the storage system plays a pivotal role. Such systems store the data which is to be required by the tasks to be executed in the system. In the case of distributed systems, complexity of storage management tends to rise with the complexity of the system. In this section, the Distributed Storage System (DSS) would be discussed together with some of the notable and relevant works in literature.

2.1.1 Taxonomical classes

Taxonomically, a DSS could be classified into 8 classes as follows:

- System function
- Storage Architecture
- Operating Environmnet
- Usage Patterns
- Consistency
- Security

The most relevant of these classes and their brief description with examples in literature is presented here:

System Function

In this class, the file systems are classified in terms of the functional requirement of the file being used. Each of the classes is described below:

- Archival: The storage system is used for persistent storage of data. Since, such persistent storage is required by users for backing up the data, fail safe reliability is of utmost concern in this case. Generally, these files do not require frequent modifications, hence the need for consistency management is eliminated. Some examples include PAST [1] and CFS[2].

- General Purpose: These systems allow a transparent extension of the user's local file system to that of a remote file system. Hence, compliance with the POSIX API standard is an absolute requirement. Some examples of such systems are NFS [3, 4] and Ivy [5].

- Publish/Share: Unlike the first two storage systems described, this system is aimed at file publishing or sharing mostly anonymously or to avoid censorship. Hence, reliability is sacrificed for anonymity. As such, a strict peer to peer approach is followed. Some examples in class include Mojo Nation [6], BitTorrent [7], etc.

- Performance: This class of DSS is tuned to perform I/O operations in a high performance computing environment. The nodes in the distributed system are connected through a high bandwidth network. Hence, the file system is tuned to adapt to the particular workload being deployed. Also, like the general purpose file system described above, transparent file system access is provided. Some examples include Lustre [8, 9] and GPFS [10].

- Federation and Middleware: Due to the possible heterogeneity in the distributed systems, interoperation between them is made possible with the use of federation middleware. Such federation middleware is concerned with administrative functions such as cross domain security, homogeneous interface in a heterogeneous environment, etc. Examples in this class include Freeloader [11] and Oceanstore [12].

- Custom: As the name suggests, these storage systems implement certain unique characteristics which are beyond the functions implemented by the above systems or have a combination of functions of the above systems. Freeloader and OceanStore allow the user to implement customized functionalities, in addition to federation functionalities.

To meet the demands of the objectives outlined in this thesis, functionally a file system would have to be tuned towards serving high performance storage needs.

Architecture

In terms of architecture, there are two kinds of storage systems; client server and peer to peer. Client server systems are the oldest systems in existence. In such systems, one of the nodes, designated as the server, has special capabilities more than the other nodes, called the clients. The server serves the data requirement of other nodes called clients. The role of nodes as either client or server is well defined and exclusive. Some examples of such architecture have been discussed in [13, 11]. The server nodes also serve administrative functions, such as authentication, replication, consistency management, etc.

The client server architecture presents a centralized approach to a DSS. In the case of a globally centralized architecture, one central server serves all the clients in the system. This creates a single point of failure in the system and has severe scalability limitations. To alleviate the problems associated with globally centralized architecture, a locally centralized architecture is proposed. In a locally centralized architecture, multiple servers are responsible for a subset of client nodes. Hence, such systems are more resilient to server failures and can scale better.

Client server architecture is suited for use in a trusted or partially trusted operating environment; or in other words, this architecture is suitable for operations in a controlled environment. Such an environment allows the system to focus on achieving good performance, maintaining a strong consistency, and providing a POSIX compliant I/O interface.

A peer to peer architecture [14, 15] in its strictest sense (pure peer to peer) is completely symmetrical, with all nodes having the same capabilities. Hence, such architecture makes the system very scalable, and such systems are capable of adapting to a dynamic environment. While the problems due to centralization has been overcome with a pure peer to peer architecture, achieving and implementing such an architecture is difficult. The nodes are likely connected through an internet which is a highly asymmetrical connection medium; thereby introducing asymmetry in the system, resulting in poor QoS. Further, administrative necessities like establishing trust between nodes and accountability require the presence of some centralized entity that is trusted by all parties, which introduces centralization.

Due to the difficulty in implementing a pure peer to peer network, two forms of centralized peer to peer architectures are proposed globally centralized and locally centralized. Napster [16] is an example of a globally centralized peer to peer architecture. A central server records the details of other peers and the files contained in them. A node trying to access a file from another node in the system would first connect with the central server to obtain the necessary details, which is then used to connect to the relevant node(s). Similar to the globally centralized client server architecture, this globally centralized architecture suffers the same problems and unsurprisingly the solution comes in the form of locally centralized peer to peer architecture. Gnutella [17] is an example of such a locally centralized peer to peer architecture. In Gnutella, certain nodes assume the server like responsibility, known as supernodes. In sufficient numbers, the use of supernodes can relieve scalability and reliability issues, while also decentralizing the system.

Operating Environment

A DSS may be in three operating environments trusted, partially trusted, and untrusted.

The trusted environment is a system under a common administrative domain and is quarantined off from the outside. This characteristic ensures a high level of QoS, lesser security issues, and freedom from unpredictable behavior from outside entities. Hence, workload analysis can be carried out and performance tuned to optimum. However, the only tradeoff is its limited scalability.

A partially trusted operating environment consists of a system with a combination of trusted and untrustworthy nodes. The common organizational bound is still maintained in terms of administrative control. Some degree of trust may be assumed but security must be on the lookout for possible rogue nodes, which may not adhere to terms of service. Further, the predictable behavior may not be guaranteed, since sharing of resources, especially networks, is assumed. In such an environment, DSS is designed for maximum compatibility.

An untrusted environment is an open and public system where no trust between nodes may be assumed. Accountability is difficult if not impossible and the system is open to attack [15]. Some of the key characteristics include a transient user base [6], the slashdot effect [18], etc., to name a few.

Usage Patterns

For the three operating environments discussed above, three types of usage patterns concern are to be considered.

In an untrusted system, the main issue is that some form of centralization leads to a potential bottleneck that needs to be relieved. The problem further aggravates due to the flash crowd effect, where there is a sudden surge in demand from the server, which cripples the server [6]. In such a case a cloud system could be used to cater to the momentary increase in demand. A secondary but important problem is that of peers only participating as a consumer of services and not as servers, thereby violating the core principle of cooperation that keeps the peer to peer system functioning [19]. An incentive based mechanism could be used to prevent this behavior [20]. Another problem is that of the asymmetrical nature of the user's internet connection, being biased towards downloads [15].

In a partially trusted environment, the major issue is the underutilization of resources [21] and utilization of optimal underutilized storage resources [22].

The most interesting aspect is seen in the case of a trusted environment. Due to its predictability and controlled nature, the main source of determining usage patterns are the applications running on them. Each application has its unique access pattern of data [23]; it is necessary to study the application's access pattern and accordingly tune the storage system, such that good performance may be obtained from the DSS.

Federation

With the integration of resources spanning beyond institutional and geographical boundaries, a middleware becomes necessary to federate resources across the sharing institutions. This forms the basis of Grid computing [24].

Concerning DSS, federation involves understanding the semantics of the data being shared and its associated metadata. Federation produces a homogeneous interface by abstracting the heterogeneity of the individual storage systems. Hence, users need not concentrate on different management of data and can instead concentrate on how to use the data. Some examples in the federation of storage services can be found in [25, 26].

2.1.2 Contextualizing Distributed Storage System in Present Work

The taxonomic classification of DSS discussed in Section 2.1.1 needs to be elaborated in the context of the work presented in this thesis. First, we consider system function; considering the objective of the work, the DSS function should be aimed at performance and as a federation middleware. Considering that the system is being developed solely for use in scientific workflow, the characteristics of high performance function are well suited. While homogeneity in network and node behavior are considered for simplicity in this work; however, the system would likely be introduced to heterogeneity as well. Since multi institutional collaboration is envisaged, heterogeneity in data storage may be assumed. Hence, federation functionality is to be incorporated in the DSS.

In regards to the operating environment, a trusted operating environment can be assumed, since the system being developed is for use by accredited and affiliated researchers. The same applies to usage patterns as well.

In terms of architecture, a peer to peer system with a locally centralized super nodes concept would be suitable. The super nodes would be responsible for maintaining administrative control over the local cluster and would have communication with other super nodes for synchronized operation.

2.2 Scientific Workflows

Workflow systems owe their origin from the business community, where a need for a group of services aimed towards a common objective is felt for precise coordination. To simplify this need for complex coordination of services, workflow technologies are introduced. Since then scientific workflows [27] have been introduced to automate large scale scientific analysis using computers.

One of the alternate automation techniques used by scientists is to use different general purpose scripting languages to automate by the integration of various analytic components. Certain special purpose languages such as R, SAS, MATLAB are now integrable with scripting languages like Perl, shell scripts, etc. for automated use during analysis.

Scientific workflows differentiate from such tool integration processes by working on the principle of dataflow language. Scientific workflows are depicted as a directed graph, where nodes represent a computation state (also called an actor), while the edges represent the data flow between each computation step. Hence, scientific workflows provide a high level tool to analyze and prove a scientific hypothesis.

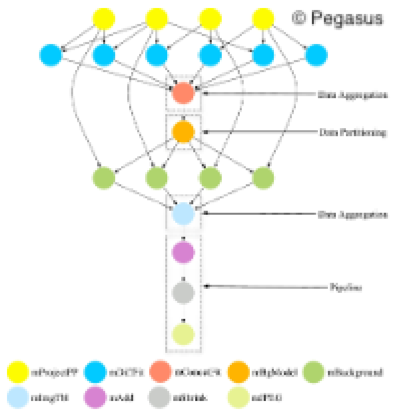


Figure 2.1: MONTAGE workflow

In our work, two workflows have been shown for MONTAGE and CYBERSHAKE experiments in Figures 2.1 and 2.2 respectively [1]

[1]Image credit: https://pegasus.isi.edu

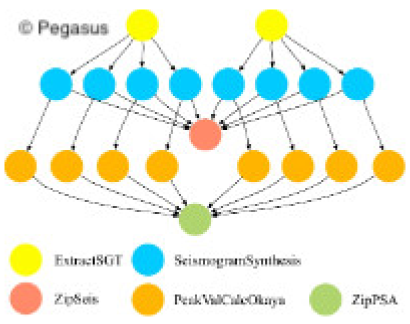


Figure 2.2: CYBERSHAKE workflow

from the Pegasus workflow gallery. The colored circles in the figures denote the different tasks which perform computation on the data. The lines connecting the task from the upper layer to the lower layer denote the dependence of data by the task at the lower layer on the upper layer. Hence, the lines also denote the flow of data from one task to another. Each task, even in the topmost layer requires data on which the task computes upon. In the lower layers, the data may also be the result of the computation of one or more upper layers.

To create a successful scientific workflow, certain desired attributes must be satisfied:

- Scientific workflows should mirror the scientist's conventional work by making use of their methods over distributed resources.
- Depending on the observer/user, the scientific workflow must be able to abstract an unique view suitable for the user for the same information.
- Scientific workflow should be able to robustly and dependably support transportation and analysis of large quantities of data distributed across various repositories.
- Scientific workflow should be able to be reused, refined over time, shared with other scientists in the field and the results of one scientific workflow could be used as an input to another scientific workflow.

Some of the examples of popular scientific workflows are Taverna [28], Kepler [29], Triana [30] and Pegasus [31].

2.3 Cloud Computing

The present work is concerned with data management of distributed scientific workflows executed in a cloud environment. Having covered distributed storage systems and scientific workflows in earlier sections, it is time to review the relevant works in literature in connection with cloud computing.

Cloud computing has been formally defined by NIST in [32] as follows:

Definition 1. *"Cloud computing is a model for enabling ubiquitous, convenient, on demand network access to a shared pool of configurable computing resources (e.g., networks, servers, storage, applications, and services) that can be rapidly provisioned and released with minimal management effort or service provider interaction."*

According to the NIST document, a cloud service consists of five essential characteristics, three service models, and four deployment models. Cloud computing is related to another distributed computing model called Grid computing. A comparative study between cloud and grid computing is available in [33].

Architecture

The architecture of a cloud computing environment can be divided into four layers:

- Hardware: Physical resources of the cloud like computing, storage, and networking elements are managed at this layer.
- Infrastructure: This layer is also known as the virtualization layer. This layer pools physical compute and storage resources and partitions them into virtual resources (Virtual Machine or VM) using virtualization tools such as Xen [34], KVM [35] and VMware [36].
- Platform: The operating system, related system applications over the VM created in the earlier infrastructure layer is created in this layer. Examples include Google App engine [37]
- Application: The topmost layer consists of the actual application which is of interest to the user.

The layers have loose coupling with other layers to ease separate evolution of each layer.

As part of the business model, different services are provided to clients by cloud providers. The following traditionally important services are as follows:

Infrastructure-as-a-Service (IaaS): On demand provisioning of computing and/or storage resources in terms of a VM. Examples include AmazonEC2 [38], GoGrid [39] and Flexiscale [40].

Platform-as-a-Service (PaaS): Caters to the need at the platform layer resources such as Operating system support. Examples include Google App Engine [37], Microsoft Windows Azure [41].

Software-as-a-Service (SaaS): This business model caters to the provisioning of on demand applications over the internet Examples include Salesforce [42], Rackspace, SAP Business ByDesign [43], etc.

Some Research Areas and Challenges

The research areas concerning cloud computing span across varied domains including architectural design of data centers, VM migration, resilience and fault tolerance, scalability, and backward compatibility.

The research area of concern in the present work is that of implementing distributed file systems over clouds. Google File System (GFS) [13] is specially designed and optimized to run data centers that are tuned to provide high data throughput, low latency, and robustness to component failures. Another related file system of concern is Hadoop Distributed File System (HDFS) [44]. HDFS is described in detail in a later section of this chapter.

The research challenges include areas like automated service provisioning, virtual machine migration, energy management, traffic management and analysis, data security, and software frameworks. Since these issues are beyond the scope of this work, a detailed discussion on these is not included in this literature survey to maintain brevity.

There are two research challenges that have relevance to the present work. The first research challenge is in regards to storage technology and the other is data management. This will be discussed in the next two sections of this chapter. The other topic concerns the evolving novel cloud architectures. These architectures will be discussed in the next two subsections

2.3.1 Decentralized Cloud Computing

Traditionally, cloud computing is treated as a centralized computing paradigm. However, with the rise in IoT and 5G technologies, distributed cloud computing systems are on the rise. Distributed cloud computing consists of frequently overlapping terms like fog computing and edge computing in addition to terms such as mobile cloud computing, ad-hoc cloud, etc. This type of cloud computing compliments the traditional centralized data-center based cloud by bringing computation closer to the client (or sometimes on the client device itself). There may be several reasons to adopt this decentralized approach. For example the latency of data transfer to and from data centers may be unacceptable, the data may be partly sensitive and that computation would be preferably computed at the client's machine, etc are some of the reasons. In literature, the following surveys discuss these decentralized cloud computing the state of the art, research challenges, etc:

- [45, 46, 47]: Various aspects of decentralized cloud computing are comprehensively covered in these surveys. They try to distinguish between ambiguous and overlapping paradigms like fog computing and edge computing, mobile cloud computing, and mobile ad hoc cloud computing, etc. apart from other concepts related to decentralized cloud computing. The survey looks into the need for the evolution of such decentralized cloud platforms, application use cases, the enabling hardware and software technologies, and future research to solve the existing issues.
- [48]: This work explains the interplay between the underlying concepts like MANET, mobile computing, etc. to implement a cloud computing system.

2.3.2 Volunteer Cloud Computing

Volunteer Cloud Computing (VCC) is a subset of the broader computing paradigm called Volunteer Computing (VC). VC includes VCC, Volunteer Grid Computing (VGC), and Mobile Volunteer Computing (MVC). Volunteer computing was created out of a need for ever increasing need for computational resources. Traditional Cloud Computing requires high data center set up and maintenance costs, with a significant portion of these costs being spent on non computational expenses like cooling systems, electricity [49], etc. On the other hand, more than 2 billion desktops remain under utilized; while [50] reports that PCs in institutions remain idle 97% of the time.

VC allows the utilization of these wasted computational resources by way of voluntary donation to run compute intensive tasks, such as that required for scientific analysis. Unlike traditional data center based cloud systems, VC does not require specialized infrastructure or requires cooling expenses, thereby making cloud computing more affordable.

Some examples in VGC include Entropia [51] a system that aggregates the computing resources of desktops within an organization into a virtual grid; Condor [52] a system which utilizes idle workstations for long running jobs; BOINC [53] the most influential volunteer computing resource which is used by CERN (LHC@home [54]) for particle physics research.

In terms of VCC examples, several sub classifications are possible such as Volunteer Desktop Clouds [55], P2P Volunteer Clouds [56], Social volunteer Clouds [57] and Volunteer Storage Clouds [58].

2.3.3 Scientific Workflows and Clouds

Cloud was built and traditionally utilized for computing needs related to business. However, due to similar requirements of scientific computation, cloud computing is being adapted for scientific computation as well. As such scientific workflow could be deployed in a scientific cloud computing environment as well.

Some of the important examples are Cumulus [59], Eucalyptus [60], OpenNebula [61]. Some of the unique requirements of scientific cloud computing encompass the need for massive parallelization in code while also catering to the inter-dependency between the tasks. Further, the data generated and used is huge often in the range of Terabytes.

Scientific cloud computing often has added advantages over grid computing like dynamic resource allocation and virtually infinite resources as per requirement, ability to deploy the legacy application through virtualization support, etc. Another popular science computing platform is High Performance Computing (HPC). Clouds suffer some disadvantages when compared to HPC. Such disadvantages are relatively slow network, virtualization overhead, etc.

2.4 Data management in Distributed Systems, Cloud and Scientific Workflows

In this section works concerning storage elements, their failure and how such failures are handled are discussed along with discussion in relevant works in literature.

2.4.1 Storage Failures

Distributed storage systems are made up of individual storage elements, each of which is prone to failures. Storage system failures are one of the leading causes of node unavailability (in terms of data) and loss of data. Predicting the impending failure of a storage element can be used for early transer data between storage elements and prevent or mitigate system downtime.

One of the earliest influential works in predicting disk drive failures in a large collection of disk drives has been the work in [62] and for distributed storage system in [63]. Based on these studies, smart disk failure prediction methods have been proposed in [64]. Similarly, work in improvement in storage efficiency by building upon the disk failures studies have been proposed in works such as [65]. Applying the disk failure prediction and studies on disk failures, efforts have been made to improve the efficiency of the data center as a whole. [66] has studied the importance of disk failure prediction in datacenter design. Similarly, [67] has proposed a reliability model for disk arrays. [68] studied hard disk failures in the context of overall data center failure and provides a more holistic picture. An example of node failure in use in making cloud computing efficient is in the work by Lin et al. [69]

2.4.2 Reliability

Data reliability is the study of increasing the durability and availability of data. There are two types of failures permanent and transient failures. Durability deals with the protection of data from permanent failures. This includes measures that prevent complete loss of data in case of a catastrophic failure of components. Transient failures are cases, where a data source may go offline for sometime and making data from that source becomes unavailable. It is to be ensured that the temporary or permanent unavailability of data from one source does not make the data completely inaccessible. Availability ensures efficient data retrieval in such transient failures. There are mainly two ways to ensure reliability in data storage; erasure coding and replication.

Erasure Coding

Erasure coding is the data reliability procedure in which redundant data is stored along with the actual data. If a file is of n bytes, we store k bytes of data, such that $k > n$, and redundancy stored is $(n - k)$ bytes. This redundant data $(n - k)$ is used to recover the n bytes of data in case of corruption. Erasure codes are natural to distributed systems. The k bytes of data (original and redundant) is divided into an equal number of 'x' stripes. These stripes are placed at different nodes of the distributed system. If one or more nodes stop data retrieval services (permanently or transiently), the remaining data in other nodes can recover the lost data and normal services may resume. If a large amount of data is lost, however, the data corruption becomes permanent. Thus, more the redundant data stored in the system (parity data), the greater the system failure that the system can recover from. However, storing more redundant data adds to the storage cost of the system; thereby creating a trade off between reliability and storage cost.

There are two types of erasure coding that may be implemented:

- MDS codes: Maximum Distance Separable codes are those which can make a system recover from any combination of m node failures (considering the entire storage of a node as one unit) for m parity data saved on them. An example of this type of code is Reed Solomon code, which is used in commercial services like Microsoft Azure [70] and Facebook [71].

- non MDS codes: non MDS codes are a type of erasure code that can recover from less than m combination of node failures while storing parity data in m nodes. These are also called locally repairable codes since the redundant data is stored in each node and the recovery involves data from the local nodes storage rather than parity blocks from other nodes. This code is suitable for recovering from an error within the node at the cost of storing extra data at each node. An example of this type of code is [72]

Regeneration codes are those for which data can be easily recovered with the minimal data download. These are traditionally divided into two classes:

- Minimum Storage Regeneration (MSR): As the name implies, these erasure coding schemes optimize the storage cost when having to store the redundant data. Some examples in literature include [73, 74].
- Minimum Bandwidth Regeneration (MBR): In this case minimizing network bandwidth usage or simply network usage is the optimization criteria of the system. Some examples in literature include [74, 75].

From the above discussions, it can be summarized that erasure codes can be applied such that either bandwidth or storage could be optimized while maintaining a high probability of recovery from a data loss. There are works in literature [76, 77, 78], which looks into the question of minimizing data storage in a high error recovery regime. The interesting question is whether, in the same high error recovery regime, both storage and bandwidth can be jointly minimized. With erasure codes, alone, may be not. However, in a distributed system, there is an option of intelligently placing data in the system. Whether an intelligent data placement can jointly optimize both storage and bandwidth in a high error recovery regime is something that will be studied in a chapter 6.

Replication

Replication is the second process of maintaining data reliability. Unlike, erasure coding, copies of the entire data (file) are made. These copies, called replicas are distributed and stored in different nodes of the distributed system based on the replica placement scheme in place. While replication takes more storage space than erasure coding, it has multiple uses. Firstly, no expensive computation is required to generate the lost data. Secondly, replication may also be preferred for other purposes like load balancing, energy conservation, minimizing data transfer time, etc. In erasure coding, all the nodes containing the stripes of data are required for recovery. Thus, the second advantage cannot be exploited. In contrast, in replication, only one among the multiple copies of the data is required.

Replication may be of two types:

- Static replication: A static replication is performed with the assumption that the system's behavior will remain constant during its lifetime. Replication decisions are taken at the system start time and such decisions are not changed (or changed manually) during the system runtime. Some examples are given in [79, 80]
- Dynamic replication: In dynamic replication, the replication system is triggered from time to time. It monitors for any behavior change and accordingly computes a new replication strategy. Some examples are given in [81, 82]

2.4.3 Data placement and Prefetching

In distributed systems, data placement deals with the problem of intelligently placing data on nodes of the distributed system to optimize one or more desired objectives. Prefetching is a similar concept to data placement. In prefetching, the likely data usage is computed by analyzing historical usage patterns and fetching data before it is used.

The desired objective in mind is to ensure that all data required by a task running at a node in the distributed system is available locally and no or minimum data needs to be migrated from a remote node of the system. A good data placement ensures such a local availability of data which improves task performance by mitigating execution stalls by reducing remote migration of data. In the case of prefetching, since the data is already present in the concerned node, not only is the execution stall time reduced, even the task does not have to wait for starting the execution due to unavailability of data.

In this work, data placement is discussed in the context of scientific workflows. The workflows consist of tasks, which may or may not be dependent on the output data of another task. For tasks that have no dependency on another task (or the output data is available), they can be run in parallel. As discussed in section 2.2, tasks require data to execute, and in the context of scientific workflows, the data consists of a list of files. When viewed in inversion, each file has a list of tasks that require the concerned file for its execution.

Interdependency between a pair of files has been defined in [83] as the number of common tasks that require both the files during its execution. Mathematically it is defined as follows: $Task\_list(fn)$ is the list of tasks which requires file fn during its execution.

Interdependency between files computes how likely both the files will be required during the execution of tasks. A high interdependency value indicates that the file pairs are likely to be used for many tasks; hence, they should be placed together.

Interdependency based data placement has been adopted in various works as presented in [84, 85, 86], etc. In various works described in the subsequent chapters this concept of interdependency have been described.

A related concept is that of interdependency based prefetching. However, since a prediction of likely future data usage is required to be made, some kind of predictive data analysis needs to be performed. In these cases, recommendation system based predictions come in handy.

There are primarily two ways in which the prefetching is performed in scientific workflows:

- Dependency graph: In these methods, the interdependency between the file pairs are collected and represented in the form of a nXn matrix, called dependency matrix where n is the number of files. The dependency matrix is used as an adjacency matrix for a graph. The resulting graph obtained is called the dependency graph where the nodes are the files, edges indicate interdependency between the files and the weight of edges represents the amount of interdependency between the file pairs. The graph represents the past file usage pattern and it is assumed that a similar pattern would be repeated in the future. The core principle is to partitioning the graph, such that each disconnected components represent a cluster of highly dependent files. Subsequently, one of these file clusters is chosen and the files in that cluster are prefetched. Examples are given in [87], [88], etc.

- Data mining: Data mining is a collection of concepts in data science that extracts useful knowledge from a huge collection of data. Frequently used in recommendation systems, it is used to find a pattern in a collection of data. This pattern is used in the recommendation system to optimize different objectives. Association rule mining is an important approach to data mining, which uses algorithms such as Apriori algorithm, to find association patterns between the items. Such an association pattern provides useful knowledge on how a pair or group of items are associated with each other. In the present context, the items are analogous to files, and associations between files are determined through data mining. Examples of such data mining techniques used in data management has been presented in [89], [90], etc.

In chapter 4, the dependency graph based approach will be explored further.

2.5 Text Mining and Document Similarity

2.5.1 Text Mining

Text mining is the process of knowledge extraction from textual data by transforming unstructured data into structured data, which can be utilized for analysis. In this section, a subarea of text mining is discussed, which is relevant to data management in distributed systems.

2.5.2 Textual Content Analysis

Content analysis techniques can be categorized into either classification (supervised) or clustering (unsupervised) techniques.

In classification, the objective is to segregate and group text data units (files) into different classes or categories. When only two classes are used for classification, it is called a binary classifier [91]. When classification into multiple classes is required, multiple binary classifiers are used [92]. In supervised technique, some amount of human categorized data is required from which the algorithm can learn about the basis of classification. When such data (called training dataset) is not available, a supervised technique cannot be used and an unsupervised technique becomes useful.

The clustering techniques utilize mathematical similarity (like Cosine distance, Manhattan distance, etc.) between files to classify them into different clusters [93].

Some example of application of content analysis of textual document in literature includes mapping published research [94, 95], patent summarization [96] and novel topics in research articles [97].

2.5.3 Similarity and Recommendation Systems

In chapter 4, content analysisbased text similarity measurement will be used to find interdependency. However, the basis of this method is in the relation between data interdependency calculation and recommendation systems. A recommendation system finds an association between a pair of items based on the number of common cooccurrences. A common example is that of market basket based item association [98]. In data management, data interdependency between a pair of files is nothing but an association between the file pairs. The analogy to an item is the file, while the analogy of the market basket is the list of tasks that require the data.

There are plenty of works in the literature on the use of text analysis for building a recommendation system. Generally, such a content based recommendation system is used when no or minimal information is available; hence, other means of recommendation are not possible. Sometimes content based analysis is used to augment recommendation systems using traditional means as well. Some examples are discussed below:

- In Almalis et al.[99], text analysis has been used to analyze CV of job seekers and propose a suitable job based on the analysis. The analysis results in clustering of similar CVs and matching each cluster with one or more jobs.
- In Amami et al.[100] a recommendation system for scientific papers has been proposed. This is the example of enhancing traditional recommendation systems like collaborative filtering with probabilistic topic modeling.
- In Wang and Blei [101], both collaborative filtering and LDA topic modeling have been combined for recommending a new scientific article whose usage information is not available, but such information on a topically similar article is available.

2.5.4 Beyond Text

Text analysis is a potent tool in recommendation systems that can also be repurposed for data management based on data interdependency in distributed systems. Text and bioinformatics data share many common structures which allow cross application of an analytical tool from one domain into the other. A common example is that word similarity measurement using edit distances. In both cases, the fundamental principle of letter sequence alignment is used. In the case of text, the letters are straightforward. While in the case of bioinformatics (say, gene bank data), the individual letter of gene sequence is considered a letter. More examples of application of Natural Language Processing (NLP) tools in bioinformatics (protein data) can be found in [102]. Similarly, tools like LDA can be used as well as and is evident from [103].

The above discussion points to the fact that beyond text data, similar algorithms can be re purposed for managing bioinformatics data as well. The consequence is that the data placement tool proposed in chapter 4 has potential for application beyond text datasets.

2.6 Tools and Methods used

Beyond the concepts discussed in the previous sections, some other miscellaneous concepts, relevant to our work, are being discussed in this section.

2.6.1 State space search

Data placement involves the grouping together of files in such a way that highly interdependent files are grouped while non interdependent files are placed in other groups. One way to look at this is that there are finite, albeit numerous combinations of file groupings that we can make; out of which, one grouping achieves the above objective. Looking in this way, data dependency based placement can be remodeled as a state space search problem. Each combination represents a state and the combination of files that achieves the stated objective represents the goal state. Starting from a random initial state, the problem is to devise an efficient state space search algorithm to find the goal state in the state space.

To aid this search effort, often assistance from estimate based guidance, known as a heuristic, is sought. Heuristic gives good results when good estimates are available which can guide the search algorithm towards the goal state. In heuristic based searches, a goal state is defined by the points of minima (or maxima), out of which only one minima is the least, called global minima. In the problem concerning our work, maxima (negative minima) is the combination of files such that each group has the highest inter dependency value. A heuristic based algorithm may end in one of the other maximas (local maximas) and declare that the goal state has been reached. In the context of the present work, such a maxima would be a grouping that has a high interdependence value, but another grouping may exist which has an even higher interdependency value, which is not explored. To prevent this undesirable result, due to unavailability of reliable heuristics, metaheuristic algorithms are used in this work.

Metaheuristics

The idea of metaheuristic is to use general problem independent strategies to solve specific hard problems [104]. Metaheuristics themselves do not solve any particular problem but it can be applied to solve a wide variety of problems.

Heuristics as has been described earlier, is a guide for the algorithm its towards goal state. The meta component is the higher level strategy that controls the underlying heuristics to overcome the limitations of pure heuristics based searching due to unavailability of reliable heuristics .

Metaheuristics based searching achieves the following goals:

- Efficiency: Find a solution to an NP hard problem in a reasonable time
- Accuracy: Find an optimal or near optimal solution to the problem. Since it is a soft computing technique, absolute accuracy is not necessary. Some level of inaccuracy as determined by the user and the resulting non optimality is acceptable.
- Escaping the local minima (or maxima): Cover the entire solution space while searching, such that the global maxima or minima is reached.

Broadly, a meta heuristic algorithm may be classified into trajectory based or population based algorithms. Due to the increased diversification achieved in population based methods, this class will be discussed in more detail. Population based algorithms are primarily of the following types:

- Evolutionary computing: These are iterative search techniques, which stochastically manipulate the present solution to produce a solution that is closer to the fitness value (mathematical expression of the objective). Genetic Algorithm (GA) [105] is the most famous example in this category.
- Swarm intelligence: This category of search algorithm has searching agents distributed throughout the state space. The agents evaluate the search space at the local level in which it is deployed. Communication between the agents with their neighbors gives a

global picture of the search space and as such the optimal point is discovered. Some popular examples include Ant Colony Optimization (ACO) [106] and Particle Swarm Optimization (PSO) [107].

Some examples that demonstrate solution of real world problem solving using metaheuristics are as follows:

- [108]: This work utilizes two variants of ACO for cryptanalysis purposes. Performance on six encryption processes is evaluated with good results, Simple substitution, affine, Feistel, carre de Polybe, Vigenere, and transposition algorithm.
- [109]: In this work, GA has been applied in the domain of bioinformatics.
- [110]: Applied GA in combination with three more metaheuristic algorithms to solve crop rotation problems in the field of agriculture.
- [111]: In Wireless Sensor Network, maximum sensor coverage with minimum device deployment and minimum energy expenditure are the objectives. In this work, ACO has been used for searching the optimal solution with parameter analysis performed with MIN MAX variant of ACO, giving excellent results.

2.6.2 Recommendation System

Another way to look into the stated problem is as a recommendation system problem. We look back into recommendation systems once more as a background to the concepts of Social Network Analysis (SNA), that we will discuss next. According to Bobadilla et al [112], recommendation systems are programs that predict the most appropriate products or services for a user. Such predictions are made based on the user's interest in the item as seen from history and also the information on the item itself; further taking into account the user's interaction with other users. In terms of the data placement problem, the user is a file and the item is also a file. The prediction has to be made on whether one file is to be grouped with the file in question. This decision is based on the file's interest in the other file (based on how many tasks they are required in common; called interdependency) and also whether this interest is more than the interest with other files.

Broadly there are two types of recommendation systems; collaborative recommendation and content based recommendation. A third hybrid recommendation type may be proposed which combines the two basic types.

A collaborative recommendation is based on the concept of a similarity matrix. The core principle is that if there are five items and two users have indicated a similar preference to the four items, it is likely that the fifth item's similarity would be similar as well.

Content based recommendation technique is the recommendation process where an item is recommended to the user if its content is similar to an item in which the user has already shown an interest. For example, if a user is interested in a book, then he/she can be recommended other books which have similarities in terms of contents of the book. Such similarity may be analyzed through a plethora of Natural Language Processing (NLP). One of the chapters in this work is based on this concept of content based recommendation.

Examples of the two dominant types of recommendation system are discussed in [113, 114, 115, 116, 117].

The third type of recommendation system utilizes graph analysis. This will be discussed in the next subsection on Social Network Analytics (SNA).

2.6.3 Social Network Analysis

Certain graphs have special structures which allow them to be analyzed by SNA. SNA studies two sets of properties:

- Relational properties: This study concerns the content and relationship between the network actors [118]. Contents are the information or material shared by actors within a network and relationship refers to the social tie between the actors of the network [119]. Generally, the network is represented using graphs with the social actors represented as a node and the edges represent the social ties between the nodes. In the context of the data placement problem, the files are the social actors and the interdependency relation between two files constitutes the relation between a pair of actors. Such a network is called a recommendation network.

- Structural properties: Structural properties concern the recommendation network graph as a whole or the subset of the graph. Primarily, the centrality of the social actor or the relation between a pair of actors is the main point of analysis. Some works which analyze social networks for structural properties are presented in [120, 121, 122].

The most important structural property to investigate in a social network is the different centrality measures [123]. Some of these centrality measures include degree, closeness, and betweenness centrality [124] and PageRank centrality [125]. PageRank centrality is especially famous for it is the measure that revolutionized web search engine technology.

Relation between recommendation systems and online social networks works both ways. The works in [126, 127, 128], illustrates how social networks have been leveraged to augment recommendation algorithms and in [129], an example where a recommendation system is being used to enhance social networks is presented.

2.6.4 Simulation Tools: NS3, HDFS, Cloudsim, Gridsim

It is rarely possible to obtain a full fledged distributed system to experiment upon. Most of such systems are costly or are in use and could not be taken offline for experimentation purposes. In such cases, simulation is an important evaluation tool to empirically validate the performance of a proposed algorithm. Simulation closely models the function of the distributed system being investigated and gives fairly accurate performance measures as one would have while experimenting on the real system.

In distributed systems, Cloudsim [130] and Gridsim [131] are used to simulate cloud and grid systems respectively. Both these systems simulate the entire cloud and grid system respectively and in cases where only the storage system's performance is to be investigated, this might be unnecessarily complicated.

Distributed storage system essentially consists of two components; a storage system and a network that connects the systems. In such a case, HDFS and network simulators such as NS-2 [132] or the more advanced NS-3 [133] could be used to specifically study the distributed storage system with much simplicity.

Chapter 3

Preliminary Studies on Reliability, Data Dependency and Distributed Storage Netwoks

3.1 Introduction

In this chapter, some important concepts will be introduced that will be useful to understand the works described in later chapters. More importantly, these concept discussions will accompany novel research work done and experiments, to reveal some important facts.

In this thesis, two topics: reliability and data placement are considered in the context of distributed systems. The concept of distributed systems has been introduced in detail in chapter 2. This chapter elaborates the concepts of reliability and data placement on such systems. Further, to study distributed systems, often a physical system is not readily available. In such cases, simulation is a very convenient alternative to study the performance of a system. NS3 is a robust network simulation environment, improving on the very popular NS2. Since distributed systems are spread across a network, analyzing network performance is an important point of study. In this chapter, NS3 has been extended to support simulation of one such popular distributed file system: HDFS, such that network specific performance aspects of HDFS can be studied through simulation by leveraging the robustness of NS3.

This chapter is organized into five sections including this introduction section. Concepts of reliability and data placement in distributed systems are explained in Section 2, through two novel research experimentation. In Section 3 HDFS extension of NS3 is discussed. In Section 4, layout of future chapters is discussed. This chapter is concluded in Section 5.

3.2 Data Reliability and Data Placement in Distributed Systems

In this section, the concepts related to data placement in a distributed system is introduced. These concepts would be explored further through two novel experiments.

3.2.1 Scientific Workflow

A scientific workflow is a specialized workflow management system for science applications that is aimed at specifying a series of computation and data manipulation steps for obtaining a result in scientific research. Science workflows are intuitive and user friendly science analysis tools, used by researchers who may not be proficient with a more technical or programming based approach to computation and data manipulation.

Another major advantage of scientific workflows is their enabling ability for collaborative research. Scientists distributed across the world may utilize a single workflow on a distributed system for executing large scale experiments or knowledge discovery. Such ease of collaborative research is a major stepping stone in advancements in the scientific knowledge of humankind.

Distributed systems offer numerous benefits as discussed in the previous chapter. However, such advantages are possible only when intelligent decisions are taken in terms of managing such systems. One such important management decision is that of data placement and ensuring data localization. In subsection 2.4.3, the concept of data placement, data localization is discussed along with a very intuitive process for localized data placement in scientific workflow systems.

The distributed system being used for this work is shown in Figure 3.1. The functional unit of the system consists of a local computational cluster, which consists of five compute cum storage nodes. These nodes serve as sites for task execution and file storage. These nodes are interconnected through a local network. The entire cluster is connected to data backup repositories, through the internet. The local computational cluster may be distributed across a large geographic distance, but their interconnection network is different from the typical internet; hence, the name local network. The local network is dedicated to connecting the nodes of the local cluster exclusively. Similar local clusters have a completely different local dedicated network for their exclusive use.

During task execution, a scientific workflow is deployed into the local computation cluster; the necessary data is downloaded from the appropriate repository on the internet. The data and task are placed at the relevant nodes by a task scheduling and data placement algorithm respectively. On completion of task scheduling and data placement, the workflow starts its execution.

3.2.2 System description

Data transfer is kept minimal in both the internet and the local network of a cluster. However, practical constraints make data transfer necessary. When data transfer needs to happen, it is preferable to keep it limited to the local network only. If the data requirement can not be satisfied by nodes in the local cluster, the global repository on the internet is accessed. The design goal of the data placement system is to minimize data transfer in the local network and internet, particularly the internet.

Nodes in the local cluster are homogeneous within the cluster; i.e, the characteristics of the node's storage computational and communication abilities are the same across all the nodes in the system. However, nodes from the different clusters may be heterogeneous.

Nodes of the system are considered unreliable unless otherwise stated. Thus, the nodes randomly suffer error conditions; a state in which they are not able to store data and compute any task, or communicate with other nodes in the cluster. This condition exists for some time before the nodes recover and come back online. A node is considered non existent during this period of error. During the error period, data stored in the concerned node is considered lost. If any other node requires the data from that node, it has to find an alternative source, preferably from other nodes in the local cluster or from the backup repository on the internet. During simulation, the nodes are considered to fail following a Weibull distribution [134].

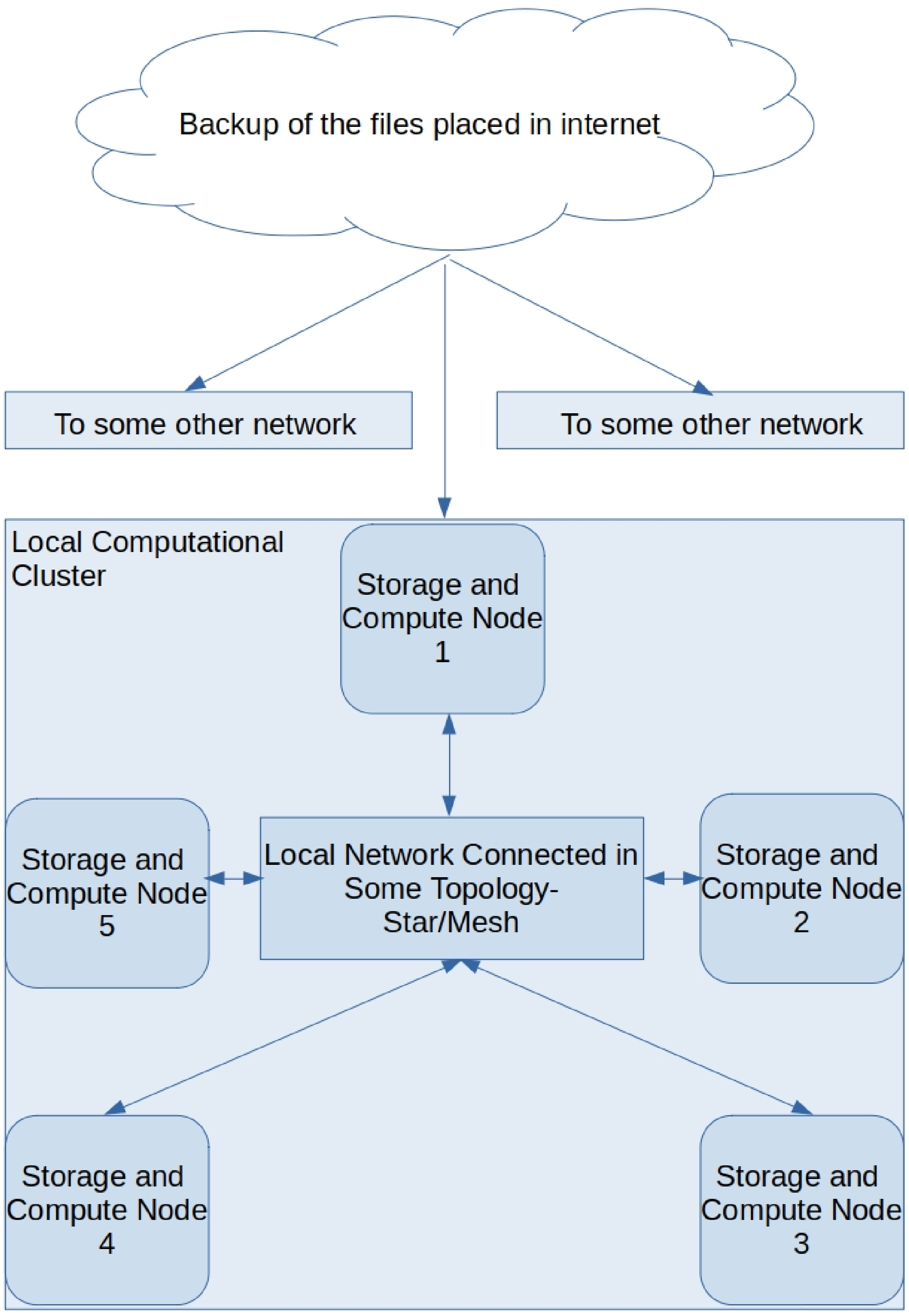

Backup of the files placed in internet
To some other network
To some other network
Local Computational Cluster
Storage and Compute Node 1
Storage and Compute Node 5
Local Network Connected in Some Topology- Star/Mesh
Storage and Compute Node 2
Storage and Compute Node 4
Storage and Compute Node 3

Figure 3.1: Distributed System

The system closely follows Hadoop's [135] system architecture and the file system is built around the Hadoop Distributed File System (HDFS) [44]. The core difference is that map reduce is not used in this case and the entire scientific workflow is already partitioned into atomic units called tasks. Further, atomic data units are files and not blocks as used in HDFS.

3.2.3 Reliability: Data and Replica placement

Data Interdependency

A workflow is a description of tasks: the fundamental unit of execution and a graphical representation of task execution. During its execution, each task requires a set of data that it can manipulate or compute upon. Hence, a description of tasks among other things consists of the set of files, that the task requires during its execution. The purpose of any data management sub system is to ensure that data is available to the executing task as fast as possible.

When a workflow is deployed in a distributed system, different tasks are allotted to different execution sites (nodes) of the system. Due to the distributed nature of the system, both in terms of data storage and task execution, it may be possible that a task and the files required by the task are placed at different sites of the distributed system. This type of data placement is known as non localized data placement.

A non localized data placement results in serious system performance penalties. During task execution, if a file is not present at the site of task execution, these files have to be migrated from remote sites in the local cluster or backup on the internet. Such migration is a time consuming affair, during which the task has to be stalled due to the unavailability of data. Further, frequent data migration leads to the wastage of precious network resources. Due to these factors, an intelligent data management sub system of a distributed system tries to avoid non localized data placement and prefers a localized data placement.

As described earlier, each task contains a set of files that will be required by it during its execution. From the set of all tasks, we can invert the set and find a set of tasks for each file, that requires the file for its execution. Considering any pair of files, when their set of tasks is compared, they are likely to exhibit one or more common tasks. These common tasks make the file pairs dependent on one another and hence, these files are called interdependent files. Higher the number of common tasks a pair of files exhibit, the higher is the interdependency between the files. Mathematically, interdependency between files can be represented by the equation as follows:

$$Interdependency(File_n, File_i) = |ds_n.Task_n \cap ds_i, Task_i| \qquad (3.1)$$

Where, $n = 1$ and $i = 3$.

The equation finds interdependency between files $File_1$ and $File_3$, where the set of tasks which require $File_1$ is described by $ds_n.Task_n$ which for $n = 1$ is $ds_1.Task_1$ and the set of tasks which require $File_3$ is described by $ds_i.Task_i$ which for $i = 3$ is $ds_3.Task_3$. The intersection between the two sets gives the number of tasks which requires both $File_1$ and $File_3$ during their its execution. Hence, the number of such common tasks, defined as the interdependency between files $File_1$ and $File_3$, is obtained by $|ds_n.Task_n \cap ds_i, Task_i|$.

The interdependency metric defined in equation 3.1, represents the number of tasks that require both files File1 and File3. Thus, higher the interdependency between the files, it is more likely that when one file in the pair is required by a task, the other file will also be required by the task. When both the files are placed at different sites and they are highly interdependent, both files have to be frequently migrated to the other's site creating data migration problem. Thus, to prevent data migration and the resulting performance degradation, two interdependent files are to be placed together at the same site of the distributed system (known as colocation) thereby ensuring data localization. The necessity for data localization has been discussed above; however, the means to achieve that localization is not discussed. Interdependency between pairs of files is used to achieve the desired data localization.

Dependency Matrix and Data Placement

There are two classifications of data placement of scientific workflow in distributed systems:

- Task placement first: The tasks of the scientific workflow are scheduled for execution at the sites of the distribution system. The data is placed at each site of the distributed system based on the data requirement of tasks placed at each site.

- Data placement first: Interdependency between each pair of files is calculated and data is placed such that each site consists of a group of highly interdependent files. The tasks are scheduled at each site, depending on the result of data placement. A site that contains most of the files required by a task is the one in which the task is to be scheduled.

In this work and subsequent chapters, the data placement first approach is used except for chapter 4. This process utilizes the concept of interdependency between files as described by equation 3.1. The basic data placement approach using this metric is described below.

In Figure 3.2, the local cluster of the distributed system presented in Figure 3.1 is presented. Here only 3 nodes are shown and these nodes are connected through a local mesh network. Each node represents a data center, consisting of 3 storage cum compute servers. Since all the servers in all the nodes are identical, the servers at each node may be aggregated into a single compute cum storage entity. The tasks $t1$ to $t5$ have been scheduled based on the prior data placement. In effect, tasks scheduled at a particular node can be said to be highly interdependent; since both their data requirement are satisfied by the same node. Hence, if one task is scheduled at a particular site, the other interdependent task would follow suit.

In Figure 3.3a the set of tasks that require by each file is shown. Thus, $ds1.Task1 = t1, t4$ means that task $t1$ and $t4$ both require file File1 denoted by $ds1.Task1$. As an aside it may be noted that task dependencies here are different than what would be inferred from Figure 3.2. Here tasks $t1$ and $t4$ are highly interdependent and will be placed together in a node. Similarly, tasks $t1$ and $t5$ are highly interdependent and would be placed at the same node.

Interdependency between each pair of files may be found through equation 3.1. Thus, files File1: $ds1.Task1$ and File2: $ds2.Task2$ would be 2 since there are two common tasks. Interdependency between a file with itself could also be found, which will be equal to the

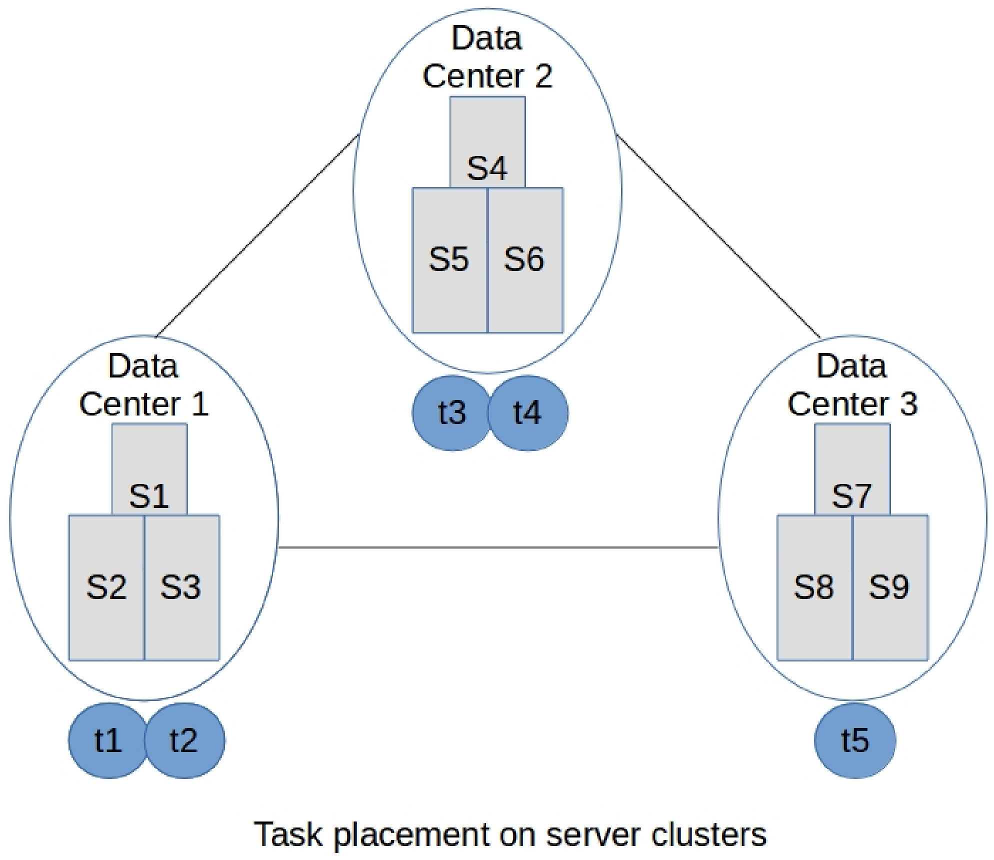


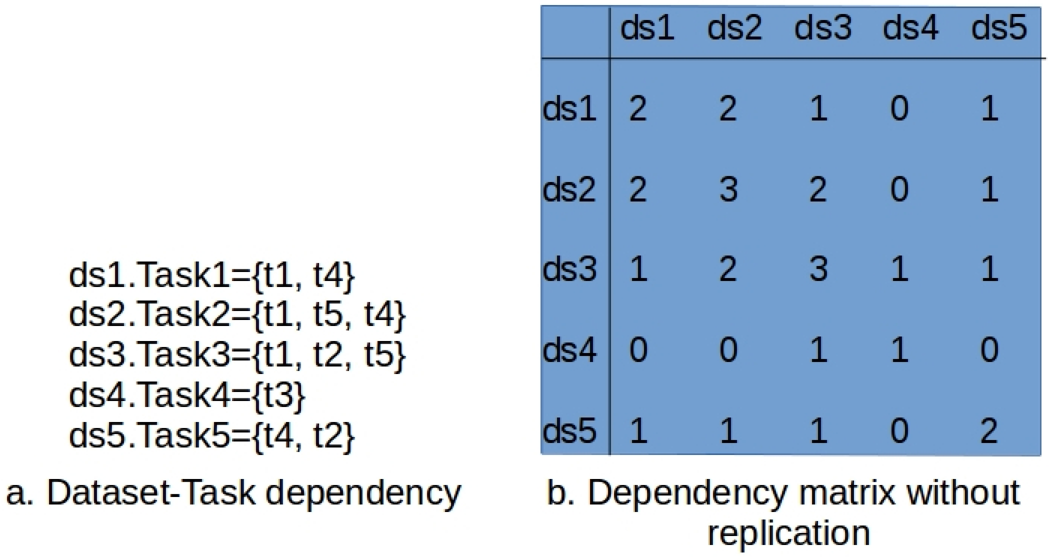
Figure 3.2: Task scheduling and data placement in distributed system

number of tasks required by the file. For file File1: $ds1.Task1$, since there are 2 tasks which require it, interdependency between file1 to itself is 2

The interdependency values may be arranged in terms of a matrix as shown in Figure 3.3. Each cell of the matrix represents the interdependency between the corresponding file pair. Thus cell ds1-ds2 (row1, column2) represents the interdependency between files File1 and File2 each of whose requirement by tasks is represented by $ds1.Task1$ and $ds2.Task2$ respectively. This matrix is referred to as the data dependency matrix.

Once the dependency matrix is available, different placement schemes may be tried, one of which is described here briefly. Such a dependency matrix may also be used for data prefetching, a concept that will be visited in Chapter 5 of this thesis.

An easy way to place files in the nodes of the distributed system is



ds1.Task1={t1, t4}
ds2.Task2={t1, t5, t4}
ds3.Task3={t1, t2, t5}
ds4.Task4={t3}
ds5.Task5={t4, t2}

a. Dataset-Task dependency

	ds1	ds2	ds3	ds4	ds5
ds1	2	2	1	0	1
ds2	2	3	2	0	1
ds3	1	2	3	1	1
ds4	0	0	1	1	0
ds5	1	1	1	0	2

b. Dependency matrix without replication

Figure 3.3: Data interdependency matrix

to arrange the files such that each file's adjacency to the adjoining file is determined by the interdependency between the files. Considering the dependency matrix, the goal is to rearrange columns such that the adjoining columns of a file represent the most interdependent file to the given column. This is achieved through the use of the Bond Energy Algorithm (BEA) [136, 137]. BEA rearranges the columns based on the interdependency values. The resulting arrangement of columns represents the list of adjacent files which are highly interdependent. Once the reordered columns are available, each file is placed at a node in order, till the node's storage limit is reached. The file placement then commences from the next node and goes on till all the files have been placed. This technique of data placement has been used in [83], which has been utilized in the development of Swin Dew C system [138], a cloud based distributed scientific workflow execution system.

3.2.4 Replica placement

There are two methods in which data reliability can be maintained in a distributed system: replication and error correction codes. Error correction codes are used only in the work described in chapter 4, otherwise, reliability is achieved through replication. Replication is the process of keeping multiple copies of a file (data) at multiple sites of the distributed system. If one site encounters data corruption, the data may be migrated from another site. Further, a file may be highly interdependent to two or more other files. It may be possible that other dependent files are placed at different sites of the system. Then, to satisfy data localization, the file interdependent on other files is copied as replicas and each replica placed at each relevant site.

In this section, an anomalous observation is described [139] in terms of data placement through the interdependency based metric introduced above. This anomaly is observed only when data placement, as described for Swin-Dew-C, is applied on replica placement; i.e., when more than one copy of the data is to be placed, the data interdependency based method encounters an anomaly.

Consider Figure 3.4, the results illustrated in Figure 3.4a and b are the same as that of Figure 3.3. The only addition is the third larger matrix in Figure 3.4c.

The rows and columns of matrix in Figure 3.4c consists of replicas of files $ds1$ to $ds5$. Only one replica is considered in this case. Thus $ds1$'s replica is $ds6$, $ds2$'s replica is $ds7$ and so on. Thus, files $ds6$ to $ds10$ are replicas of files $ds1$ to $ds5$.

Empirical Observations

To understand the anomaly, an experimental replica placement is conducted by implementing the above data placement scheme using C++. In the experiment, the placement of the data and their replicas on data centers (sites) of the distributed system will be observed. The observation is made for 2 cases: 1st case consists of no replication and only 1 replica per file; the second case consists of 2 replica and 3 replica per file respectively. The number of data centers, tasks, and size of files is kept constant. A set of tasks is built randomly and files required for each task are allotted randomly as well. The information is used for data placement as described above, with the dependency matrix for 0 replica and 1 replica cases being taking a similar form as in Figures 3.4b and c. The 3 replica and 4 replica case matrices are similarly built as in 3.4c.

The essential parameters of the experiment is presented in Table 3.1

The allocation of files or replica of files is shown in Figure 3.5 for

ds1.Task1={t1, t4}
ds2.Task2={t1, t5, t4}
ds3.Task3={t1, t2, t5}
ds4.Task4={t3}
ds5.Task5={t4, t2}

a. Dataset-Task dependency

	ds1	ds2	ds3	ds4	ds5
ds1	2	2	1	0	1
ds2	2	3	2	0	1
ds3	1	2	3	1	1
ds4	0	0	1	1	0
ds5	1	1	1	0	2

b. Dependency matrix without replication

	ds1	ds2	ds3	ds4	ds5	ds6	ds7	ds8	ds9	ds10
ds1	2	2	1	0	1	2	2	1	0	1
ds2	2	3	2	0	1	2	3	2	0	1
ds3	1	2	3	1	1	1	2	3	1	1
ds4	0	0	1	1	0	0	0	1	1	0
ds5	1	1	1	0	2	1	1	1	0	2
ds6	2	2	1	0	1	2	2	1	0	1
ds7	2	3	2	0	1	2	3	2	0	1
ds8	1	2	3	1	1	1	2	3	1	1
ds9	0	0	1	1	0	0	0	1	1	0
ds10	1	1	1	0	2	1	1	1	0	2

c. Dependency matrix with replication

Figure 3.4: Data interdependency matrix (with Replication)

Table 3.1: Essential Parameters

Parameter	Value(s)
Number of datacenters	10
Datacenter capacity	3 datasets per datacenter
Number of replica(s)	0, 1, 2, 3
Number of tasks	10
Number of distinct datasets	5

0-replica and 1-replica case and Figure 3.6 for 2 replica and 3 replica case.

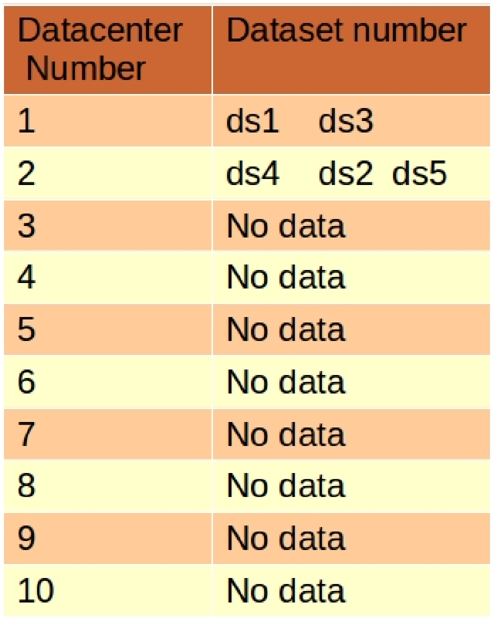
Let us consider Figure 3.6, which represents the anomalous 2 replica and 3 replica placement case (multiple replica placement). It may be observed from Figure 3.6a, which represents 2 replica case, replicas $ds6 - ds11$, $ds8 - ds13$, and $ds10 - ds15$, which are all replicas of the same file are placed at the same site or datacenter. In Figure 3.6b, which represents 3 replica case, a similar observation of replicas being placed at the same site can be observed: $ds14 - ds19$, $ds7 - ds12$ and $ds15 - ds20$.

From Figure 3.5a, i.e., the no replication case, a good data placement can be observed as expected, in case of 1 replica also; i.e., no replicas are placed at the same datacenter.

Discussions

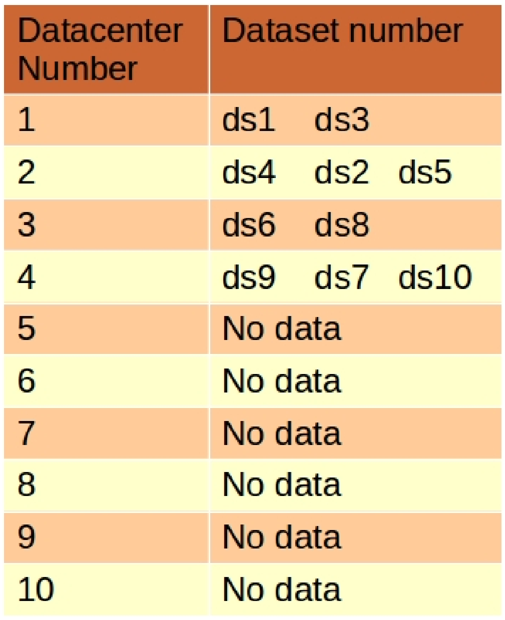
The placement of replicas of the same file to the same data center runs counter to the objective of replication as follows:

- Reliability: If a system maintains 3 copies of a file (main copy + 2 replicas), with the assumption that if one node failure occurs, 2 copies of the file may survive. In the above observed scenario, loss of 1 node results in the unavailability of 2 copies, and only 1 copy is left. If one more node fails, the system will lose all the copies of the file.
- Availability: 3 copies of the file may have been required because the file is highly interdependent to file clusters at three different nodes of the system. Hence, placing two copies of a file at one data center would deprive localization at another data center.



Datacenter Number	Dataset number
1	ds1 ds3
2	ds4 ds2 ds5
3	No data
4	No data
5	No data
6	No data
7	No data
8	No data
9	No data
10	No data

a. Allocation of 5 datasets with no replica on 10 datacenters



Datacenter Number	Dataset number
1	ds1 ds3
2	ds4 ds2 ds5
3	ds6 ds8
4	ds9 ds7 ds10
5	No data
6	No data
7	No data
8	No data
9	No data
10	No data

b. Allocation of 5 datasets with 1 replica on 10 datacenters

Figure 3.5: Results for no replica and 1 replica

Datacenter Number	Dataset number
1	ds1 ds3
2	ds4 ds 2 ds5
3	ds6 ds11
4	ds8 ds13 ds9
5	ds14 ds7
6	ds12 ds10 ds15
7	No data
8	No data
9	No data
10	No data

a. Allocation of 5 datasets with 2 replica on 10 datacenters

Datacenter Number	Dataset number
1	ds1 ds3
2	ds4 ds 2 ds5
3	ds6 ds11 ds16
4	ds8 ds13
5	ds18 ds9
6	ds14 ds19
7	ds7 ds12
8	ds17 ds10
9	ds15 ds20
10	No data

b. Allocation of 5 datasets with 3 replica on 10 datacenters

Figure 3.6: Results for 2 replica and 3 replica

Further, any other highly dependent file which could have been placed in the place of the redundant copy. This is now not possible further aggravating data nonlocalization. All these results in data migration during task runtime, which could have been avoided

- Load imbalance: When multiple replicas are placed at different nodes, the system can balance data migration load among multiple node; thereby reducing the load on a single node. With multiple replicas being placed at a single node, a fewer number of nodes are available for data migration needs. Hence, the system is deprived of its load balancing capabilities.

The reason for placing multiple replicas of a file at the same datacenter could be attributed to the following inequality:

$$|ds_n.Task_n \cap ds_n.Task_n| \geq |ds_n.Task_n \cap ds_i.Task_i| \tag{3.2}$$

Where, $n \neq i$.

The inequality 3.2 means that the interdependency of a file with itself is always greater than or equal to the interdependency of a file with another file. In some cases, a file's self interdependency may be the highest. In the traditional method of data placement, the replica is considered just like another file. Hence, interdependency between the replicas of the same file would also be highest in these cases. Under such circumstances, the data placement algorithm will place the replicas together in place of another highly interdependent file.

The solution lies in informing the data placement algorithm which files are the replicas of another file and the algorithm treating their interdependencies as self interdependency. This way, the algorithm ensures that only interdependency between two different files are considered as the true interdependency.

3.2.5 Reliability: File Fragmentation based Placement

In the earlier discussion, a file is treated as an atomic unit while making its placement decision. In such a situation, a file assigned to be placed at a node is placed in its entirety if storage space is available, else the entire file is not placed.

This type of file placement has some disadvantages. Consider Figure 3.7a; file $f1$ is to be placed in Node $n1$; however, there is not sufficient space to place the entire file on the node. Thus, the file is not placed and at runtime, the entire file would have to be migrated. Further, if the free space available is not enough to accommodate any file, the free space would be wasted.

Now, considering Figure 3.7b, if the file is split into three equal parts then, two fragments of the file: $s2$ and $s3$ could be accommodated in the available space. In this case, only fragment $s1$ would have to be placed elsewhere and would have to be migrated during runtime. Thus, the amount of data migration at run-time has been reduced by about 66%; which significantly enhances task execution performance by cutting down task stall time. Also, the available free space in the node has been utilized fully.

Considering the above argument, it is tempting to try a file placed in a distributed system by splitting the file and placing the fragments instead of the entire file. This work [140] describes a study to examine the feasibility of this approach through empirical observations.

Motivation for File Fragment Placement

There are several motivations for such a placement technique.

The most obvious motivation is that of paging in operating systems memory management module. In paging, the entire data unit is split into multiple fragments, and only necessary fragments are brought and placed into the memory.

Distributed systems may be viewed as an extension of the monolithic system's memory hierarchy. In a monolithic system, memory hierarchy ends with the main memory of the device. Sometimes the secondary memory may be considered as an extension of the main memory since unused pages are stored and retrieved from the secondary storage. In distributed systems, one more memory hierarchy is added which is the secondary storage in a separate device connected through a local network. When the internet based backup repository is concerned, one more extension is added to the hierarchy, i.e., the backup repository. Thus, in a distributed system, the highest memory hierarchy is the processor registers, followed by cache memory, main memory, secondary storage of the device, secondary storage of

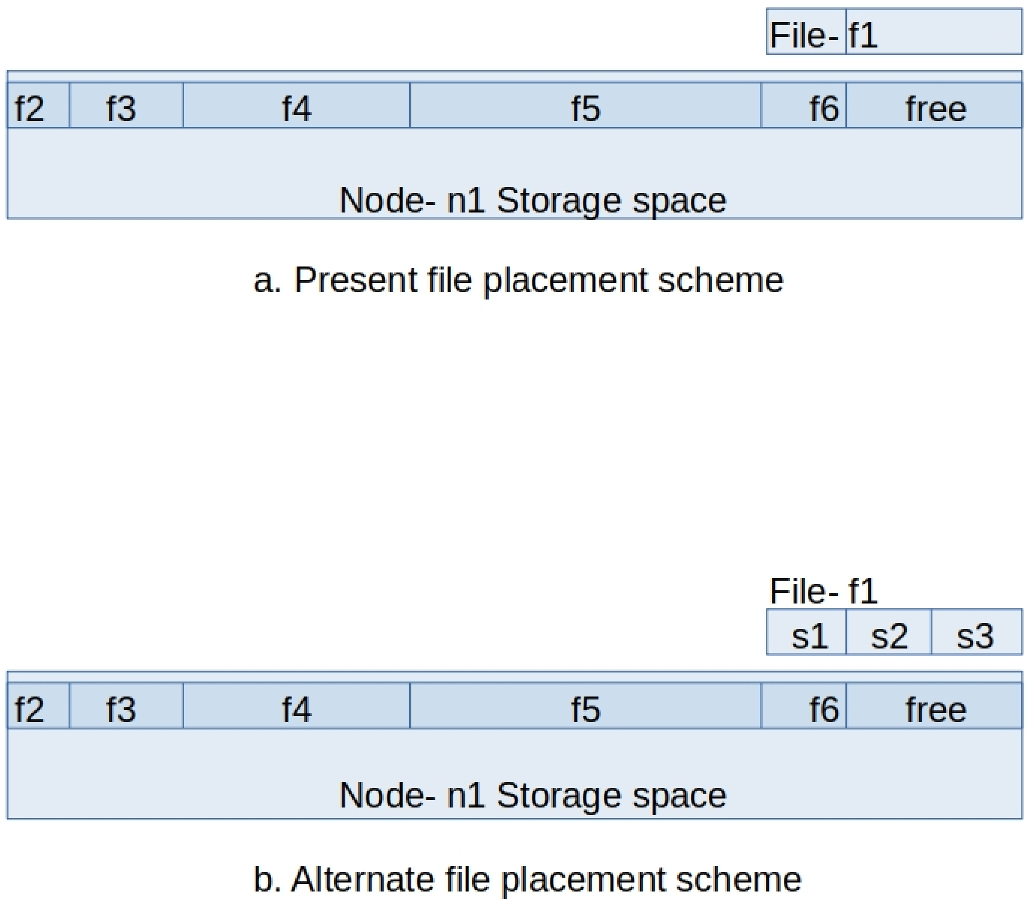


Figure 3.7: Fragmented vs Non fragmented file placement

another device in the local cluster, and the backup repository on the internet, in order.

The second motivation is the Hadoop Distributed File System (HDFS). In HDFS the atomic unit of storage is a block of data, which may be smaller than the size of a file. However, if the interdependency based placement is attempted, the interdependency would be found for each block and the blocks placed as per their mutual interdependence value. This is the same as replacing the file placement method described above with a block. The point of concern in this work is that interdependency is known between the files, not between the sub units of files such as blocks or fragments. This is where this work is different from HDFS.

Further motivations can be obtained from the fragmentation of workflows into segments and storage methods like bit torrent protocol, discussed in the literature survey.

Emperical observations

To conduct the feasibility study, an experiment has been performed to compare the performance of data placement traditional vs fragment based. A distributed system similar to that illustrated in Figure 3.1 is considered. The interdependency between the files are randomly determined. Since the point of concern is the actual placement and not the calculation of interdependency value, the random interdependency determination is deemed not to affect the outcome of the experiment. Once the interdependency values are available, files are assigned to respective nodes. The actual placement depends on the availability of the space in the node. The file size is randomly determined, so is the capacity of the node. Once again these random decisions have no bearing on the results produced by the experiment. In the fragmented approach, the number of fragments equals the number of nodes. The network traffic generated under the following circumstances are observed:

- Traditional vs Fragmentation approach: The network traffic generated is compared between the traditional approach of placing the entire file vs the alternate proposed approach of splitting the files into fragments and placing them.
- Internet vs P2P(or local cluster): The traffic on the internet and that in the P2P network is separately measured. Among the traditional and alternate proposed approaches, minimal traffic is expected. For the network traffic that gets generated, which placement minimizes traffic in the internet is observed.
- Error vs no error nodes: The traffic generation by both the placement approach is measured in both error conditions: cases where data placed in nodes become unavailable due to failure and no error condition: every node can cater to the data needs.
- Cummulitive vs Per file: A task might require every file in its set of file requirements before starting its execution. Another approach is that a file might need one file at a time during its execution. In such a case a task executes when its current file

requirement is satisfied. As the task executes, the next required file is migrated. This is the per file approach. The experiment compares both these approaches.

The essential parameters are provided in Table 3.2 beolw:

Table 3.2: Essential Parameters

Parameter	Value(s)
Number of nodess	20
Total number of files to be assigned	50, 100, 150, 200
Total number of files to be placed	10, 20, 30, 40
File size	Variable 1GB to 800 GB
Number of files assigned per node	random
Number of files (entire file) assigned per node	2 (selected randomly)

Results and Discussions

The results of the experiment are available in Figures 3.8 to 3.11.

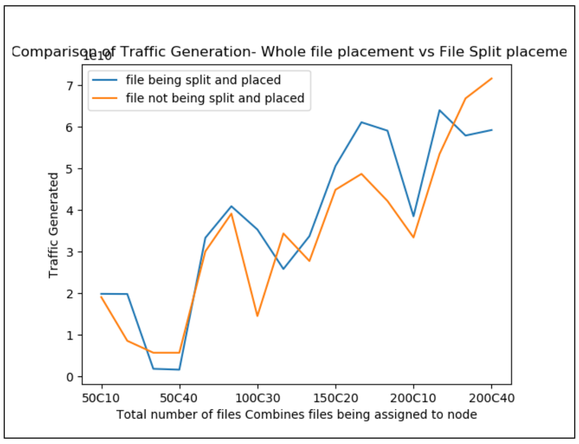


Figure 3.8: Traffic generated on a cumulative basis in the internet with some nodes in local clusters going offline

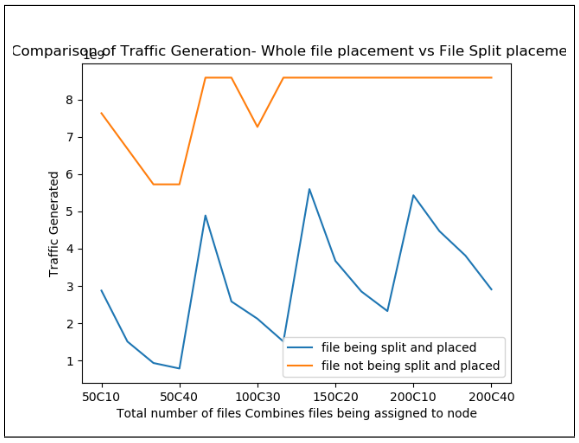


Figure 3.9: Traffic generated on a per file basis in the internet with some nodes in local clusters going offline

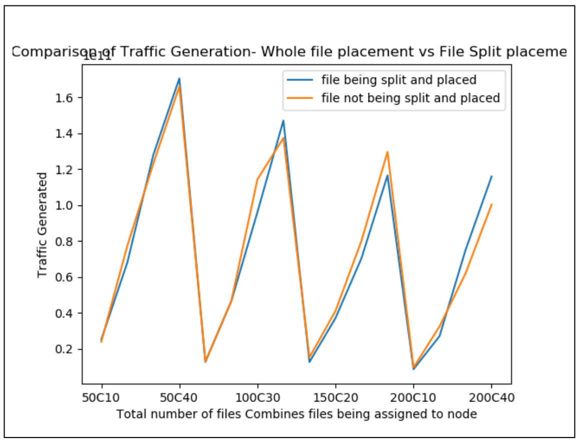


Figure 3.10: Traffic generated on a cumulative basis in the P2P network with some nodes in local clusters going offline

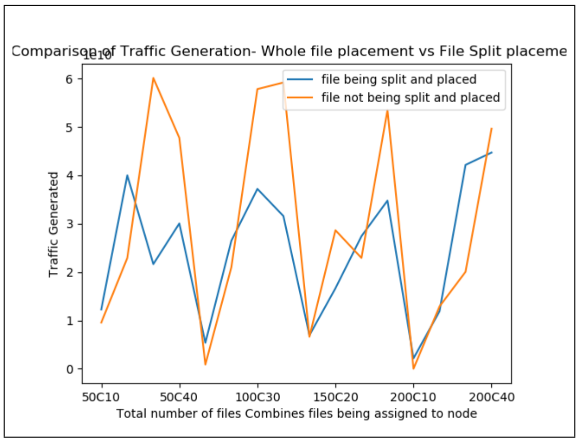


Figure 3.11: Traffic generated on a per file basis in the P2P network with some nodes in local clusters going offline

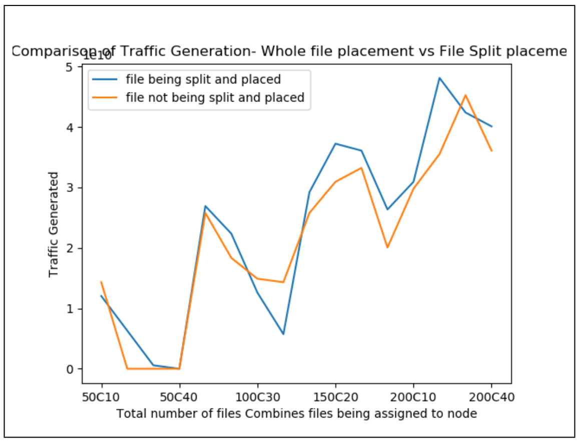


Figure 3.12: Traffic generated on a cumulative basis in the internet with no nodes in local clusters going offline

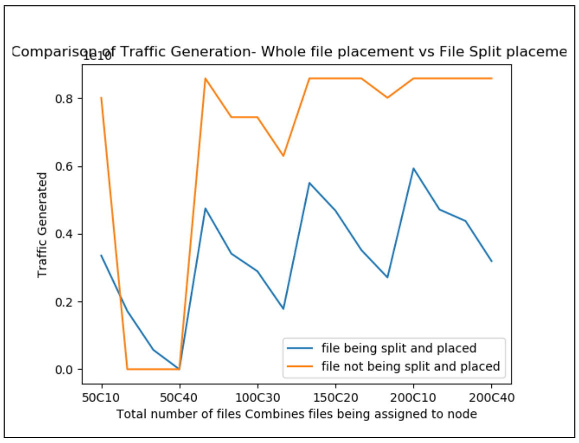


Figure 3.13: Traffic generated on a per file basis in the internet with no nodes in local clusters going offline

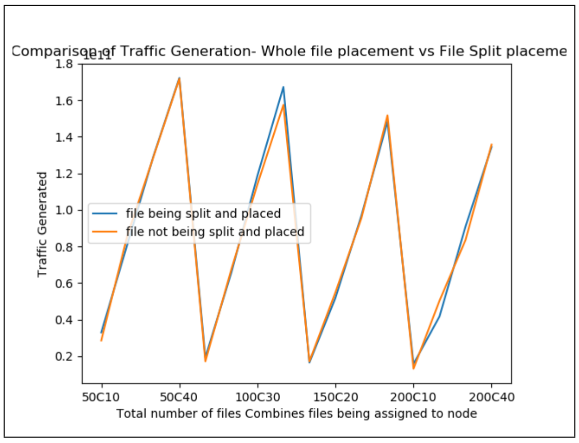


Figure 3.14: Traffic generated on a cumulative basis in the P2P network with no nodes in local clusters going offline

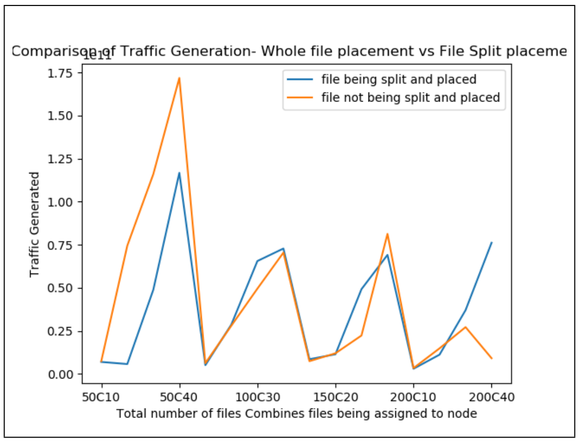


Figure 3.15: Traffic generated on a per file basis in the P2P network with no nodes in local clusters going offline

From the results the following inferences could be drawn:

- When the per-file case is considered, the approach of splitting the file and placing the fragments generates far less network traffic in almost all the cases: internet vs p2p, cumulative vs per-file, and error vs no-error condition. The results are presented in Figures 3.15, 3.13, 3.11 and 3.9

- In the cumulative case, both approaches exhibit identical results for p2p and both no-error and error conditions. In the case of internet traffic somewhat greater network traffic is observed for the fragmentation-based approach. The results are presented in Figures 3.14, 3.12, 3.10 and 3.8

The network traffic generated is due to data migration resulting from non localized file placement. Thus, less the network traffic, more localized file placement is achieved. For the per file case, the fragmentation based approach yields the most localized placement under all the conditions. In the case of cumulative cases, both the approaches are tied with the traditional approach performing better in the case of the internet.

It may be noted that a random decision has been taken in regards to the placement of the fragments. This may have had some impact on these results. Making an intelligent choice on which fragment is to be placed at which node could play a role in a better placement, resulting in high localization.

However, considering that random decision making has yielded this result, similar if not a better result could be expected from intelligent decision making regarding fragmentation. Thus, at the very least splitting a file into fragments and placing these fragments yield good localization when a task requires one file at a time. In case when a task requires all the files at once, future work with intelligent decision making on fragmentation would have to be developed and studied.

3.3 Extension of NS-3 for HDFS Simulation

Since evaluating the performance of an algorithm on a physically distributed system is not possible, simulation tools are required to comprehensively analyze the performance of an algorithm through empirical evaluation. Good simulation tools for cloud and grid simulation are available [130, 131]. Most of the simulation in data placement requires only measuring network characteristics like network usage; hence, using cloud simulation tools becomes overkill, and setting them up is tedious. Hence, a simpler simulator to measure the distributed system's network performance has been developed [141].

3.3.1 Brief introduction to HDFS and NS3

The relevant concepts in regards to HDFS and NS3, required to understand the extension are described here.

Hadoop Distributed File System (HDFS)

HDFS is the distributed file system used to support Hadoop's data storage and retrieval requirements for big data analysis. Thus HDFS has been developed keeping in mind the following design goals:

- Hardware failures: Hardware failures are considered to be a common occurrence and HDFS is expected to deal with such frequent hardware failures.
- Batch job: HDFS expects to host data for use by batch jobs. This eliminates the low latency requirement while high I/O throughput is the performance criteria of concern.

- Big data: Since Hadoop is geared towards big data analytics, HDFS needs to support the characteristics of big data as well.
- Data localization: HDFS tries to optimize data localization for the same reason as discussed in the last section.

Considering the design goals in mind, HDFS observes the following design principles:

- Replication: In HDFS files are stored as blocks of 64 MB each, except the last block and each file has 3 replicas. These are default numbers and are reconfigurable. HDFS has its own replica placement algorithm and policy, but other replica placement schemes as will be discussed in the subsequent chapters may be used. After initial replica placement, replication may be triggered if underreplication is detected.
- File system metadata persistence: HDFS has two kinds of nodes; NameNode and DataNode. NameNode stores the file system metadata like I/O activities on the files, change in replication factor, etc. These are recorded in a file called EditLog stored in the NameNode. EditLog file is used for HDFS or checkpointing purposes as well.
- Robustness: In HDFS NameNode, DataNode or network partitions may fail. HDFS is designed to ensure the minimal effect of the failures on the continued working of the system. The robustness implemented by HDFS to counter the effect of the failures are as follows:
  - DataNode failure: The status of each DataNode is reported by periodic Heartbeat message to NameNode after periodic intervals. If NameNode does not receive a heartbeat message from a DataNode after a fixed time, that DataNode is assumed dead. No further I/O is sent to that DataNode and a rereplication is carried out if under replication occurs due to the dead DataNode.
  - NameNode failure: There are multiple NameNodes maintained, and periodically copies of EditLog and FsImage are

backed up in the backup NameNodes. Once a NameNode fails, one of the backup NameNode assumes the responsibility and starts from the checkpoint in the last EditLog backed up.

–Data integrity: Individual blocks may fail, even when the entire DataNode may be otherwise functional. NameNode checks the checksum to detect data integrity in the blocks and blocks which have anomalous checksum are discarded. Periodic BlockReport sent to NameNode helps NameNode keep track of the data integrity of the blocks. If underreplication occurs due to discarding the blocks, rereplication is triggered.

Having discussed the design principles and assumptions, the design architecture of Hadoop can now be discussed.

Data are deemed to be stored in datacenter, a generic name to define a collection storage elements. Each datacenter is composed of racks and each rack contains a set of nodes which have storage and computational power. The schematic datacenter design is illustrated in Figure 3.16

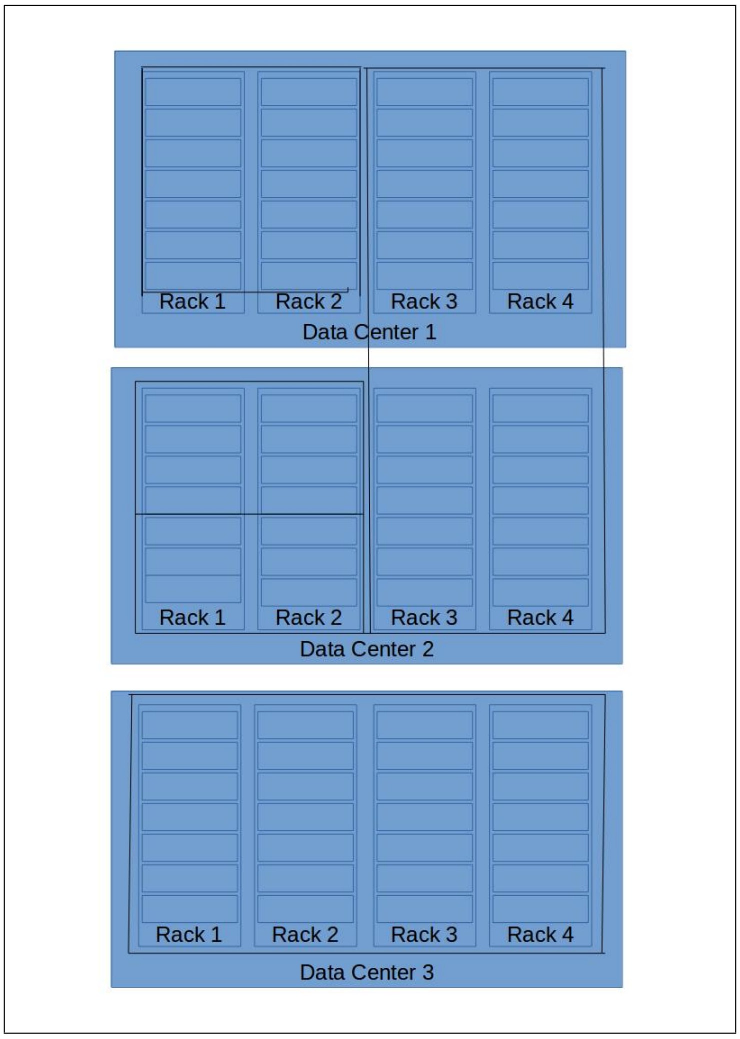


Figure 3.16: Physical Datacenter architecture [1]

Logically, the architecture is somewhat different. The functional unit of HDFS consists of a single node designated as a NameNode and multiple nodes designated as DataNode under the control of NameNode. This collection of NameNode and its DataNodes is called a cluster. The NameNode and the DataNodes may be from the same or different racks in the same or different data center. One functional NameNode is used per cluster, while there might be multiple nodes in the cluster kept as standby to assume NameNode responsibility when the functional node fails. The actual data storage and retrieval are performed on the DataNodes. The logical architecture is illustrated in Figure 3.17

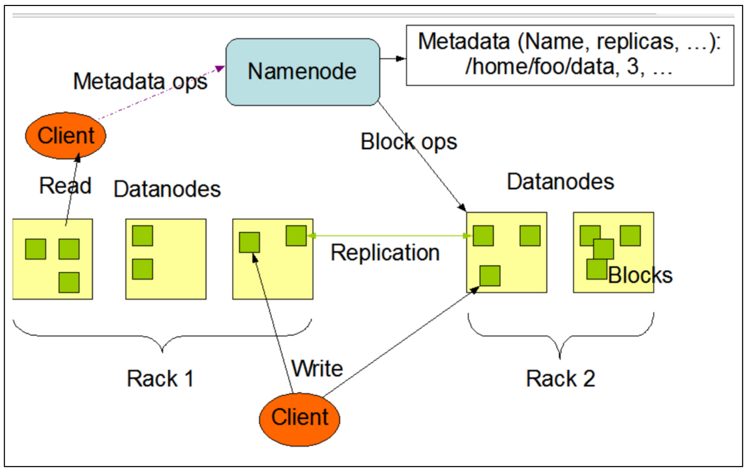


Figure 3.17: Logical HDFS cluster architecture [142]

NS3

NS3 is the new version of the popular discrete network event simulation tool NS-2. The version upgrade has come with a complete overhaul of the tool, with the new tool being implemented entirely in C++ and can be used through C++ or python scripting. NS3's improvement allows it to be more flexible and scalable than NS2 with the facility for real world testbed implementations.

A data center network is not natively implemented in NS3. However, an extension of the NS3 for implementing data center networks has been proposed by NTU's DCN implementation. It is this extension that is being extended to implement HDFS. For topology, Fat tree topology is preferred, due to its high throughput and low latency properties.

3.3.2 NS3 Extension for HDFS Implementation

The extension of NTU's DCN extension of NS3 is being further extended in this work to implement the HDFS. The details of this project constitute this section[2].

Brief Description of the System

Without sacrificing the simulation ability, some of the functionalities of HDFS have been omitted or simplified while extending the NS-3

[2]https://github.com/hindolb/NS3

for HDFS as follows:

- No actual data is saved; file is modeled as to contain data of a particular size.
- File system operation times like seek, open, etc are disregarded: We are interested only in measuring network usage and time spent in migrating the data. Hence, these node hard disk specific operations can be disregarded.
- NameNode failure is not considered: NameNode is not involved in data storage or migration. Thus their failure is inconsequential to network usage measurement due to data migration
- All DataNoodes and NameNodes are respectively considered homogeneous.
- Network errors are disregarded: Network errors or delays do not contribute to the study of data localization. Hence, it can be disregarded.

In implementing HDFS through the extension of the NTU DCN model of NS3, the following changes have been made:

- NS3 has a basic node container object. A subclass of HDFS server type is created by extending this object. The HDFS-server class is further subclassed into DataNode and NameNode subclass. Due to the subclassing, HDFS server inherits the characteristics and function from node class, the combined characteristics of the HDFS server and node class are inherited by NameNode and DataNode class.
- The following helper functions have been added:
  - Functions to introduce errors (DataNode and block error) as configured by the user.
  - Data placement: The default or user implemented data placement algorithm is present. A function either calls the default algorithm or any other user defined algorithm.
  - Task placement: By default, tasks are placed randomly. A specific task placement algorithm could be also called.

  - Data migration: On completion of data and task placement, a list of non local data is prepared. The data from the list would require migration
  - Server selection: When multiple replicas are present in different nodes, a selection needs to be made on which server's replica is to be chosen. Either the default Hadoop algorithm could be used or a user defined algorithm could be called.

- Application data transfer is implemented by extending NS-3's TCP Bulk send and Packet send application. Application data consists o data like read request and reply, metadata reply, block-report, heartbeat, etc.

Description of Classes and Helper Functions

The proper description of the NS3 C++ classes used and modified for achieving HDFS simulation is provided here:

- HDFS server: The generic class Node Container which describes a node in NS3 is extended by creating a sub class HDFSServer, such that the specialized role as HDFS nodes can be implemented. An HDFS node or server can be further classified into DataNode and NameNode. Thus HDFSServer class can be further subclassed into the following classes:
  - DataNode container: This is the subclass that extends the HDFSServer class such that specific functionalities and characteristics specific to DataNode
  - NameNode container: Like the DataNode container, the specific characteristics and functionalities of NameNode are implemented by NameNode class which is again a subclass of HDFSServer class.

- TCP Bulk Send Application: This class has been modeled similar to the MyApplication class in seventh.cc file of NS-3. The modification is made to the use of time defined stop; the new definition stops transmission and closes socket when the entire packet has been transmitted.

- Error Correction and Recovery: This class is used to mimic the failure of nodes during HDFS runtime as well as subsequent recovery of the node during the runtime of HDFS.
- Server selection: This is a helper function that determines the best DataNode which should be used for data access. While a default policy has been implemented, the user may modify the policy as per his/her necessities.
- Algorithm: This helper function calls the data placement algorithm that the user wants to use. By default Hadoop's algorithm is used and called by this unction. A user may define his/her function which implements a different algorithm. In that case, a user must configure to call the particular function.
- Task placement and dependency calculation: These are two helper functions. The helper function performs dependency calculation is called at startup. Post the dependency calculation, the Algorithm function is called; which places the data on the Hadoop DataNodes. Followed by data placement another helper unction is called which carries out task placement.
- PacketSink: Besides the Node Container class, PacketSink class of ns3 has been subclassed to prevent the invocation of stop function at a scheduled time as NS3 implements. The stop function is invoked on the completion of reading the entire packet.

Working Details

The flowchart of the working principles of HDFS NS3 is shown in Figure 3.18. A brief description of the simulator's work is explained as follows:

Step1: The topology of the network is defined and DataNode and NameNode objects are instantiated. TCP/IP is installed and an IP address is assigned. Dependency between the data units is calculated.

Step2: A data placement algorithm is invoked, which by user's configuration, calls the Hadoop's default implementation or a different algorithm's implementation. The data is placed as blocks in the DataNodes as per the algorithm invoked.

Step3: Periodic events like heartbeat message, block report, and making a node "dead" to simulate error conditions are scheduled, such that they may be triggered.

Step4A: Block read requests per DataNode is scheduled. Metadata request is sent to NameNode which replies by returning the list of servers that contains the replica of the required block. On receipt of the list of servers, a packet transfer is requested using the customized MyApp application. This is queued by the NameNode.

Step5A: The metadata requests from DataNode are processed by the NameNode one by one from the queue. A TCP bulk reply is sent by the NameNode to the DataNode which sent the request, The reply contains the list of servers that can satisfy the DataNode's particular data requirement.

Step6A: The metadata reply on being received by the DataNode, it finds the best server from the list and a request is sent for the data to the desired DataNode server.

Step7A: The data serving DataNode server on receipt of data request sends a MyApp reply mimicking the data block.

Step8A: The requesting DataNode on receiving the data block mimicking packet, removes the concerned block from its read request queue.

Step4B: Each DataNode sends a MyApp packet mimicking either a heartbeat message or block report message to the NameNode.

Step5B: NameNode on receiving the messages, records the events

Step6B: On trigger of a node or block failure event, NameNode on receiving the next Myapp message mimicking as heartbeat or block report message from the particular DataNode makes the necessary adjustments. On receipt of heartbeat, the "dead" node is marked unavailable. While on receipt of blockreport, the particular block is added to the list of free spaces, while deleting the record of the replica that it held.

Step4C: At specific intervals, rereplication events are scheduled. During this NameNode examines whether a block is under replicated. If so, under replicated blocks are rereplicated.

Step9: For ending the simulation, the time based ending of NS3 has been replaced with the event based ending. On completion of all scheduled events in all nodes, a Simulator::Stop() function is called to end the simulation.

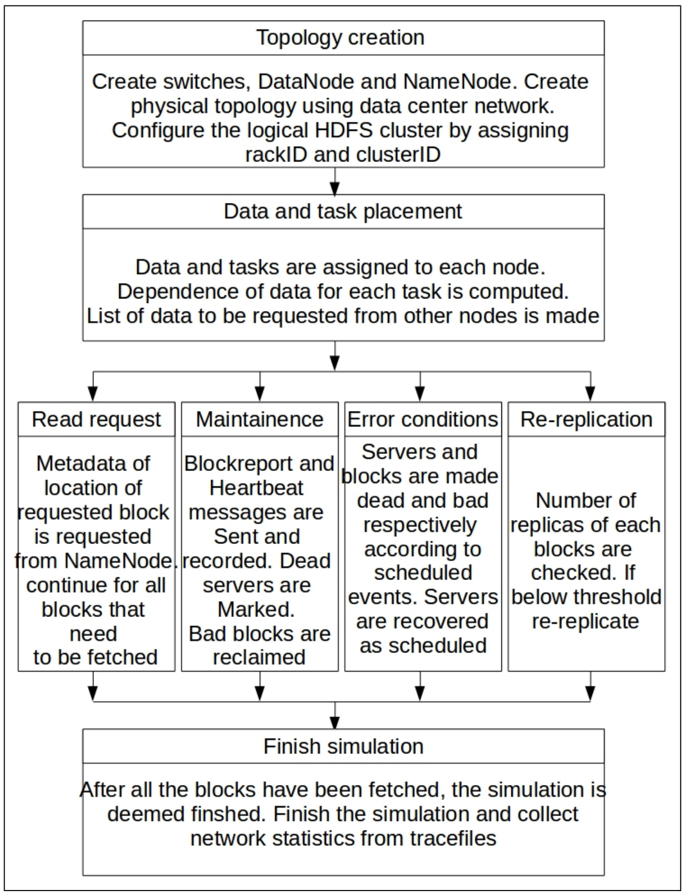

Topology creation
Create switches, DataNode and NameNode. Create physical topology using data center network. Configure the logical HDFS cluster by assigning rackID and clusterID
Data and task placement
Data and tasks are assigned to each node. Dependence of data for each task is computed. List of data to be requested from other nodes is made
Read request
Metadata of location of requested block is requested from NameNode. continue for all blocks that need to be fetched
Maintainence
Blockreport and Heartbeat messages are Sent and recorded. Dead servers are Marked. Bad blocks are reclaimed
Error conditions
Servers and blocks are made dead and bad respectively according to scheduled events. Servers are recovered as scheduled
Re-replication
Number of replicas of each blocks are checked. If below threshold re-replicate
Finish simulation
After all the blocks have been fetched, the simulation is deemed finshed. Finish the simulation and collect network statistics from tracefiles

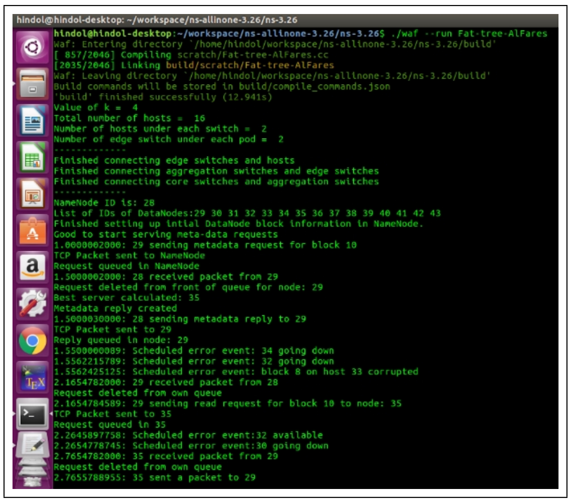
Figure 3.18: Flow of execution- HDFS simulation in NS3

Sample Run

A test run of the system is being shown in Figure 3.19 to 3.21. Every scheduled events have been appended with a cout statement such that the event is displayed at the console. The essential parameters are tabluted in Table 3.3.

Table 3.3: Essential Parameters

Parameter	Value(s)
Number of nodess	16
Number of tasks	8
Number of clusters	1
Number of datasets	12
Number of replicas per datasets	3
Task placement	Randomly set
Data placement	Randomly set
Re-replication placement	random
Error and recovery time	Randomly set
ID of block and server made erroneous	Randomly set
ID of block and server recovered	Randomly set
Heartbeat interval	HDFS default
Blockreport interval	HDFS default
Server invalidation interval	HDFS default
Bad block reclamation interval	Random
Re-replication interval	5 seconds

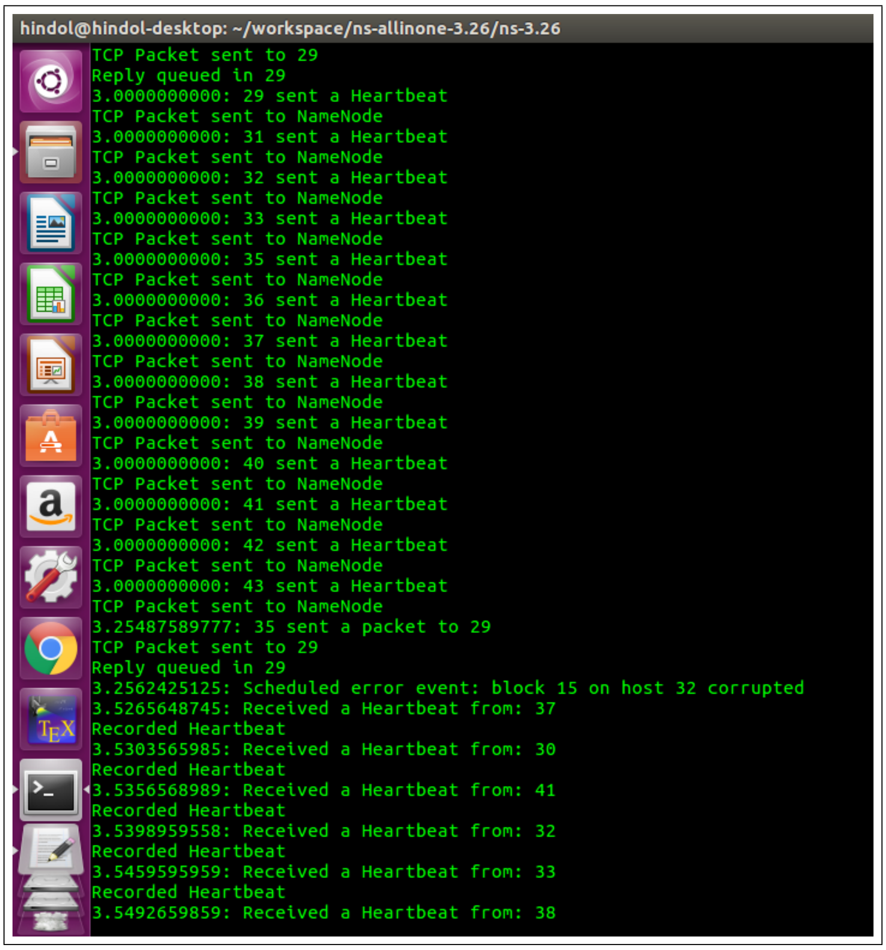


```
hindol@hindol-desktop: ~/workspace/ns-allinone-3.26/ns-3.26
hindol@hindol-desktop:~/workspace/ns-allinone-3.26/ns-3.26$ ./waf --run Fat-tree-AlFares
Waf: Entering directory `/home/hindol/workspace/ns-allinone-3.26/ns-3.26/build'
[ 857/2046] Compiling scratch/Fat-tree-AlFares.cc
[2035/2046] Linking build/scratch/Fat-tree-AlFares
Waf: Leaving directory `/home/hindol/workspace/ns-allinone-3.26/ns-3.26/build'
Build commands will be stored in build/compile_commands.json
'build' finished successfully (12.941s)
Value of k =  4
Total number of hosts =  16
Number of hosts under each switch =  2
Number of edge switch under each pod =  2
-------------
Finished connecting edge switches and hosts
Finished connecting aggregation switches and edge switches
Finished connecting core switches and aggregation switches
-------------
NameNode ID is: 28
List of IDs of DataNodes:29 30 31 32 33 34 35 36 37 38 39 40 41 42 43
Finished setting up intial DataNode block information in NameNode.
Good to start serving meta-data requests
1.0000002000: 29 sending metadata request for block 10
TCP Packet sent to NameNode
Request queued in NameNode
1.5000002000: 28 received packet from 29
Request deleted from front of queue for node: 29
Best server calculated: 35
Metadata reply created
1.5000030000: 28 sending metadata reply to 29
TCP Packet sent to 29
Reply queued in node: 29
1.5500000089: Scheduled error event: 34 going down
1.5562215789: Scheduled error event: 32 going down
1.5562425125: Scheduled error event: block 8 on host 33 corrupted
2.1654782000: 29 received packet from 28
Request deleted from own queue
2.1654784589: 29 sending read request for block 10 to node: 35
TCP Packet sent to 35
Request queued in 35
2.2645897758: Scheduled error event:32 available
2.2654778745: Scheduled error event:30 going down
2.7654782000: 35 received packet from 29
Request deleted from own queue
2.7655788955: 35 sent a packet to 29
```

Figure 3.19: Screenshot of sample run of NS3-HDFS extension- 1

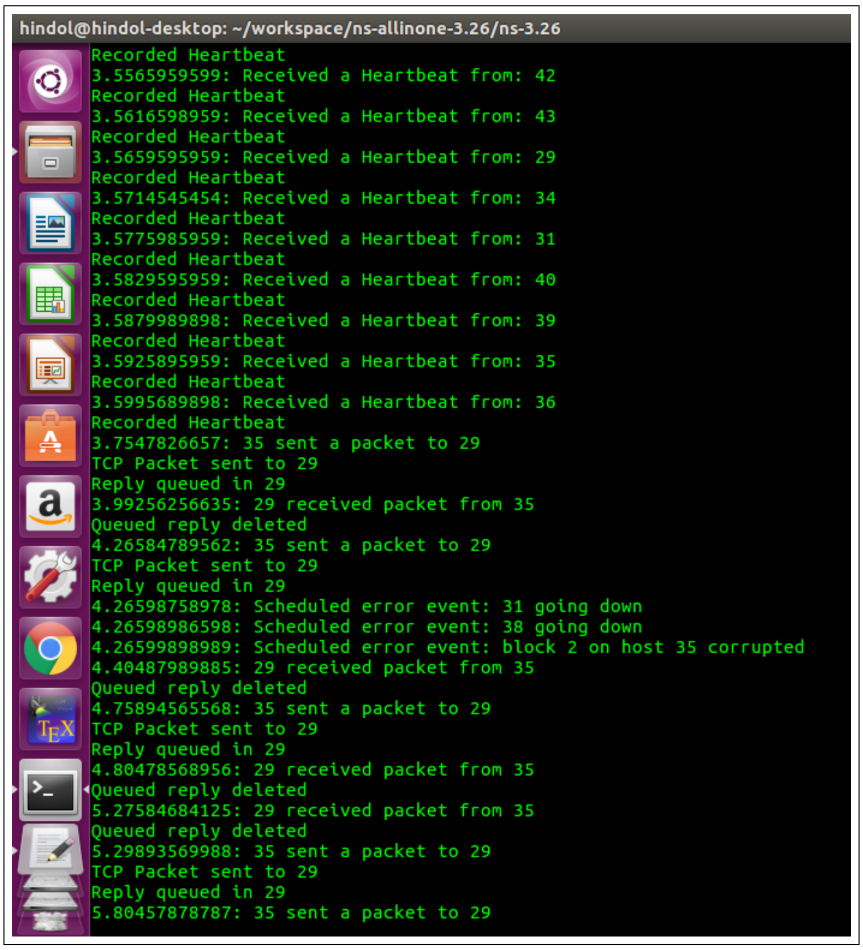
3.4 Concept Mapping to Subsequent Chapters

The subsequent chapters discuss broader novel research performed concerning data placement in distributed systems. In chapter 4, a



```
hindol@hindol-desktop: ~/workspace/ns-allinone-3.26/ns-3.26
TCP Packet sent to 29
Reply queued in 29
3.0000000000: 29 sent a Heartbeat
TCP Packet sent to NameNode
3.0000000000: 31 sent a Heartbeat
TCP Packet sent to NameNode
3.0000000000: 32 sent a Heartbeat
TCP Packet sent to NameNode
3.0000000000: 33 sent a Heartbeat
TCP Packet sent to NameNode
3.0000000000: 35 sent a Heartbeat
TCP Packet sent to NameNode
3.0000000000: 36 sent a Heartbeat
TCP Packet sent to NameNode
3.0000000000: 37 sent a Heartbeat
TCP Packet sent to NameNode
3.0000000000: 38 sent a Heartbeat
TCP Packet sent to NameNode
3.0000000000: 39 sent a Heartbeat
TCP Packet sent to NameNode
3.0000000000: 40 sent a Heartbeat
TCP Packet sent to NameNode
3.0000000000: 41 sent a Heartbeat
TCP Packet sent to NameNode
3.0000000000: 42 sent a Heartbeat
TCP Packet sent to NameNode
3.0000000000: 43 sent a Heartbeat
TCP Packet sent to NameNode
3.25487589777: 35 sent a packet to 29
TCP Packet sent to 29
Reply queued in 29
3.2562425125: Scheduled error event: block 15 on host 32 corrupted
3.5265648745: Received a Heartbeat from: 37
Recorded Heartbeat
3.5303565985: Received a Heartbeat from: 30
Recorded Heartbeat
3.5356568989: Received a Heartbeat from: 41
Recorded Heartbeat
3.5398959558: Received a Heartbeat from: 32
Recorded Heartbeat
3.5459595959: Received a Heartbeat from: 33
Recorded Heartbeat
3.5492659859: Received a Heartbeat from: 38
```

Figure 3.20: Screenshot of sample run of NS3-HDFS extension- 2



```
hindol@hindol-desktop: ~/workspace/ns-allinone-3.26/ns-3.26
Recorded Heartbeat
3.5565959599: Received a Heartbeat from: 42
Recorded Heartbeat
3.5616598959: Received a Heartbeat from: 43
Recorded Heartbeat
3.5659595959: Received a Heartbeat from: 29
Recorded Heartbeat
3.5714545454: Received a Heartbeat from: 34
Recorded Heartbeat
3.5775985959: Received a Heartbeat from: 31
Recorded Heartbeat
3.5829595959: Received a Heartbeat from: 40
Recorded Heartbeat
3.5879989898: Received a Heartbeat from: 39
Recorded Heartbeat
3.5925895959: Received a Heartbeat from: 35
Recorded Heartbeat
3.5995689898: Received a Heartbeat from: 36
Recorded Heartbeat
3.7547826657: 35 sent a packet to 29
TCP Packet sent to 29
Reply queued in 29
3.99256256635: 29 received packet from 35
Queued reply deleted
4.26584789562: 35 sent a packet to 29
TCP Packet sent to 29
Reply queued in 29
4.26598758978: Scheduled error event: 31 going down
4.26598986598: Scheduled error event: 38 going down
4.26599898989: Scheduled error event: block 2 on host 35 corrupted
4.40487989885: 29 received packet from 35
Queued reply deleted
4.75894565568: 35 sent a packet to 29
TCP Packet sent to 29
Reply queued in 29
4.80478568956: 29 received packet from 35
Queued reply deleted
5.27584684125: 29 received packet from 35
Queued reply deleted
5.29893569988: 35 sent a packet to 29
TCP Packet sent to 29
Reply queued in 29
5.80457878787: 35 sent a packet to 29
```

Figure 3.21: Screenshot of sample run of NS3-HDFS extension- 3

scenario is considered where interdependency data between the files are not available or could not be computed. In such a case, the dependency could theoretically be computed through similarity in the content of the files. The feasibility of this idea has been explored in chapter 4. In chapter 5, the data interdependency based prefetching technique has been considered. The dependency matrix can be interpreted as an adjacency matrix of a dependency graph. This graph can be used to analyze the file usage pattern. Since a graph is being used, the opportunity for social network analysis opens up. In this chapter, social network analysis is applied to solve a particularly intractable problem in dependency graph analysis for file usage pattern analysis. Data placement using error correction coding is explored in chapter 6, where an optimized data placement is achieved using a metaheuristic technique to minimize storage and network usage costs while maintaining a high data recovery probability.

3.5 Conclusions

In his chapter the important concepts required for understanding the works described in future have been discussed, Three novel research experiments have also been discussed[140, 139, 141], to further illustrate the concepts.

Chapter 4

Examining Data Dependency Relations Between Text Files from Topic Analysis

4.1 Introduction

In chapter 3, data and replica placement policy based on the concept of interdependency between data units has been introduced. The intuitive concept of interdependency has served the data placement requirements with considerable success as is evident from the works in literature.

The problem is that data interdependency information may not always be available and may not be derivable from other sources of information as well. Without the interdependency data available, the data and replica placement works proposed in the literature will not work. Hence, a way to bypass this limitation needs to be designed.

In the absence of interdependency or prior use data, the data placement algorithm only has the contents of the data at its disposal. It is necessary to find if the contents of the data could be used as a surrogate for interdependency values.

In this work, text based data is considered. The files being text files makes it possible to utilize topic modeling on them through the use of tools such as Latent Dirichlet Allocation (LDA). As discussed in [83], files of similar types generally exhibit higher interdependency between themselves. In many works in the literature concerning Natural Language Processing (NLP) and Information Retrieval (IR), it has been shown that similarity in topic between two text files is a good estimate to find the similarity between the files. Hence, it stands to reason that text files with similar topics could be considered as highly interdependent one another. Hence, by analyzing the content of the files (topics distributions), a good estimate of interdependency between a pair of files can be obtained.

There is good interoperability between the tools used in the NLP domain to those used in the bioinformatics domain (like genetics). Hence, the method of finding the similarity between files through LDA acting as a surrogate to interdependency between a pair of files can be extended to the realm of bioinformatics as well.

In this work, LDA has been used to find similarities between pairs of files. The topic distribution obtained through LDA is compared for similarity by the use of statistical distances. The similarity obtained is used to classify the files thereby establishing the use of LDA based similarity as a surrogate for interdependency values.

4.1.1 Formal Problem Statement

The problem is formally described in this sub section. For data interdependency, we use the the equation as defined in equation 3.1. However, the data on $ds_n.Task_n$ (data required by task n) and $ds_i, Task_i$ (data required by task i) are not available. Hence $Inter-dependency(File_n, File_i)$ needs to be defined as follows:

$$\begin{split} Inter-dependency(File_n, File_i) = |1- \\ Dist(Distr(File_n), Distr(File_i))| \end{split} \tag{4.1}$$

Where $Distr(File_n)$ is the topic distribution of text in file $File_n$ and $Distr(File_i)$ is the topic distribution of text in file $File_i$. $Dist(Distr(File_n), Distr(F$ computed the statistical distance between the topic distribution of the two files, in effect finding how different the two files are in terms of topics. The statistical distance can range between 0 and 1. Thus, $1 - Dist(Distr(File_n), Distr(File_i))$ gives the similarity between the distributions and hence similarity between the files. This similarity between the file pair is the interdependency between the file pair.

With interdependency calculated through equation 4.1, similar or interdependent files can be classified. This work [143] aims to figure out if such a grouping is possible with interdependency calculated using equation 4.1. If so, then file content's topic distribution could be used to determine file interdependency, when relevant data are not available.

4.1.2 Chapter Organization

This chapter is organized into five sections, including this introductory section. The necessary background concepts have been discussed in brief in Section two. The justification for the use of LDA instead of other state of the art language modeling approaches has also been discussed in Section two. The solution is discussed in Section three, with an empirical validation of the proposed solution discussed in Section four. This chapter is concluded in Section five.

4.2 Background Concepts

This work aims to solve the cold start problem in the determination of file interdependency. While, we have discussed file interdependency in details in previous chapters. In this section, other important concepts that will be required in formulation of the solution will be discussed.

4.2.1 Data Dependency through Topic Modeling

During the discussion of data dependency in [86], authors have discussed the concept of interest locality. Interest locality presents the idea that data are accessed as a group; the membership in the group is decided by commonality in the topic of the data. The idea is very intuitive. Consider a workflow task launched by an nucelar physicist, a particle physicist and a molecular biologist. At a broader level, the two physics related tasks would be more topically common than the task of molecular biologist. At a granular level, the two physics related tasks would differ topically. It is obvious that the data requirement by tasks would follow the same pattern.

Text data and certain scientific data (like protein bank, gene bank data) can be analyzed through topic modeling techniques. These documents are created based on the concept of latent topics and these topics can be estimated by techniques such as Latent Dirichlet Allocation [144]. It is possible to compute similarity between these datasets by comparing their topic distribution. These concepts will be discussed in the following subsection.

Considerng the above discussions, it is the aim of this work to find out if topic modeling could be used as a reliable alternative to data interdependency metric. This becomes important in cases where file usage data is not available and hence traditional interdependency calculation is not possible.

Latent Dirichlet Allocation based Topic Modeling

As discussed in the last section, text documents are considered to be made up of topics. However, these topics are not apparent on the document. Hence, the distribution of topics on the document has to be estimated through statistical analysis of the document itself. Latent Dirichlet Allocation is such a technique for statistical estimation of latent topics in a document. Before understanding the estimation technique, it is necessary to understand the document generation process first.

Document Generative Model

The document generative model considers that there is a distribution of topic associated with the document $\hat{\theta_{kj}}$ and there are words which are associated with the topics $\hat{\phi_{wk}}$. Hence, a document is formed by writing the words in them such that both the distributions are preserved. The generative model is shown in the form of probability graph diagram below:

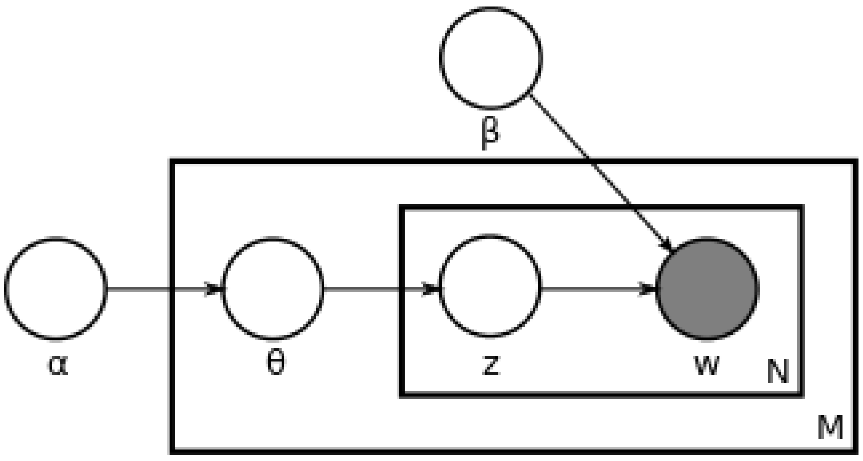


Figure 4.1: Document generative model of LDA

In the diagram, α and β are hyperparameters used to adjust the Dirichlet priors.

Estimating the distributions

The document generative model is a theoretical model which gives an intuition on what the topic distribution entails. In reality. documents are available and the topic distributions are to be estimated. A correct estimation, in theory, would be able to generate the document (in topical sense) through the document generation model discussed above.

Essentially the topic distribution estimation process is like reversing the document generative process. One way to achieve this is through Gibb's sampling as would be briefly described next.

Gibb's Sampling

In the original work of Blei et al. [144], the generative model in Section 3.1 of the paper has been described by the following equation:

$$p(w,t) = \int p(\theta)(\prod_{n=1}^{N} p(z_n|\theta)p(w_n|z_n))\, d\theta \tag{4.2}$$

Equation (4.2) represents the probability of a sequence of words and topics that would form the document in question. In the R.H.S of the equation, we have three probability terms. Let us focus on the innermost probabilities product: $p(z_n|\theta)p(w_n|z_n)$. We can note that it is formed of two independent probabilities: $p(z_n|\theta)$, which represents the probability that a topic z_n is assigned to document given distribution of topics in the document; while $p(w_n|z_n)$ represents the probability that the word w_n is assigned to topic z_n.

The interesting point to note here is that $p(w_n|z_n)$ states that probability of w_n is conditionally dependent on z_n only. From section 3 of [144], we know that $z_n \sim Multinomial(\theta)$ and $\theta \sim Dirichlet(\alpha)$, where α is a hyperparameter. Hence, we can see that probability of w_n is conditionally independent on the document or any sub units of the document and rather is computed for the entire corpus.

We end our discussion on the analysis of the document generative model noting the following important theorem.

Theorem 1.3.1: Word assignment to a topic is independent of any particular document and rather computed from the whole corpus.

The above Theorem can be proved as follows:

*Proof.*At the end of Gibb's sampling estimation iterations:

$$p(z_{ij} = k|z^{!ij}, x, a, b) = \hat{\phi_{wk}}\dot{\theta}\hat{kj},$$

$$\text{where, } \hat{\theta_{kj}} = \frac{N_{kj} + \alpha}{N_j + K\alpha}, \hat{\phi_{wk}} = \frac{N_{wk} + \beta}{N_k + W\beta}$$

$$\text{removing constants-}K,\ W,\ \beta,\alpha \implies \hat{\theta_{kj}} = \frac{N_{kj}}{N_j} \quad and \quad \hat{\phi_{wk}} = \frac{N_{wk}}{N_k} \tag{4.3}$$

Since $\hat{\theta_{kj}}$ and $\hat{\phi_{wk}}$ are independent, estimation of topic k being assigned to word w- $\hat{\phi_{wk}}$, is independent of j- in this case representing the document (s- in case of sentences). Hence prooving equation 4.6.

Gibb's sampling is the process of posterior estimation of topic distribution; i.e. estimating the topic probabilities given a document.We now move on to how the topic distribution for a document is determined through Gibb's sampling. We are going to closely follow the work of Porteous et al. [145].

The fundamental equation for Gibb's sampling based posterior estimation for a document $j \in D$, where D is a collection of documents making the corpus, is as follows:

$$p(z_{ij} = k|z^{!ij}, x, a, b) = \frac{1}{Z} a_{kj} b_{wk}$$

$$\text{where} \quad a_{kj} = N_{kj}^{!ij} + \alpha, \quad b_{wk} = \frac{N_{wk}^{!ij} + \beta}{N_k^{!ij} + W\beta} \quad \text{and} \quad Z = \sum_k a_{kj} b_{wk} \tag{4.4}$$

Gibb's sampling consists of many iterations, where variables a and b are adjusted to find a suitable value of $p(z_{ij} = k|z^{!ij}, x, a, b)$- the conditional probability that topic z is associated to word i of document j. Equation (4.4) defines how this conditional probability value is calculated for a document. Variable a represents the computation of assignment of topic k to document j and in equation (4.5), b represents the computation of assignment of topic k to word w. The third equation (4.4) is about the computation of normalization constant.

Why LDA

Since, LDA was introduced, there has been a lot of development in the realm of text modeling techniques. Modern word embedding methods such as Paragraph Vectors (PV) [146] and Sentence Bidirectional Encoder Representation from Transformers (SBERT) [147] have been shown to deliver better results than LDA [144]. So the natural question is why LDA is being used in this work.

Topic modeling based file interdependency measurement is based on the concept of measuring similarity between documents. It has been shown in [148], that measuring senquence alignment between similar sentences of the document pair gives a better result in terms of similarity measurement. It has been shown in the same work that similarity between document pairs measured through such method with LDA performs better than similarity measured through the traditional manner with the use of advanced methods like PV.

In this sense, it would appear reasonable to adapt state of the art techniques such as PV and SBERT to measure similarity through sequence alignment of similar sentences and expect better results. However, our emperical results suggests otherwise. The reason behind this counter intuitive empirical observation is explained by interpreting the mathematics behind each method's work principle.

The mathematical reasons are explained first, followed by the empirical observations in its support.

For a sub unit (sentence in this case) $s \in j \in D$, where j is any document in the corpus, equation 4.4 is modified as follows:

$$p^{'}(z_{is} = k|z^{!is}, x, a, b) = \frac{1}{Z^{'}} a^{'}_{ks} b^{'}_{wk}$$
$$\text{where} \quad a^{'}_{ks} = N^{!is}_{ks} + \alpha, \quad b^{'}_{wk} = \frac{N^{!is}_{wk} + \beta}{N^{!is}_{k} + W\beta} \quad \text{and} \quad Z^{'} = \sum_{k} a^{'}_{kj} b^{'}_{wk} \tag{4.5}$$

Considering Theorem 4.2.1, assignment of topic k to word w is not dependent on whether we are computing for a document or a sub unit of a document. Hence, we can state the following:

$$b_{wk} = b^{'}_{wk} \tag{4.6}$$

Thus, we can write the following equation:

$$Z^{'} \frac{p^{'}(z_{is} = k|z^{!is}, x, a, b)}{a^{'}_{ks}} = Z \frac{p(z_{ij} = k|z^{!ij}, x, a, b)}{a_{kj}} \tag{4.7}$$

Hence, a_{kj} can be derived in terms of a_{ks} as follows:

$$a_{kj} = \frac{Za^{'}_{ks}p(z_{ij} = k|z^{!ij}, x, a, b)}{Z'p'(z_{is} = k|z^{!is}, x, a, b)} \tag{4.8}$$

From equation 4.8 we can represent the computation of assignment of topic k to any document j in terms of computation of assignment of topic k to any sentence s. Alternatively, we can derive a_{ks} in terms of a_{kj} as well.

With LDA, we can represent any document in the corpus to any sub unit of the same or any other document in the corpus that establishes the relationship between a document and the sub units. This relationship is established in terms of topic distribution which semantically represents the document or the sub units of the document. Since LDA can represent a document in terms of its sub units, similarity measurement such as SIMDOC is possible; where sentences of the document can be used to represent the respective documents.

4.2.2 Statistical Distances and Divergences

Latent Dirichlet Allocation is essentially a probability distribution of topics in a document corpus. In order to measure similarity of two document, we need to find how the two probability distributions are similar to each other. The similarity is indirectly measured from statistical divergence between the distributions. Statistical diveregence is a measure of how two probability distributions are different from one another. A low value of statistical divergence means the probability distrbutions are similar, which in topic modeling context means that the two documents are similar in topic. A related concept of statistical distance is used when symmetric measure is needed; i.e., in statistical distance difference between distributions q and p is the same as difference between distributions p and q. We will use the Hellinger distance measure in this work as this has been shown to be appropriate measure for topic modeling purposes in [149].

4.3 Proposed Solution

The solution to the problem discussed in section 4.1 is discussed in Algorithm 1. The Algorithm pseudocode is followed by the explanation of the algorithm.

Algorithm 1 DependencyMatrixPrediction

```
 1: procedure CalculateBEA
 2:     $numTopics \leftarrow Number\ of\ topics$
 3:     $trainSet \leftarrow Collection\ of\ text\ files\ for\ training$
 4:     $testSet \leftarrow Collection\ of\ text\ files\ for\ prediction$
 5:     $model \leftarrow TrainTopicModel(trainSet, numTopics)$
 6:     $numTopicDistr[.] \leftarrow PredictTopicDistribution(model, testSet)$
 7:     $stat\_dist_{ij}[.][.] \leftarrow CalculateStatDist(numTopicDistr[.])$
 8:     $RearrangedMatrix \leftarrow BEA(stat\_dist_{ij}[.][.])$
 9: procedure TrainTopicModel(trainSet,numTopics)
10:     $bagOfWords \leftarrow Extract\ each\ word\ from\ trainSet$
11:     $model \leftarrow LDA(bagOfWords, numTopics)$
12:     return model
13: procedure PredictTopicDistribution(model, testSet, numTopics)
14:     while $num(testSet) \neq 0$ do
15:         $numTopicDistr[.] \leftarrow Topic\ distribution\ for\ the\ test\ dataset$
16:         return numTopicDistr[.]
17: procedure CalculateStatDist(numTopicDistr[.])
18:     for All elements in numTopicDistr[.] do
19:         $firstDistr \leftarrow each\ element\ of\ numTopicDistr[.]$
20:         for All elements in numTopicDistr[.] do
21:             $secondDistr \leftarrow each\ element\ of\ numTopicDistr[]$
22:             $stat\_dist_{ij}[.][.] \leftarrow Hellinger(firstDistr, secondDistr)$
23:             $dependency_{ij} = 1 - stat\_dist_{ij}[.][.]$
        return $dependency_{ij}$
24: procedure Hellinger(firstDistr,secondDistr, numTopics)
25:     for every $i$ in numTopics do
26:         $temp \leftarrow temp + (\sqrt{firstDistr_i} - \sqrt{secondDistr_i})^2$
27:     $hellingerDist \leftarrow \frac{1}{\sqrt{2}} temp$
28:     $return\ hellingerDist$
```

4.3.1 Explanation of the Algorithmic Pseudocode

The algorithm consists of five functions which are described as follows:

- CalculateBEA: This is the main procedure from which other procedures are invoked as required. The algorithm's main purpose is to produce a clustered dependency matrix such that it could be used by data placement algorithms.

- TrainTopicModel: This is first procedure to be called by CalculateBEA. This procedure takes the number of topics and collection of training data as inputs and is used to train a topic model using LDA.

- •PredictTopicDistribution: This is second procedure to be invoked by CalculateBEA. This procedure takes the trained model, collection of new datasets, and the number of topics as input. The probability distribution of all the topics as per the input number of topics is calculated for each dataset.

- •CalculateStatDist: This is the procedure that is used to invoke a procedure Hellinger, which calculates the Hellinger distance between all pairs of datasets. This procedure accepts the result of the Hellinger distance and computes the dependency between the datasets.

- •Hellinger: This procedure computes the Hellinger distance between two topic distributions. In our algorithm CalculateStatDist procedure invokes this procedure with two topic distribution. The Hellinger distance is calculated and returned to the invoking procedure.

In calculating the BEA rearrangement, an interdependency matrix needs to be computed. Typically such a matrix would be calculated from the equation defined in equation 3.1, chapter 3.2. However, as discussed in section 4.1 of this chapter, use of such equation is not possible in this case. Instead, equation 4.1 defined in the introductory section of this chapter is used. For this purpose topic distribution of files is required, which requires training LDA model, which is accomplished by $TrainTopicModel$ function which returns the trained LDA model. Once the trained model becomes available, the topic distribution of the files in the test set is inferred through the $PredictTopicDistribution$ function. Once the topic distribution for each file is available, the statistical distance, determined by squared Hellinger distance is determined through $CalculateStatDist$, which in turn uses the $Hellinger$ function. Once the Hellinger distance is available, the interdependency between the files is found by subtracting the distance from the maximum possible distance. Once all interdependencies are available, the dependency matrix can be created followed by BEA based file placement.

Once the BEA rearrangement is available, the files can be grouped into separate clusters.

4.4 Empirical Observations

To test the validity of the solution proposed in the last section, a set of experiments have been designed. In this section, the description of the experiments and their results along with the discussion on the results are discussed.

For the experiment, the BBC news article dataset provided by [150] has been used. The experiments aim to determine the clustering ability of topic distribution based file interdependency measurement.

4.4.1 Setup

The text files in the BBC dataset consist of five top level categories: Business, Entertainment, Sports, Technology, and Politics. The Sports category is further subcategorized into Athletics, Cricket, Football, Rugby, and Tennis. These are considered the true clusters. Hence, a group of files relating to business has high interdependency between themselves, while a low interdependency between files of other groups.

The LDA topic distribution of each file is computed using MATLAB's^{TM} text analysis toolbox. On obtaining the topic distribution for each file, statistical distance is computed in Python for each pair of files. An interdependency matrix is created from the interdependence computed between all pairs of files. Finally, BEA is applied to the matrix to rearrange the files such that highly interdependent files are adjacent.

The essential setup parameters are given in Table 4.1.

The parameters which are not included in the table are deemed to take the default value. Classification and clustering of high level datasets, namely Business, Entertainment, Sports, Technology, and Politics, use 10, 20, and 30 datasets (files) from each high level category for testing and another 10, 20, and 30 datasets from each high-level category for training. The individual categorization of files is removed by mixing them. This is to ensure that the algorithm is only able to classify and cluster the files by the interdependency metric. After application of BEA on the interdependency matrix, ten elements(dataset ID) in every iteration are extracted from the resulting rearranged interdependency matrix called clustered matrix. We measure what percentage of datasets in the clusters are actually from the same high level categories. For 100% accuracy, BEA is expected to cluster all datasets from a high level category in their respective cluster. For example, if datasets 1 to 30 are from the business category, then for 100% accuracy, we will be able to extract three groups of ten datasets from the BEA rearranged clustered matrix, which belongs exclusively to the business category. We call this measure Correct Predictions. Further, a confusion matrix has been presented to understand the prediction errors made by the developed system.

For an experiment involving Sports category subcategorization, apart from the above measure, we also measure whether the subcategories have also been correctly clustered in addition to the high level categories.

4.4.2 Results and Discussions

The empirical observations are presented in terms of confusion matrices in Tables 4.2, 4.3. The prediction accuracy and cluster wise data layout is presented in Tables 4.4 and 4.5.

The interpretation of the results is as follows:

- For high level categorization, there is a cluster prediction accuracy of 72%, 93%, and 92%, for 10, 20, and 30 datasets per category respectively. The cluster prediction accuracy validates that our proposed algorithm could be used for grouping files based on their topic; thereby making it suitable for consideration as an alternative to the interdependency matrix.

- In the sports category, further fine grained subcategory classification has been performed. Cluster prediction at a granular level is at an average of 88.89%. Most of the category specific predictions are at 90% accuracy. Hence, the proposed algorithm is capable of producing correct grouping even at a granular level.

- With the rise in the number of datasets, the algorithm's prediction accuracy gets better.

- Misclusterings are at border level, i.e., whenever a wrong cluster is assigned, it is in the neighborhood of the correct cluster. Thus, slight fine tuning could be applied.
- Confusion matrices also corroborate the cluster prediction accuracy results by having very few group mispredictions- both at the high and granular level.

Table 4.1: Important setup parameters

Parameters	Values
Total number of datasets (per category)	training: each top level category 10, 20, 30 sports subcategory classification each subcategory: 10 testing: each toplevel category 10, 20, 30 sports subcategory classification each subcategory: 10
Number of Categories	top level categoriees 5 sports subcategory classification 9 4 for high level category 5 for Sports subcategory
Number of topics	50
Short words	$\leq$ 2 characters
Long words	$\geq$ 15 characters

Table 4.2: Confusion Matrix for 10 dataset sports subcategory classification

		Predicted								
		Athletics	Cricket	Football	Rugby	Tennis	Business	Entertainment	Politics	Tech
Actual	Athletics	9	1	0	0	0	0	0	0	0
	Cricket	0	9	1	0	0	0	0	0	0
	Football	0	0	9	1	0	0	0	0	0
	Rugby	0	0	0	9	1	0	0	0	0
	Tennis	0	0	0	0	9	1	0	0	0
	Business	0	0	0	0	0	9	1	0	0
	Entertainment	1	0	0	0	0	0	8	1	0
	Politics	0	0	0	0	0	0	0	9	1
	Tech	0	0	0	0	0	0	1	0	9

Table 4.3: Confusion matrix for 10, 20 and 30 dataset

10 dataset per category

		Predicted				
		Business	Entertainment	Politics	Sport	Tech
Actual	Business	7	0	0	1	2
	Entertainment	3	7	0	0	0
	Politics	0	2	7	1	0
	Sport	0	0	3	7	0
	Tech	0	1	0	1	8

20 dataset per category

		Predicted				
		Business	Entertainment	Politics	Sport	Tech
Actual	Business	19	1	0	0	0
	Entertainment	1	18	1	0	0
	Politics	0	0	19	1	0
	Sport	0	1	0	18	1
	Tech	0	0	0	1	19

30 dataset per category

		Predicted				
		Business	Entertainment	Politics	Sport	Tech
Actual	Business	25	0	0	1	4
	Entertainment	1	29	0	0	0
	Politics	0	1	29	0	0
	Sport	1	0	1	28	0
	Tech	3	0	0	1	26

Table 4.4: Clusterwise Dataset layout and Prediction Accuracy

10 dataset sports subcategory classification

Cluster ID	Dataset ID										Correct Predictions
Cluster 1	88	90	66	87	86	85	84	83	82	81	90.00%
Cluster 2	80	79	78	77	76	89	75	74	73	72	90.00%
Cluster 3	71	70	69	68	67	1	65	64	63	62	80.00%
Cluster 4	61	60	59	58	57	56	55	54	53	52	90.00%
Cluster 5	51	50	49	48	47	46	45	44	43	42	90.00%
Cluster 6	41	40	39	38	37	36	35	34	33	32	90.00%
Cluster 7	31	30	29	28	27	26	25	24	23	22	90.00%
Cluster 8	21	20	19	18	17	16	15	14	13	12	90.00%
Cluster 9	11	10	9	8	7	6	5	4	3	2	90.00%
										Overall	88.89%

10 dataset

Cluster ID	Dataset ID										Correct Predictions
Cluster 1	47	20	48	46	45	44	43	42	41	40	80.00%
Cluster 2	37	36	34	38	35	33	31	30	29	28	70.00%
Cluster 3	27	26	25	24	23	22	21	19	39	15	70.00%
Cluster 4	18	17	16	3	14	13	11	10	9	12	70.00%
Cluster 5	50	8	49	32	7	6	5	4	1	2	70.00%
										Overall	72.00%

20 dataset

Cluster ID	Dataset ID										Correct Predictions
Cluster 1	99	100	96	98	97	71	95	94	93	92	90%
Cluster 2	91	90	89	88	87	86	85	84	83	82	100%
Cluster 3	81	80	79	78	77	76	75	74	73	72	90.00%
Cluster 4	40	70	69	68	67	66	65	64	63	62	90%
Cluster 5	61	60	59	58	57	56	55	54	53	52	90%
Cluster 6	51	50	49	48	47	46	45	44	43	42	1000%
Cluster 7	41	18	39	38	37	36	35	34	33	32	80%
Cluster 8	31	30	29	28	27	26	25	24	23	22	100%
Cluster 9	21	20	19	1	17	16	15	14	13	12	90%
Cluster 10	11	10	9	8	7	6	5	4	3	2	100%
										Overall	93.00%

Table 4.5: Clusterwise Dataset layout and Prediction Accuracy (Contd...)
30 dataset

Cluster ID	Dataset ID										Correct Predictions
Cluster 1	148	150	149	147	8	9	144	143	142	141	80.00%
Cluster 2	139	138	11	146	136	136	135	134	133	132	90.00%
Cluster 3	131	130	129	128	127	126	10	124	123	122	90.00%
Cluster 4	121	120	119	118	117	116	115	114	113	112	90.00%
Cluster 5	111	110	109	108	107	106	105	104	103	102	100.00%
Cluster 6	101	100	99	98	97	96	95	1	93	92	90.00%
Cluster 7	91	90	89	88	87	86	85	84	83	82	90.00%
Cluster 8	81	80	79	78	77	76	75	74	73	72	100.00%
Cluster 9	71	70	69	68	67	66	65	64	63	62	100.00%
Cluster 10	61	60	59	58	57	56	55	54	53	52	90.00%
Cluster 11	51	47	49	48	47	46	44	44	43	42	100.00%
Cluster 12	41	40	39	38	37	36	35	34	33	32	100.00%
Cluster 13	31	30	29	28	27	26	25	24	23	22	90.00%
Cluster 14	21	20	19	18	17	16	15	14	13	12	100.00%
Cluster 15	137	94	145	146	7	6	5	4	3	2	70.00%
										Overall	92.00%

4.5 Conclusions

Interdependency between datasets is the key to good file placement in distributed systems. However, when file usage information is not available, an alternative method is required. In this work, such an alternative method using topic modeling has been proposed for text files. The algorithm has been empirically validated to produce good group predictions based on topic analysis.

It is expected that the proposed algorithm could be used in other data types which have analytical inter operability with text datasets like bioinformatics datasets. It would be interesting to adapt the algorithm for the specific needs of such datasets. This work has limitations on its applicability in terms of data types. A workaround for multimedia data types should be sought.

Chapter 5

A Social Network Analytics Based Dependent File Pre-fetching in Distributed System

5.1 Introduction

In the earlier chapters, the necessity for the localization of data in a distributed computing environment has been established. In Chapter 3, we discussed the concept of inter dependency of data and how this concept could be used to formulate an effective data localization scheme. However, we have discussed the concept of data localization and placement in terms of real time data placement only.

While data placement minimizes data migration at task execution time and improves system performance, substantial time would have to be spent to localize the data before any task execution could begin. This presents the scope for further improvement by attempting to minimize the wait time that a task has to endure before beginning its execution.

In Chapter 2, prefetching techniques have been discussed which are aimed at retaining a subset of the data resident at the local site during a previous execution. With an accurate prediction of file use in the future; the expectation is that the necessary files would be retained at the local site. These will be the files that will be required during the execution of the task(s) in the future. Thus, with data already present at the site, the tasks immediately begin their execution without waiting for initial data placement, and considering accurate prediction, no further data migration would be necessary during task execution.

In the literature, different solutions of this problem have been discussed. Different methods including data mining, graph theory, etc have been utilized to propose highly accurate prediction methods of future file usage and file prefetching based on it.

One particular method of choice is through the analysis of a data dependency graph. Chapter 3 discusses how data inter dependency can be calculated between pairs of data units (files). Such computed metric between every pair of files may be represented in the form of a graph; such that the vertices of the graph represent the files, while the edges between each pair of vertices represent the dependency between the files. Such a graph may be analyzed to predict the inter dependency between files.

While there are numerous other methods proposed in literature apart from graph theory, the graph theoretic approach offers numerous analytical tools which are worth exploring. Social networks are the interpretation of a graph as representing the social bond between individuals in a society. Different applications of social network analytics have been discussed in the literature survey (Chapter 2). Here an important concept of social network analysis and indeed graph theory has been discussed in connection with the improvement of data inter dependency based data prefetching in distributed systems.

5.1.1 Chapter Organization

The contents of this chapter are organized into seven sections including this introductory section. The problem that is being addressed through the work described in this chapter is discussed in Section 2. The use of social network analysis in addressing the problem and the associated hypothesis is discussed in Section 3. The hypothesis is empirically verified; the empirical observations and the associated details on the setup of the experiments are discussed in Section 4. On confirmation of the hypothesis, a concrete algorithm is proposed as a solution to the problem. The details of this algorithmic solution are discussed in Section 5. The algorithm is validated empirically with another set of experiments. This experimental setup and the empirical observations are discussed in Section 6. The chapter is concluded in Section 7.

5.2 A Discussion on the Problem

Before introducing the Social Network Analysis and its application in the solution of our problem, a discussion on the problem being solved is presented in this section.

Before discussing the problem, it is vital to understand the three steps generally followed in a prefetching algorithm:

- Dependency graph formation: The interdependency between each pair of files is computed. These are arranged in terms of a dependency matrix, where each cell in the matrix represents the dependency between the corresponding pair of files. A dependency graph can be created by interpreting the dependency matrix as an adjacency matrix. A pair of files are determined interdependent if they are requested within a fixed time interval (determined by a threshold). The number of such requests is the dependency between the two files.

- Partitioning the dependency graph: The dependency graph may be a singly connected component or multiple connected components. Some of the edges are representative of significant dependencies between two files, which are representative of a file usage pattern. While others are insignificant and incidental and do not represent a file usage pattern and are likely a result of noise. Since these noisy dependencies distort the actual file usage pattern, they must be omitted to reveal the actual file usage pattern and hence an accurate prediction. A threshold is determined and edges having weight above this threshold value are deemed significant dependencies; an edge having a lower weight is considered as an incidental dependency and by the above logic is discarded from the graph. The deletion of these edges may result in partitioning the graph into one or more disconnected components.

- Choosing the correct set of files: The set of vertices in each of the components obtained from the above process represents a set of highly interdependent files. The important criterion is to determine which of these set(s) is/are to be prefetched. Generally, a set of files that have been requested most recently or often among the competing sets, is the one that is prefetched.

The performance of a dependency graph based solution is dependent on the choice of the value of different threshold values. The most important threshold value to be determined is the one that is used to partition the graph into disconnected components, among which the set of nodes (representing the files) of one of the subgraphs is selected for prefetching.

The value of graph partitioning threshold is usually passed as a hyper parameter to the prefetching algorithm. Hence, the performance of the prefetching algorithm depends on the computation of a good threshold value. A lower threshold will include too many files, many of which may represent noises. A higher threshold would reject interdependencies among files that could potentially be used in the future. Indeed, an accurate future prediction is contingent upon the selection of an accurate threshold value, which can filter the appropriate file usage pattern.

The problem can be summarised as follows:

Definition 2. *Problem Statement: Find a threshold value that can partition the dependency graph such that each subgraph contains the set of highly dependent files. The vertices of these subgraphs each represent a fie usage pattern among which the most relevant one is chosen.*

The above problem can be defined mathematically as follows:

$$\begin{aligned} Find \quad threhold_v, \\ partitions \quad G(V,E)->G_1(V,E),...,G_n(V,E) \\ Weight_{Edge(E_{an},E_{am})} > Weight_{Edge(E_{an},E_{bm})} \end{aligned} \tag{5.1}$$

The above equation describes an optimization problem that finds an optimal threshold value. This threshold value is used for partitioning the dependency graph into disconnected components, such that the edge weight between two vertices in the same component is always greater than the edge weight between two vertices in separate components. This ensures that vertices in each of the components represent the set of files that are historically highly interdependent.

E_{an} and E_{am} are two edges n and m, each in the same community a; while, E_{an} and E_{bm} are two edges n and m, each in the different community a and b respectively.

Once the set of highly dependent files being partitioned, the only task left is to select one or more of these sets of files which are to be prefetched

5.2.1 Scope of Work

As discussed above, there are three steps involved in the prefetching process. In this work, the second step of determining the threshold to partition the graph and determine sets of highly interdependent files is being considered. This work explores the possibility of using social network based analytics such that graph partition as per equation 5.1 is satisfied [151].

5.3 Social Network Analytics and its Applications in Current Problem

This chapter is aimed at the application of a social network concept in the domain of data prefetching in distributed systems. Social networks, a relevant concept, and its application in the data prefertching problem are described in this section.

A social network is a representation of social structure wherein interactions between social entities (individuals, organizations, and as we will describe in this case; data) are captured in the form of graphs having a certain structure. The vertices of the graph represent the social entities, while the edges define the interaction between the social entities.

A graph is considered a social network if it displays the characteristics of a small world network. Small world networks are graphs which tend to contain cliques or near cliques. Such structures are prone to community formation, i.e., groups of vertices having high inter connections amongst themselves, but few connections to other vertices in the graph. Such communities of vertices are linked through one or few edges; such edges are called local bridges or weak ties. The identification of local bridges is an important activity in social network analytics, not only because communities can be isolated by the identification and deletion of such edges but also because such edges have a special purpose of their own. Mathematically, a graph is determined to be a small world network, if its Omega value is as close to 0 as possible.

The weak ties are identified by applying the Girvan Newman algorithm [152]. This algorithm computes the betweenness centrality of all edges of the graph. The edges having a high betweenness centrality are deemed to be local bridges the removal of which leads to partitioning of the graph.

Betweenness centrality is the most important metric to be used in this work. Hence we summarize as follows:

- Edges having high betweenness centrality are the edges that connect the communities of a graph
- Deletion of such edges will lead to disconnected components.
- Each of the groups of vertices, obtained by deletion of high betweenness centrality edges, are very densely connected amongst each other.
- The betweenness centrality of edges within a community is less than that of local bridges.

The usefulness of social networks spans across different disciplines: From sociology, criminology, health care, linguistics, etc. The utility of social network analysis in computer science has been studied in the literature. Leaving aside the controversial analytics by the social network platforms, recommendation systems utilize social network analytics for more fine tuned recommendations.

One interesting work in this regard is [153]. This work studied the Amazon copurchase dataset. The objective of the paper was to determine whether the item on demand had any correlation with some of the graph metrics: one of them being betweenness centrality. It was found that there is a good correlation between betweenness centrality and the demand for an item.

There is a strong similarity between the Amazon copurchase dataset and the file dependency dataset. Both the cases are represented as a graph. Vertices, in the case of the Amazon dataset, represent the items on sale, while in the case of the file dependency dataset, vertices represent files. The edges in the Amazon dataset represent the copurchase between the items, i.e., how often two items are purchased together. Similarly, in the file dependency dataset, edges represent how often the two files are requested together by a task.

Owing to the similarity in the two datasets, it makes sense that like in the study with Amazon dataset, the file interdependency problem may exhibit the same pattern that higher the betweenness centrality of an edge, the higher would be its weight thereby higher the interdependency between the files.

Further, intuitively files in a scientific workflow tend to form communities; like physics related files, biology related files; the physics sub communities like particle physics, astrophysics, etc. Hence, the dependency graph of these files is also likely to exhibit the small world network structure described earlier, making it a good candidate for social network analysis.

The above intuition can be rewritten as the following hypothesis:

Hypothesis 1. *The edges having the highest betweenness centrality are the edges having the highest weights.*

Since, the problem stated above is concerned with segregating dependency (represented as edges) based on the weight of the edge, hence by the above hypothesis, high betweenness could be exploited to solve the stated problem.

The next section is aimed at empirically verifying the validity of the hypothesis. After verifying the hypothesis, an algorithmic solution to the problem will be proposed.

5.4 Empirical Observations on SNA Hypothesis

The solution proposed in the next section is contingent upon the validation of hypothesis proposed in the last section. A hypothesis in social networks analytics can be validated through empirical observations only. Hence, an experiment has been set up to study the validity of the stated hypothesis in the context of data interdependency.

Validation of the hypothesis would enable the use of the metric of betweenness centrality to select the group of files to be prefetched, thereby obtaining a solution to finding an appropriate threshold for graph partitioning and file selection.

5.4.1 Description of the Experiments

For this experiment, we consider two synthetic workflows: MONTAGE and CYBERSHAKE from the Pegasus synthetic workflow collection. This synthetic workflow collection has been chosen due to it being an established dataset of repute [154][1].

The dataset consists of a record of executable tasks and the files it requires during its execution. The record could be inverted to create a dataset consisting of a collection of files, each with its list of tasks that have requested the file for its execution.

The two specific workflow from the collection has been chosen due to their opposite characteristic. In MONTAGE, the number of files far outnumber the number of tasks. A lower task to file ratio makes it more likely that two or more files will share a common task. Hence, a dense and information rich dependency matrix is expected. In the case of CYBERSHAKE, the inverse is true. The number of tasks are almost same as the number of files. Thus, there is a considerable possibility that a pair of files are unlikely to be requested by a common task; such pairs of files are likely to exhibit no interdependency. Hence, due to such a low task to file ratio, a sparse, information poor dependency matrix could be expected. We wish to test the validity of our hypothesis on a condition that is likely to give a poor outcome so that the limitations could be tested as well.

Table 5.1: Node to Edge Ratio of Main Disconnected Component Graph

MONTAGE	CYBERSHAKE
0.0081	0.32932

The list of tasks per file dataset allows the computation of data interdependency between each pair of files. The mathematical formula for this computation is as follows:

$$Inter - dependency(F_a, F_b) = Number(S(F_a) \cap S(F_b)) \qquad (5.2)$$

Where, $S(F_a)$ and $S(F_b))$ are the the list of tasks that requires file F_a and F_b respectively.

[1]https://confluence.pegasus.isi.edu/display/pegasus/WorkflowGenerator –deprecated, this page links to a new version

F_a and F_b are any two files, whose interdependency is being calculated. Interdependency is the number of tasks that have requested both files. Hence, such files are common in both file lists or more appropriately sets.

For a collection of files F_1 to F_n, the dependency between all files can be arranged in the form of a matrix called dependency matrix. The matrix is represented as follows:

$$\begin{bmatrix} F_{11} & F_{12} & F_{13} & \dots & F_{1n} \\ F_{21} & F_{22} & F_{23} & \dots & F_{2n} \\ \dots & \dots & \dots & \dots & \dots \\ F_{d1} & F_{d2} & F_{d3} & \dots & F_{dn} \end{bmatrix} \tag{5.3}$$

$$Where, \quad F_{dn} = Number(S(F_d) \cap S(F_n))$$

The matrix in equation 5.3 can be used as an adjacency matrix to create a graph; called dependency graph, the vertices of the graph being the files, and the value in each cell of the matrix represents the weight of the edge between two files.

It is this graph that needs to be partitioned through deletion of edges whose weight is lower than a threshold.

The betweenness centrality of each edge of the graph can be calculated as well using the following formula:

$$g(v) = \sum_{s \neq vt} \frac{\sigma_{st}(v)}{\sigma_{st}} \tag{5.4}$$

In equation 5.4, we are finding the betweenness centrality of edge v, which is an edge connecting vertices representing files; F_s and F_t. $\sigma_{st}(v)$ is the total number of shortest path from node s to node t; σ_{st} is the number of shortest path between nodes s and t, through edge v.

Edges are ranked according to their betweenness centrality value. More than one edge can have the same betweenness centrality value. In such a case, the weight of the edge is used to rank the edges.

A separate ranking is performed based solely on the weight of the edges. A higher correlation between the ranks validates the stated hypothesis.

5.4.2 Results and Discussions

The experiments performed below measures the correlation between the weight and betweenness centrality of each edge. Based on their decreasing value of weight and betweenness centrality, the edges are ranked. Edges sharing the same betweenness centrality value are ranked based on their weight. The two rankings are measured for their Kendall Tau correlation

The Kendall Tau correlation measure for MONTAGE is presented in Figure 5.1, while the same measure for CYBERSHAKE is presented in Figure 5.2.

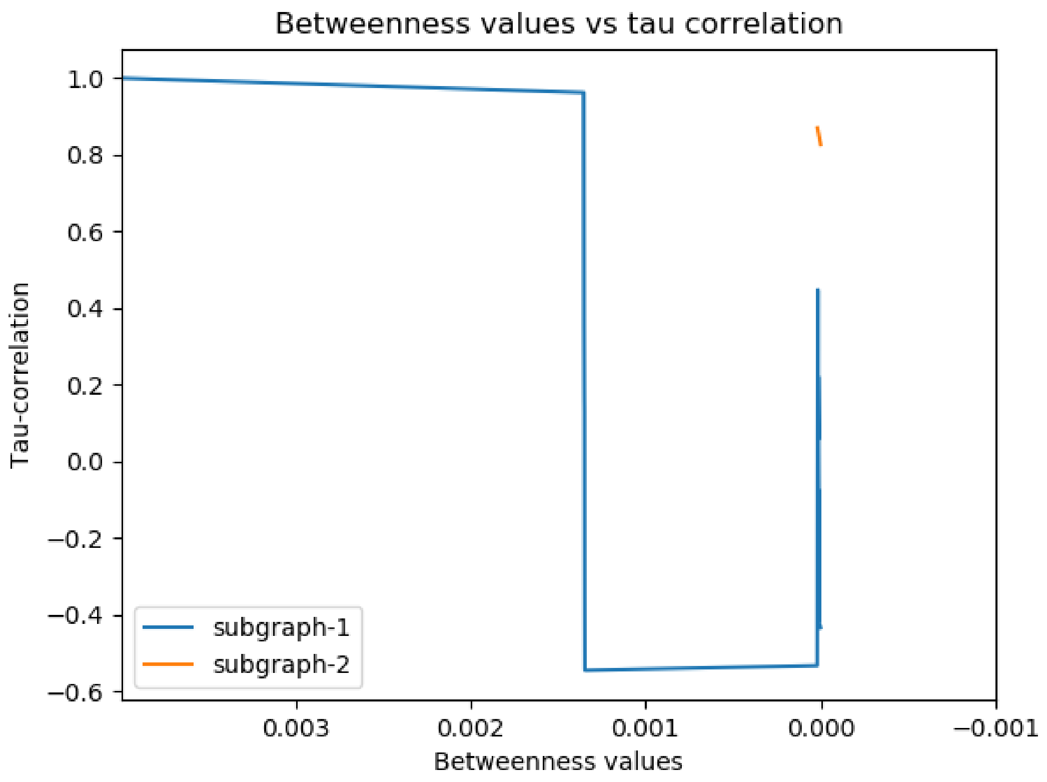


Figure 5.1: Tau Correlation for each incremental betweenness values in MONTAGE

The Kendall Tau correlation in the case of MONTAGE as presented in Figure 5.1 is more clear. For subgraph 1, there is a steady decline in Tau correlation from perfect correlation in the first (highest) betweenness centrality value, which abruptly drops to the negative correlation between 0.002 to 0.001. The negative correlation remains steady for some more betweenness centrality value, with a sharp rise to positive

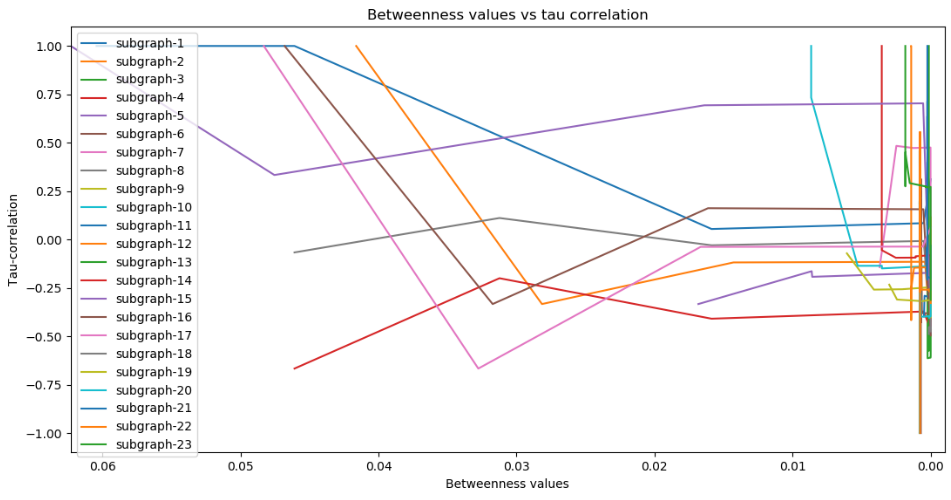


Figure 5.2: Tau Correlation for each incremental betweenness values in CYBERSHAKE

correlation towards the least betweenness centrality value (0). However, overall, the first few betweenness centrality values (the highest betweenness centrality value) have a very high correlation, validating hypothesis stated earlier. Subgraph 2 also confirms the same. In the case of CYBERSHAKE, the correlation between betweenness centrality value and edge weight is confirmed by some of the subgraphs, while other subgraphs show anomalous correlation. This anomalous results could be attributed to the sparse information scenario presented above. As will be clear in subsequent discussions, this sparsity would have a detrimental effect on the application of the solution.

While Figures 5.1, 5.2, validates the hypothesis, two more results are presented to gain further insights into the relaton between betweenness centrality and edge weight.

While, we have received an overall picture of the correlation from Figures 5.1, 5.2, the quartile deviation plots further illustrate the relationship between betweenness centrality and edge weight. Eight quartiles have been considered, each quartile representing 1/8th of the total value. From Figure 5.3 in CYBERSHAKE, the quartile behavior among the 23 subgraphs is quite erratic. However, a common trend of rising deviation with higher quartiles could be observed. The rising deviation can be observed in MONTAGE as well, though quite uniformly between the two subgraphs as suggested by Figure 5.4.

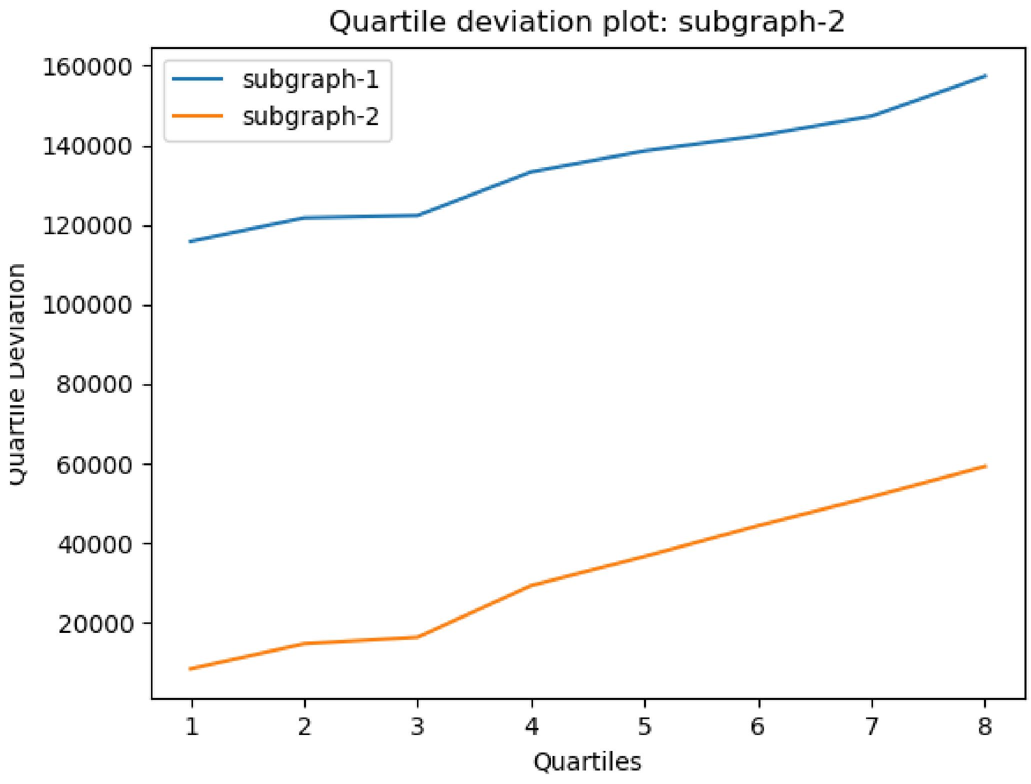


Figure 5.3: Quartile Deviation plot in case of MONTAGE

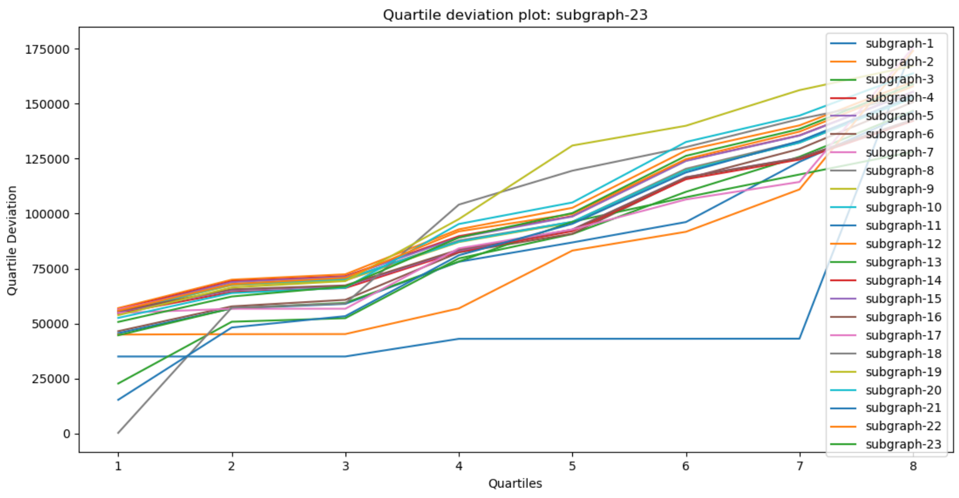


Figure 5.4: Quartile Quartile plot in case of CYBERSHAKE

In summary, the empirical observation suggests that in an information dense dataset, betweenness centrality could be used as an effective metric for the solution to the graph partitioning problem as will be discussed next, as per Hypothesis 1.

5.5 Proposed Solution

5.5.1 Solution Calculations, Intuitions and Algorithm

We have established the validity of the hypothesis from the empirical observations presented in the last section. We also obtained some insights into the relation between edge betweenness centrality value and edge weight in a file dependency graph. We are in a good position to exploit the insights gained to propose a solution to our problem.

Calculating the Threshold Value

According to the validated hypothesis 1, the highest betweenness value contains the highest weighted edges. Thus the threshold can be calculated by the equation below:

$$threhold_v = \frac{\sum_{e \in Betweenness_1} Weight(e)}{|Betweenness_1|} \tag{5.5}$$

Equation 5.5 finds threshold by finding the average weight of all the edges having the highest betweenness centrality value. It is this threshold value, $threhold_v$, which is used to delete edges and partition the graph such that the optimization problem described in equation 5.1 is satisfied.

The intuition behind the arithmetic mean based solution lies in the fact that the edges having the highest betweenness centrality value are the most significant edges (determined by weight). Ideally, all the significant edges would exhibit the highest betweenness centrality value. However, in reality, some insignificant edges also exhibit high betweenness centrality value and vice versa. Thus, the mean weight of the edges in the highest betwweenness centrality value that finds the weight at which significant edges can be seperated from the insignificant ones. The collection has an imbalance , where the significant weights outnumber the insignificant edges. Thus, arithmetic mean, having a bias towards the dominant class, is the most suitable mean to partition the collection into significant and insignificant edges. This value is considered closest to the ideal threshold weight. Thus, significant edges in other betweenness centrality values are included while discarding the insignificant edges with high betweenness value.

Graph Partitioning Algorithm

Having presented with the necessary background for the solution, we present our graph partitioning algorithm (Algorithm 2).

Algorithm 2 Calculating Threshold for Graph Partitioning

```
1: procedure CALCULATING_THESHOLD($G, G.edge\_weight$)
2:     $bc\_dict \leftarrow dictionary$                          ▷ Declaring a dictionary
3:     $sorted\_bc\_dict \leftarrow dictionary$                  ▷ Declaring a dictionary
4:     $edge\_list \leftarrow list$                              ▷ Declaring a list
5:     $Cumm\_Weight \leftarrow 0$                               ▷ Declaring a variable
6:     $count \leftarrow 0$                                      ▷ Declaring a counter variable
7:     while $\exists e \in G$ do
8:         $bc(e) \leftarrow Find\_BC(e)$
9:         if $bc(e) \in bc\_dict.key()$ then
10:            $bc\_dict.value(bc(e)) \leftarrow append(e)$
11:        if $bc(e) \notin bc\_dict.key()$ then
12:            $bc\_dict.key() \leftarrow bc(e)$
13:            $bc\_dict.values(bc(e)) \leftarrow append(e)$
14:    $sorted\_bc\_dict \leftarrow Sort(bc\_dict)$
15:    $edge\_list \leftarrow sorted\_bc\_dict[0]$
16:    while $\exists e \in edge\_list$ do
17:        $Cumm\_Weight \leftarrow Cumm\_Weight + edge\_weight(e)$
18:        $count \leftarrow count + 1$
19:    $threshold_v \leftarrow Cumm\_Weight/count$
20:    $return(threshold_v)$
```

Algorithm 2 receives the dependency graph G and the weight of the graph's edges $G.edge\_weight$ as its input. It creates a dictionary *bc_dict* with key as the betweenness centrality values and value as the list of edges corresponding to the key's betweenness centrality value. The betweenness centrality of each edge in the graph is computed by function $Find\_BC(e)$, which implements the betweenness centrality formula. The edge is listed at the appropriate list in the *bc_dict* dictionary, depending on the betweenness centrality value computed.

The *bc_dict* dictionary is sorted based on the key value, i.e., the betweenness centrality value. The dictionary is stored in *sorted_bc_dict* dictionary such that the key is sorted from highest to lowest betweenness centrality value. The list of edges of the first and therefore the highest betweenness centrality from *sorted_bc_dict* dictionary is selected. This list is assigned to *edge_list*.

Once the list of edges is available, their weights are summed up and the average of their weight is returned as the threshold value for graph partitioning: $threshold_v$. This returned threshold value is used as an input in Algorithm 3, used to partition the dependency graph is discussed next.

Algorithm 3 Algorithm for Graph Partitioning

```
1: procedure GRAPH_PARTITIONING($G, G.edge\_weight, threhold_v$)
2:     $selected\_files \leftarrow list$
3:     while $\exists e \in G$ do
4:         if $G.edge\_weight(e) < threhold_c$ then
5:             Delete_edge(e)
6:     if $G.multiple\_components == True$ then
7:         $selected\_G \leftarrow Select\_subgraph(G)$
8:     if $G.multiple\_components == False$ then
9:         $selected\_G \leftarrow G$
10:    for $vertices \in selected\_G$ do
11:        $selected\_files \leftarrow nodes$
```

The inputs to Algorithm 3 are the dependency graph G, weight of the graph's edges $G.edge\_weight$ and the threshold $threshold_v$.

Graph G is to be partitioned or its edges are to be deleted. The weight of each of the edges of the graph is compared to $threshold_v$; if the weight is found lower than the threshold, the edge is deleted. After processing all the edges, the graph is checked if it has been partitioned into disconnected components. If multiple disconnected components are present, one among them is chosen. The specific algorithm for the choice of the component is beyond the scope of this work. The reader may choose any algorithm existing in literature for the purpose. If the graph is a single connected component, there is nothing to be done. The selected graph or the single connected component graph is stored in *selected_G*. The vertices of the graph *selected_G* are selected as the set of selected files to be prefetched.

5.6 Empirical Observations on Performance of Proposed Solution

We present the empirical observations illustrating the performance of our proposed algorithm. The result of the algorithm is to produce a set of files to be prefetched, which it deems to be useful in the future. The performance of the algorithm is determined based on the accuracy of this prediction.

For test purposes, we have earlier split both MONTAGE and CYBERSHAKE datasets into test and train sets. The training sets are the tasks and files which have already been executed and are to be analyzed to detect a file usage trend. The test set contains tasks that are to be executed in the future. The test set consists of a group of tasks, each task containing a list of files that it will require. The task in the test set is deemed to be the task that is to be scheduled for execution in the future. The proposed algorithm's prediction accuracy is measured by how the file requirements of these tasks are predicted and hence prefetched by the algorithm in advance.

5.6.1 Experiment Details

The objective of our experiment is to test the performance of our algorithm on the following two metrics:

- Hit Ratio: Also known as the number of true positives returned on a test set. The purpose of this test is to measure the prediction accuracy of the algorithm. A higher hit ratio indicates that tasks in the test set can meet its file demands from the set of prefetched files; hence, less data migration would be required. This translates to less execution waiting time and improved task execution performance. This metric tests the algorithm's performance in terms of ability to localize the data.

  For each task, hit ratio is calculated by counting the number of files that the task requires and that is prefetched by the algorithm. An average of all the tasks is reported.

- File under usage: A high hit ratio can also be obtained by prefetching every file in the training set without making any intelligent predictions. Hence, it is necessary to observe what percentage of the files prefetched are utilized by the tasks in the future. If there are many files which are not being used by the tasks, then these have been unnecessarily prefetched and they only waste storage space. An intelligent algorithm would exclude files that are unlikely to be used. If a low file under usage is observed then the algorithm is deemed efficient in respect of storage space utilization. Another way to look at files under usage metrics is a measure of false positives. This metric ensures that the algorithm does not achieve the primary objective of localization in an inefficient manner.

For each task, we determine, how many of the prefetched files have not been used. A percentage of the unused file per task is calculated and an average of all the tasks is reported.

The details of the experiment, performed using Python3, is being presented for ease of reproduction as follows:

- The interdependency between files in the training set is determined and arranged in terms of a dependency matrix. For simplicity, we do not consider the time of request of the file by the tasks.
- Using Python Networkx's graph library, we have created a weighted graph using the dependency matrix created above. The graph's vertices represent the files of the training set, while the edges represent the interdependency, the weight determining the amount of interdependency.
- The graph has disconnected components; only the components with more than one node are considered.
- For each subgraph, the betweenness centrality of each edge is computed.
- A dictionary keeps the list of edges for each corresponding betweenness centrality value. The dictionaries of each of the graphs are merged into one dictionary. The dictionary is sorted based on the betweenness centrality value in descending order of the value of the keys; thus the highest betweenness centrality value is the first entry in the dictionary.

- The maximum weight among all the edges is determined.
- The weight of each edge in the highest betweenness value is used to calculate the threshold by equation 5.5
- Our set of prefetched files, obtained from the computed threshold is compared against graph partitioning based on various thresholds: 0%, 1%, 2%....10%, 20%...80%, 90% and 100% of the maximum edge weight. This means that at each case of n^{th} threshold, edges of weight below $\frac{n \times maximum\_edge\_weight}{100}$ are deleted and the graph is partitioned. 0% threshold includes all the files in the training set and has the highest theoretical hit ratio possible.
- The graph is partitioned based on each of the threshold values calculated, creating multiple disconnected components.
- File requirement of each task in the test set is computed on all the disconnected components created by the threshold based partitioning. The best results among all the partitions based on hit ratio and file under usage metrics are reported for every threshold.

We follow the above steps for both CYBERSHAKE and MONTAGE workflows. Note that we conduct the experiment from the point of view of a single site of the distributed system.

5.6.2 Results and Discussions

We present the experimental results of both CYBERSHAKE and MONTAGE workflows here.

CYBERSHAKE workflow was included as an example of a dataset with a relatively high number of tasks to a low number of files as compared to MONTAGE. It was argued that such a high task to file ratio would lead to a very sparse dependency matrix and graph. The reason for the sparseness is that due to a low number of tasks, a task is likely to request any file, but inversely a file's set of required tasks would be very small. With such a small list, there is very little possibility of intersection among the task requirement set between pairs of files. Since dependency is calculated as the number of intersecting tasks between a pair of files, a low possibility of intersection would result in little or no dependency between the pair of files. The resulting sparsity of the matrix and graph presents little historic data for the algorithm to predict a trend and prefetch the files. Hence, an extremely low hit ratio of 9.8% is observed in both cases; control and the proposed method.

Cybershake's result indicates the fundamental limitation with dependency graph based measures. Data sparsity reduction techniques could be applied as a possible solution to this limitation. Sparsity reduction is likely to create a more informative and dense matrix and graph.

However, sparsity reduction would convert the dependency graph into a complete graph, i.e., every pair of vertices would be connected by an edge. This would destroy even the slightest small world characteristics in CYBERSHAKE's dependency graph. In such a case two things could be tried:

- The easiest choice would be to set a threshold, beyond which low weight edges are deleted and small world characteristic of the dependency graph is regained. However, this solution reintroduces the same problem of finding a threshold that this work tried to solve.
- The definition of betweenness, as defined in equation 5.4 is based on the notion of the shortest path. In a weighted graph, this definition changes taking into account the effect of the edge's weight. Hence, the betweenness centrality is redefined by the following formula:

  $$s_i = \sum_{j=1}^{N} a_{ij} w_{ij} \tag{5.6}$$

  where s_i is the betweenness centrality of the node i; a_{ij} and w_{ij} are adjacency and weight matrices between nodes i and j, respectively. In this case, the algorithm would have to be redefined, since the betweenness centrality of a node is being calculated instead of the edge.

MONTAGE workflow on the other hand was introduced as an example of a dataset having a low task to file ratio, i.e, the number of files far exceeds the number of tasks. By the inverse logic of that applied to CYBERSHAKE, the dependency matrix and graph is likely to be highly dense and informative. A well designed algorithm with proper parameters like threshold is likely to predict a trend from the historic usage data; thereby it will likely be able to prefetch appropriate files, given the threshold and other parameters are accurately provided.

The results show that for the threshold of 0%, the performance of control is better. 0% threshold essentially means that the algorithm selects all the files and essentially performs no work, so this result can be excluded. While the proposed method outperforms the second best of control, i.e., 1%, it is behind the threshold of 0% by around 2.25% only.

The results are provided in Table 5.2. For the control, results up to threshold 9% are shown as the hit ratio up to this threshold is above 50% and hence, could be considered valid. For the results of the proposed measure, only a single threshold is calculated; hence, only one threshold (89.72) is valid for the results of the proposed methods. The results are graphically shown in Figures 5.5 and 5.6.

Table 5.2: Results of MONTAGE: hit ratio

Threshold% Threshold	Subgraph (Number of files) out of 872)	Results Hit Ratio%	Results File Under Usage%
0.0%, 0.0	1 (501 files)	98.74	0.0
1.0%, 73.26	1(432 files)	95.41	3.274963546948744e-107
Calculated by Algorithm 2, 56.75	1(416 files)	96.49	3.1421576357699636e-107
2.0%, 146.572	1(350 files)	88.15	2.5234435836163434e-10
3.0%, 219.858	1 (297 files)	78.75	2.5234435836163434e-10
4.0%, 293.144	1 (265 files)	73.70	2.5234435836163434e-10
5.0%, 366.43	1 (265 files)	73.70	2.5234435836163434e-10
6.0%, 439.716	1(233 files)	68.64	2.526433441489944e-10
7.0%, 513.002	1(231 files)	68.27	2.526433441489944e-10
8.0%, 586.288	1(205 files)	64.11	2.526433441489944e-10
9.0%, 659.574	1(199 files)	63.07	2.526433441489944e-10

5.7 Conclusions

The focus of this chapter is to study the dependency graph from a graph theoretic/social networks perspective. An additional objective was to utilize the knowledge gained through analytical study to solve

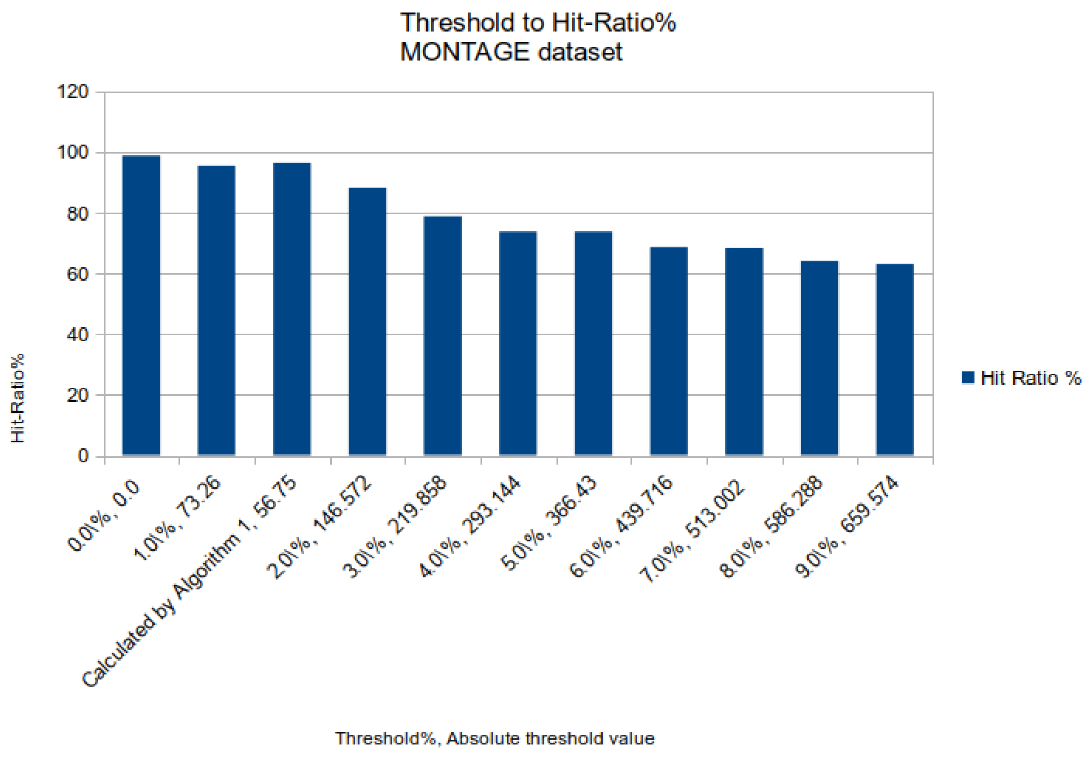


Figure 5.5: Tau Correlation for each incremental betweenness values in MONTAGE

the threshold finding problem to partition the dependency graph, one of the most important steps in data prefetching algorithms.

As per our first objective, we deduce that the concept of the betweenness centrality of edges indeed strongly correlates with the weight of the edges. This provides us with a solution to the threshold problem. Graph partition threshold is essentially the determination of significant and insignificant edges based on the weight. The high correlation of betweenness centrality and edge weight results in the accumulation of almost all the significant edges (edges with high weight) at the highest betweenness centrality value. Hence, the average weight of edges with the highest betweenness value serves as a threshold to determine what is significant and what is not.

Based on the concept, an algorithm is also developed. Empirical observations show that the performance of the algorithm is close to the optimal theoretical performance possible.

There are scopes for some improvements, however. The performance observed is close to the theoretical performance, but not optimal. This opens up further scope for improvement; perhaps a combination of some other metrics of graph theory to refine the betweenness centrality based approach could be tried. It has also been observed that performance is very poor for a sparse dataset. This is not something unique to this problem. Without much data, any prediction-based method will fail and this problem is no exception. The issue is that the nature of this problem prevents the use of standard sparsity reduction techniques as discussed earlier. Some possible workaround has been suggested, which needs to be developed and verified.

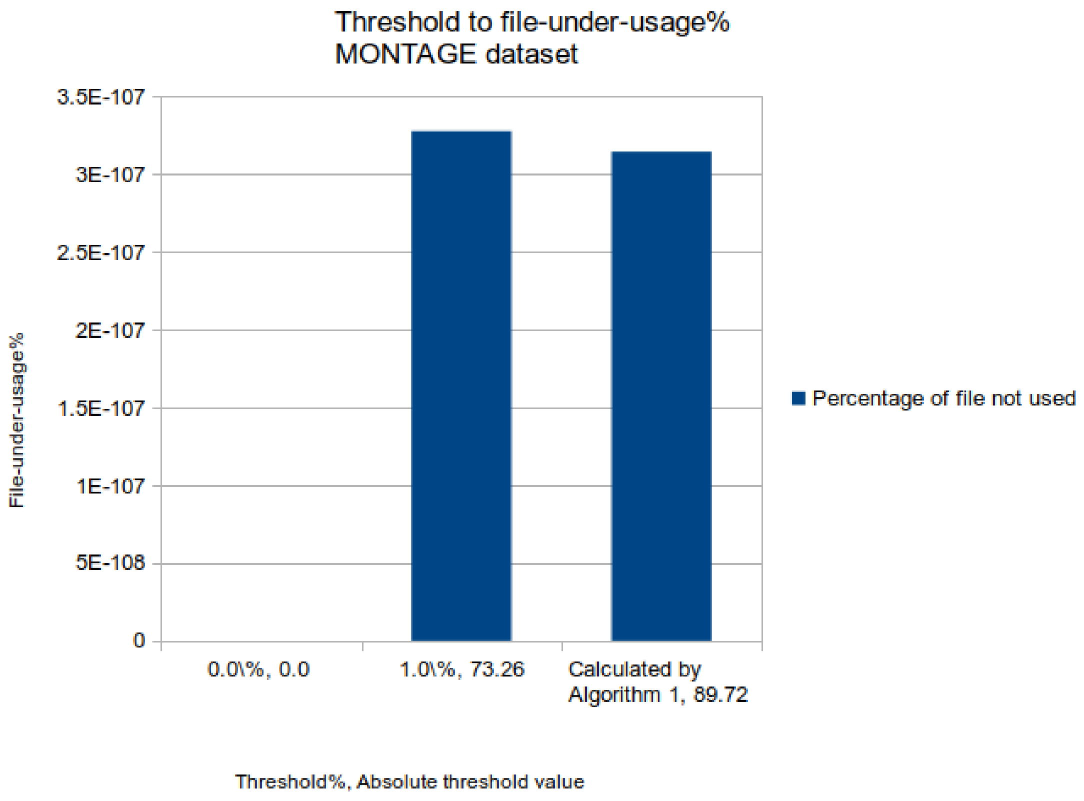


Figure 5.6: Tau Correlation for each incremental betweenness values in MONTAGE

Chapter 6

Data Placement in Distributed Systems under Storage and Network Constraints

6.1 Introduction

In a computerized storage system, the reliability of the data being stored is the most important and non-compromisable criterion. It is necessary to ensure that data stored in the system remains as it is, without any alteration to the data. Hence, a storage service provider should not only protect the subscriber's data against intentional malicious data modifications but also against accidental modifications inherent due to natural causes. Separate strategies are in place to deal wuth the two types of data protection. In this work, protection agaist non malicious accidental modification is addressed.

As observed in the system description section of the previous chapter (Section 3.2, Figure 3.1), in a scientific workflow environment, generally local compute clusters have their data backed up at a secure server on the internet. If data is lost in the local cluster due to node failure, a copy from backup is downloaded via the internet. Data loss due to nodes going offline in the local cluster causes increaded internet traffic, which should be avoided as much as possible. Hence, if data can be recovered from the remaining online nodes in the local cluster then internet traffic could be avoided. In this chapter, the focus will be on data placement in a distributed system(local cluster) such that occassional node failures do not require data retrieval over the internet.

In recent years, technological enhancement has allowed for the large scale adoption of distributed computing wherein multiple separate devices connected over a network have a common goal. Data storage reliability management and its contribution to the prevention of data loss preceded distributed systems. The use of error correction codes for error detection and recovery have been in use or a long time now. However, generating and storing correction codes is a costly undertaking. Much research has been done to create an efficient error correction codes that offer both high reliability and have a low cost, The research into this is an ongoing effort.

The emergence of distributed computing and its applications open up new opportunities in our quest for low cost and reliable data storage systems. As the results of this work would demonstrate, intelligent placement of data in such systems can lead to a highly reliable storage system with a low storage cost.

Since distributed system is being considered, another parameter to be considered is network cost. The network is not a cheap resource and its economic usage must be considered as well. It has to be investigated whether the three objectives: high reliability, low cost storage, and low network storage can be accomplished together at all or a tradeoff exists. In the event of joint optimization of all objectives bieng possible, a data placement solution to such a system must be formulated.

This chapter is organized into seven sections including introductory section. In Section 2, some of the necessary concepts discussed in the literature survey is revisited and elaborated upon. In Section 3, the problem would be formally and elaborately described and a mathemantical formulation is built. The feasibility of joint optimization of the stated objectives has been investigated and the empirical observations are reported and their implications are discussed in Section 4. The algorithmic solution to the problem is presented in Setion 5. The performance of the solution is empirically validaed and the results are presented in Sextion 6. The concluding remarks are presented in Sextion 7.

6.2 Important Background Concepts

The concept of data allocation and storage budget has been discussed in the literature survey. It may be recalled that the data location has to be such that high reliability is ensured with a minimal storage budget. To this effect, the following allocation pattern may be classified as follows:

- Symmetric allocation: Symmetric allocation is such that each node of the distributed system is allotted the same amount of data.
- Non symmetric allocation: Non symmetric allocation is such that each node of the distributed system is allotted a different amount of data. It may happen that some nodes may not be allotted any data at all.

Besides the above classification, data allotment may also be classified based on how the distributed nodes are used for data storage:

- Maximal spread: Most, if not all the nodes in the distributed system, are used to allocate data.
- Minimal spread: The data allocation is only concentrated at a few nodes of the distributed system.

Based on the above classifications, the following four allocation patterns are possible:

- Symmetric maximal spread: The data is allocated equally among all the available nodes in the distributed systems.
- Symmetric minimal spread: The data is allocated equally, however only a subset of nodes of the distributed systems are used.
- Non symmetric maximal spread: While all the available nodes in the distributed system are used, the data is not allocated equally in each of the nodes.
- Non symmetric minimal spread: A subset of the nodes in the distributed system are used and data allocated in each node is not equal.

Besides the above allocation patterns, the storage allocation and access may be classified based on the following:

- Probabilistic: The allocation or access of data to and from a node in the distributed system is based on a probability distribution.

- Deterministic: The allocation and access of data to and from a node in the distributed system are based on a predetermined manner which remains fixed every time.

Hence, based on the probabilistic and deterministic allocation and access policies, the following allocation and access policies are possible:

- Probabilistic allocation Deterministic access: The allocation of data is made in a probabilistic manner, but the access of data is made in a deterministic way
- Probabilistic allocation Probabilistic access: Both data allocation and access are made in a probabilistic manner
- Deterministic allocation Deterministic access: Both data allocation and access are made in a deterministic manner
- Deterministic allocation Probabilistic access: The allocation of data is made in a deterministic manner, but the access of data is made in a probabilistic way

There are two types of node that a distributed system might have:

- Homogeneous Nodes: The characteristics of the nodes in the distributed system are similar in terms of computation and storage.
- Heterogeneous Nodes: The characteristics of the nodes in the distributed system are different either in terms of computation or storage or both.

The last two concepts that need to be discussed are that of Storage Budget and probability 1 error recovery regime. Probability 1 error recovery regime is the concept that the distributed system saves data in such a way that any data can be recovered from corruption due to non malicious reasons. This ensures high reliability of the system in terms of data storage. Storage budget is used to measure the amount of data that is being stored in the system including the redundancies that ensure the probability 1 error recovery regime. Similarly, network budget is used to measure the amount of network usage due to data migration during run time of the tasks of the scientific workflow. A composite budget or budget is a combination of both storage and network budgets, which are to be minimized.

Error correction codes are redundant information stored in the system such that data loss to a certain extent may be tolerated by the system. The data is recovered by mathematically combining the existing data and the redundant parity data. The advantage of this error recovery is that data can be reconstructed, thereby avoiding the download of data from the internet. The disadvantage is that the mathematical operation in data retrieval may be computationally expensive. It may be noted that error correction can retrieve data only till a certain data loss.

In the work presented in this chapter, only homogeneous nodes will be considered. The literature in distributed storage allocation problem attempts to find the best allocation and spread pattern for data, such that the composite budget may be decreased; while retaining the probability 1 error recovery regime.

One of the advantages that distributed system offers is that by intelligently placing data on the system, the storage budget may be decreased while maintaining high system reliability.

6.2.1 System Description

The distributed system being considered is similar to the one described in Figure 3.1 of chapter 3. Since, the aim of this work is to avoid internet traffic (except initial download, which is not the concern of this work). Only the local cluster with its constituent distributed system is being considered. The distributed system considered in this case is illustrated in Figure 6.1:

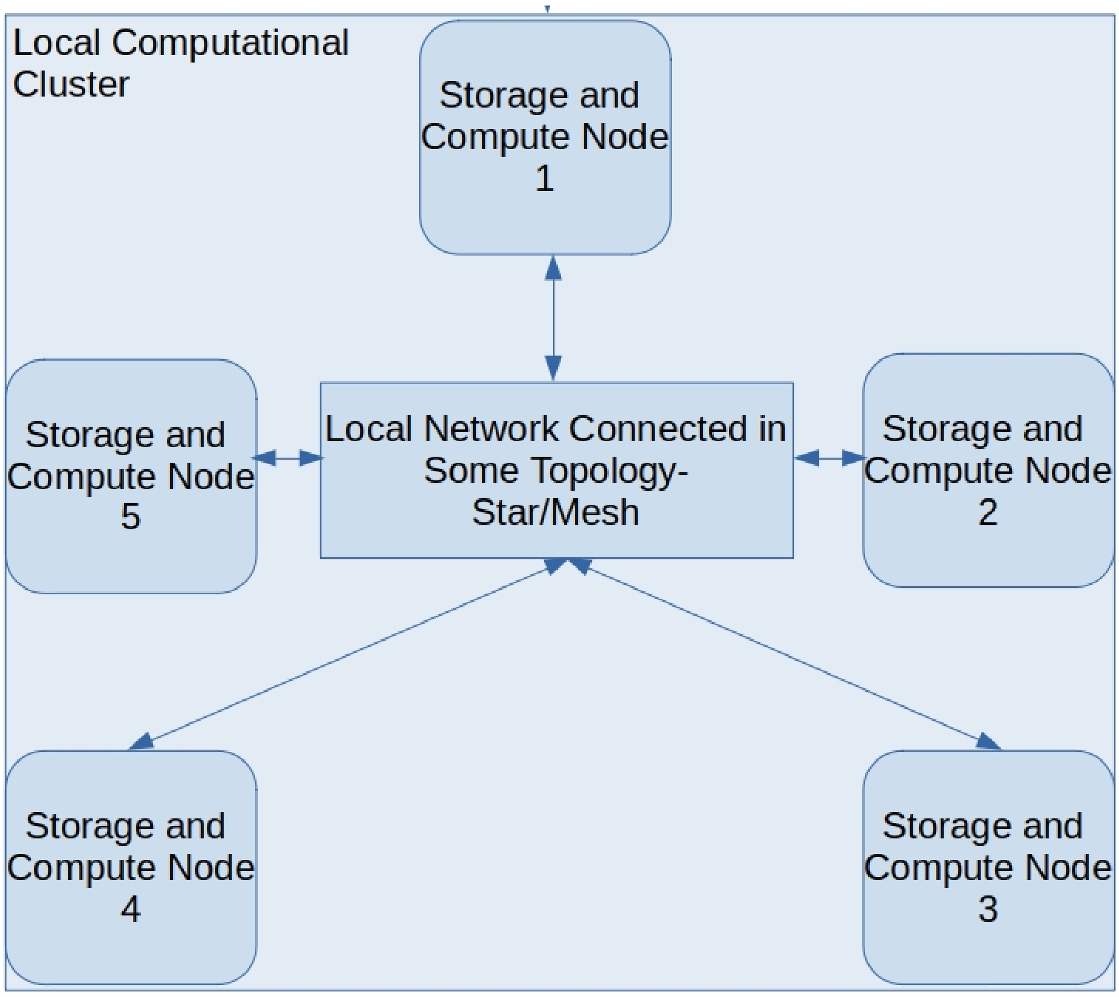

Local Computational Cluster
Storage and Compute Node 1
Storage and Compute Node 5
Local Network Connected in Some Topology-Star/Mesh
Storage and Compute Node 2
Storage and Compute Node 4
Storage and Compute Node 3

Figure 6.1: Distributed system under consideration

In this work, the distributed system illustrated in Figure 6.1 and error correction code based system is utilized.

6.3 Problem Statement

The works available in the literature have proposed different solutions to the problem of lowering the storage budget while maintaining the probability 1 error recovery of the distributed system. The central idea is that how a distributed system's data may be placed, such that the data can always be retrieved with minimum redundant data stored in the system.

To achieve the above objective, the network usage issue has been missed out. In a distributed system that not only stores data but also caters to computation as in scientific workflow, network congestion, and the resulting delay in transfer of data leads to severe performance penalty. Hence, high network usage is considered a serious problem as discussed in the last chapter. Also, a high data transfer indicates non localized data placement, i.e., data is placed at a different node of the distributed system than the one where it is needed.

From the above discussion, it may be summarized that a successful distributed compute cum storage system would require optimization of both storage and network usage while maintaining high reliability through a probability 1 recovery regime.

To formally define the problem being addressed in this work, the following optimization equation is presented below:

$$Minimize \quad T = \sum_{i=0}^{n} x_i + \sum_{i=i,j=1}^{n} y_{ij} \tag{6.1}$$

Subject to

$$x_i \geq 0 \quad and \quad y_{ij} \geq 0 \quad and \quad i \neq j \tag{6.2}$$

$$\sum_{i \in r} x_i \geq 1 \quad and \quad \sum_{ij \in r} y_{ij} \geq 1 \quad and \quad \forall r \in R \tag{6.3}$$

Where

$$R = \binom{n}{r} \quad r \subset 1, 2, ..., n \tag{6.4}$$

T in equation 6.1 denotes the composite budget which consists of both the storage budget x_i and network usage budget y_{ij}. i is the node of concern for which storage budget is being considered. j is the source node from which a data if required, would have to be migrated to node i. y_{ij} is the network budget, i.e. the amount of network usage incurred by the system.

Equation 6.2 represents the constraints of the budget variables x_i and y_{ij}. Since storage and network usage cannot have a negative number, these variables are constrained to have a value greater than 0. Further, since the source and a destination node in data transfer should be different, so i and j cannot have the same value.

Equation 6.3 represents all the combinations in which a node can be selected from the total number of nodes, which forms a set R. In $\binom{n}{r}$, n is the number of nodes to be selected for retrieval, while r is the total number of nodes.

Equation 6.3 represents the constraint that the summation of storage budget and network budget of all the node combinations is at minimum 1 or greater than it.

The following related problems are addressed in the work described in this chapter [155]:

- Network simulation study
- Constraint optimization feasibility study
- Optimized Data placement under the constraints

6.4 Feasibility Studies

In this section, the feasibility studies are discussed. That is, which allocation and spread model translates into lower network usage and whether network usage and storage space required can be jointly minimized in a probability 1 recovery regime, are studied.

Throughput is defined as the number of packets that pass through a network per second. In essence, throughput measures the efficiency of the network. Data migration is an inevitability in a distributed system due to space constraints. When data migration does occur, such migration should be completed as soon as possible. With high network throughput, data migration is completed faster, thereby improving task performance by reducing stall periods.

The four allocation and spread pattern combinations discussed in Section 6.2 are compared to find out which spread and allocation patterns result in high throughput in a data center network topology. A spread and allocation pattern with high throughput will make a good candidate for use in distributed systems.

6.4.1 Network Simulation Study

The network simulation study is aimed at understanding the network characteristics of the four combinations of data allocation and spread, through the metric of throughput. The combination of data allocation and spread which produces the high throughput would be considered as a better candidate for use. The four combinations of storage allocation and spread may be recalled as follows:

- Symmetric allocation minimal spread
- Symmetric allocation maximal spread
- Non symmetric allocation minimal spread
- Non symmetric allocation maximal spread

Simulation setup

Data in a large distributed storage is stored and computed in data centers, which are specialized reliable storage facilities of data. Hence, a data center network is considered for the network simulation study. The network topology in a data center network comes in a variety of forms like Fat tree, Dcell, Bcube, etc. Since the metric being studied is the throughput, the throughput due to extraneous conditions should be as limited as possible. Fat tree has proven capability of delivering high throughput and low latency[156]; hence, it becomes our obvious choice of topology for simulation.

Simulation is performed using NS-3. Nanyang Technological University's implementation of Data Center Network [157] on NS-3 has been used for simulating the data center network.

The essential simulation parameters are listed in Table 6.1

For each allocation pattern, symmetric and non symmetric, 8 spread patterns are considered.

- Data placed in only one server (node): minimal allocation
- Data placed in the first 8 servers: minimal allocation: half of the available servers are used
- Data placed in the last 8 servers: minimal allocation: a different perspective of the first 8 server allocation
- Data placed in the middle 8 servers: minimal allocation: a different perspective of the first 8 and last 8 server allocation.

Table 6.1: Essential Parameters

Parameters	**Value**
Total number of nodes	50
Maximum number of access nodes	16
Number of access nodes	1, 8, 16
Topology	Fat tree
Links	edhe host: CSMA, rest: P2P
Data rate	1 Gbps
Delay	0.001 ms
Application	On-Off
Packet size	1024 bytes
On-Off data rate	default
Total data transferred	125 MB
Total client nodes	5

- Data allocated in 8 odd interleaving servers: half of the available servers used, but in a different arrangement than the last three arrangement
- Data allocated in 8 random servers: half of the available servers used, selection of the servers being random.
- Data allocated in all 16 servers: maximal allocation

Results and Discussions

The results of the network simulation study is presented in Figures 6.2 to 6.4.

The results under symmetric allocation are illustrated in Figure 6.2 while the results under non symmetric allocation are illustrated in Figure 6.3. The consolidated results comparing both the spreads are illustrated in Figure 6.4.

The salient observations are as follows:

- From Figure 6.4, where the consolidated results are available, it can be inferred that under all allocation conditions non symmetric spread of data yields a higher throughput. Thus, it is be a prudent decision to allocate different amounts of data on different nodes/servers of the distributed system

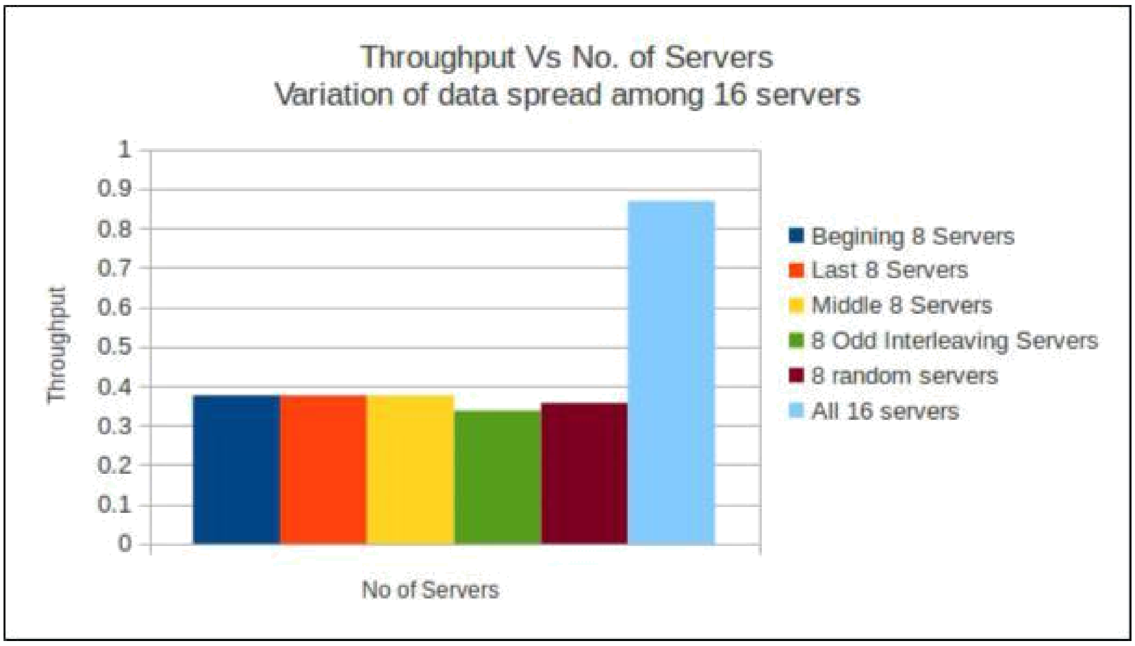


Figure 6.2: Results of non symmetrical allocation

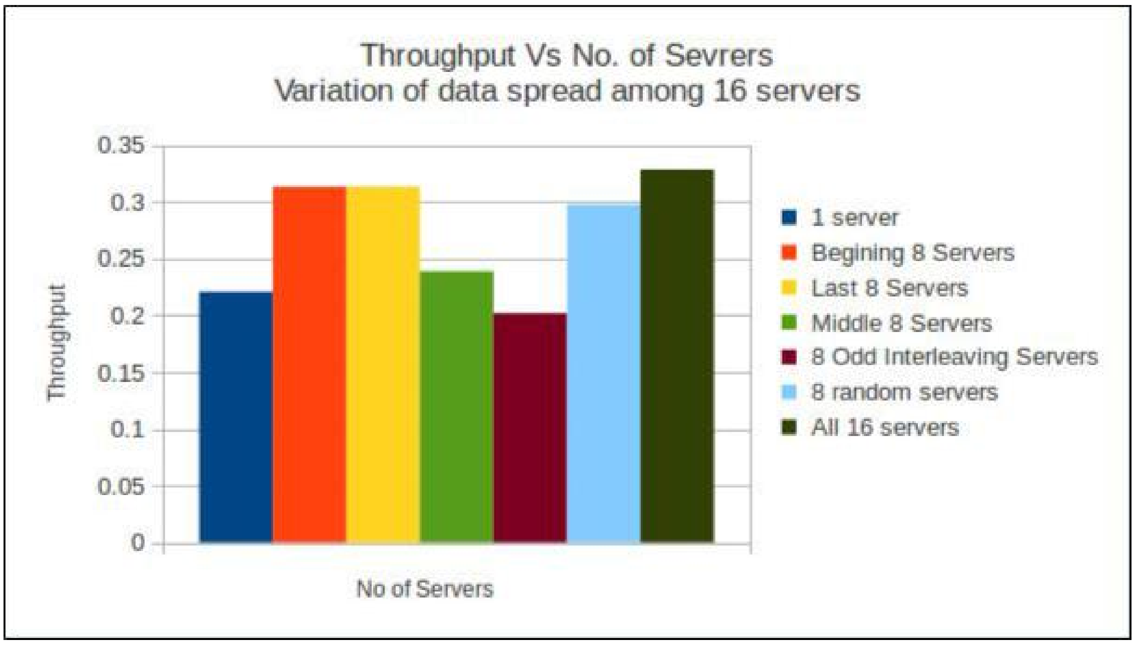


Figure 6.3: Results of symmetrical allocation

- In non symmetric allocation, we observe that maximal allocation yields the best throughput, by a considerable margin over other allocation patterns. All the minimal allocation patterns have almost the same throughput; whether only one is used or half of the available nodes are used.

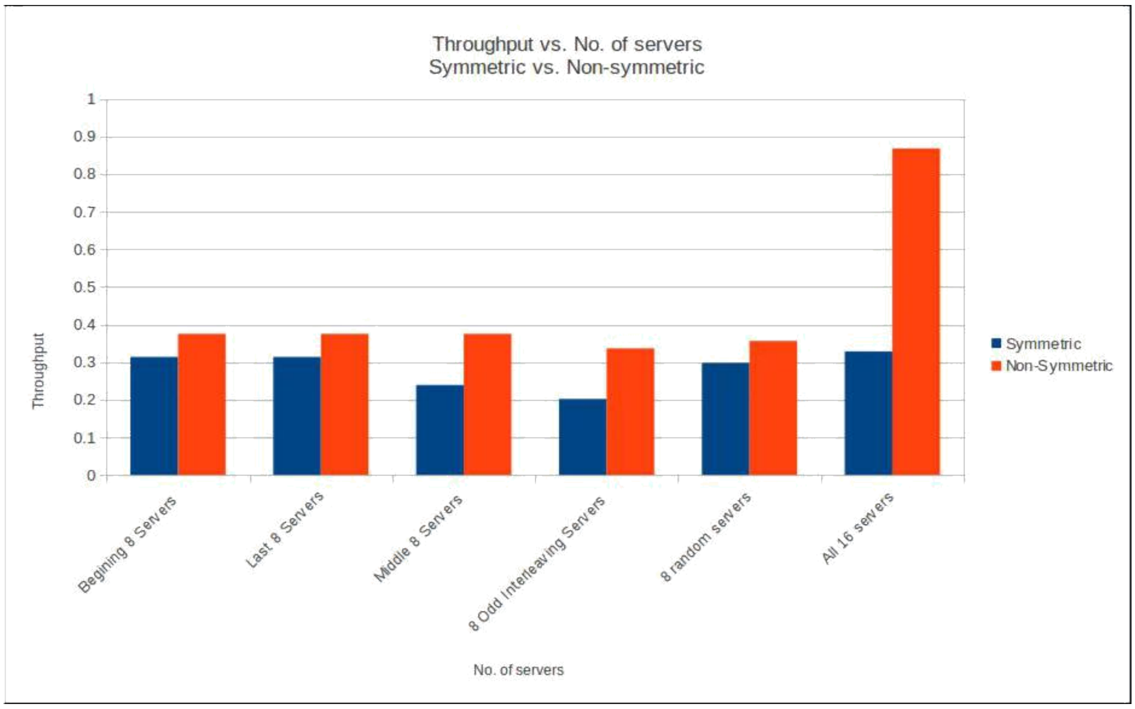


Figure 6.4: Consolidated results

Constraint Optimization Problem

The feasibility study for the constraint optimization problem is aimed to study whether the twin objectives of network usage and storage space minimization as stated in equation 6.1, can be jointly optimized keeping the constraints stated in equations 6.2 to 6.4 in mind.

The problem has been studied using a hybrid of two algorithms: Genetic Algorithms (GA) and Particle Swarm Optimization (PSO); GA PSO [158]. The GA part explores the solution space efficiently to find the global minima instead of being stuck in the local minima. While the PSO part enhances faster convergence of the algorithm which would be slow in the case of a pure genetic algorithm.

The essential parameters of the experiment are as follows:

Results and Discussions

From Figure 6.5, we can deduce that the cost of storage and network usage jointly rises and falls in every case. Only in the case of node 1, negligible variation can be observed. In this simulation, bandwidth cost represents the network usage. Hence, we can conclude that both storage and network usage can be jointly optimized.

Table 6.2: Essential Parameters

Parameters	**Value**
Population size	150
Maximum generations	100
Probability of crossover	0.85
Probability of mutation	0.15
Beta (model coefficient)	1.5
Number of storage nodes	5
Number of nodes selected for retrieval	2
Number of runs	50

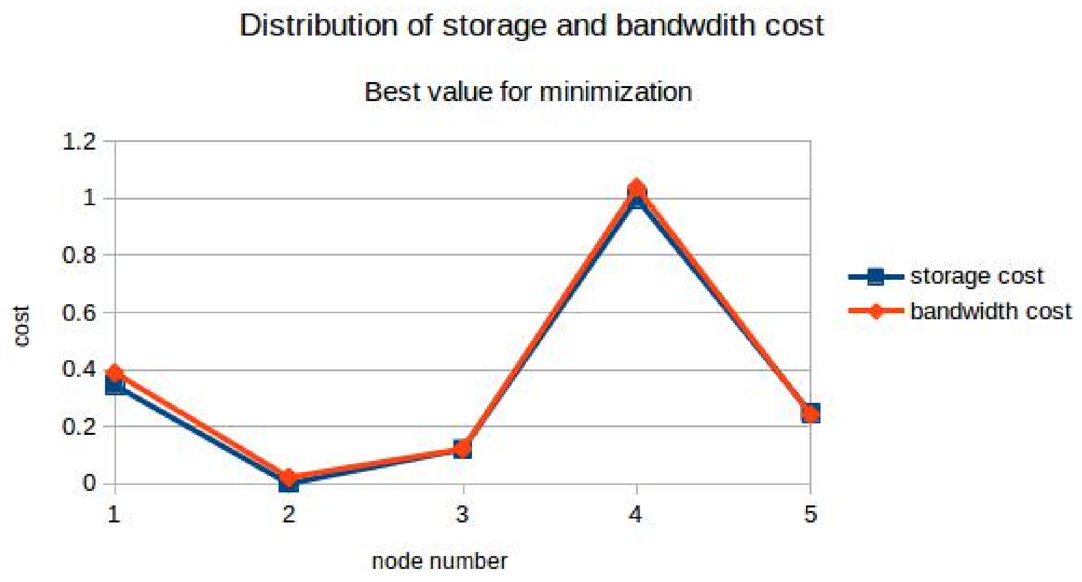


Figure 6.5: Graphical view of best value for minimization

Summary

We had conducted two separate experiments: one to study which allocation and spread patterns is best for data placement and another to study the feasibility of joint optimization of minimizing network and storage space usage. In summary, the empirical observations point that a non symmetric spread with a maximal allocation is the best placement strategy and joint optimization of minimizing storage and network usage cost is possible.

Based on these empirical observations, a data placement solution is proposed.

6.5 Data Placement Solution

In the last section, the feasibility of data placement in the distributed system jointly optimizing both network and storage space usage has been established. Thus, a data placement solution with the same objectives and constraints is being proposed in this section.

Considering the system at hand, high reliability is an essential requirement. Hence, this desired trait of data placement is not considered as an optimization objective, but rather as a constraint to the optimization problem. A candidate data placement solution may provide the minimum network and storage usage; however, if the reliability condition of a probability 1 recovery regime is not achieved, this solution will not be considered.

Considering that two objectives are to be optimized: minimum network usage and minimum storage space, a multiobjective optimization strategy, more appropriately a bicriteria optimization strategy needs to be applied. Since both objectives can be optimized simultaneously, the optimization problem is trivial. Hence, finding a set of Pareto optimal solutions can be avoided.

Optimization problems are essentially state space search problems. All the possible data placement solutions are already known. The task of the algorithm is to find the optimal data placement. To this effect, a fast efficient, and nearly complete search algorithm is required. Hence, a metaheuristic based approach has been considered for this purpose.

The algorithm being proposed is a hybrid of two metaheuristic approaches; Genetic Algorithm (GA) and Particle Swarm Optimization(PSO). The reason behind the use of these two algorithms is explained below.

- Genetic Algorithm: Genetic Algorithm adapts the biological process of natural selection which is used by species to evolve into a population that is more adaptable to survive in their environment. In terms of the state space search problem, the most optimal solution is considered the state of the fittest population. Starting from a random state, GA stochastically guides the search agents to the optimal point, such that the population of the search agents at the optimal point is maximized. The advantage of GA is that due to its stochastic nature, it explores the entire search space even in areas that may not look promising at first. Hence, GA's have a higher chance of escaping the local minima and discovering the global minima. The biggest disadvantage of GA is its very slow convergence to the optimal solution

- Particle Swarm Optimization: In this approach, the candidate data placement solutions are modeled as particles that collectively form a swarm of particles. In each iteration, each particle's position and velocity are adjusted and the swarm is observed. The simplicity of the algorithm leads to its quick convergence to a solution. However, this algorithm has the disadvantage of providing a sub optimal solution since it can get stuck in local minima.

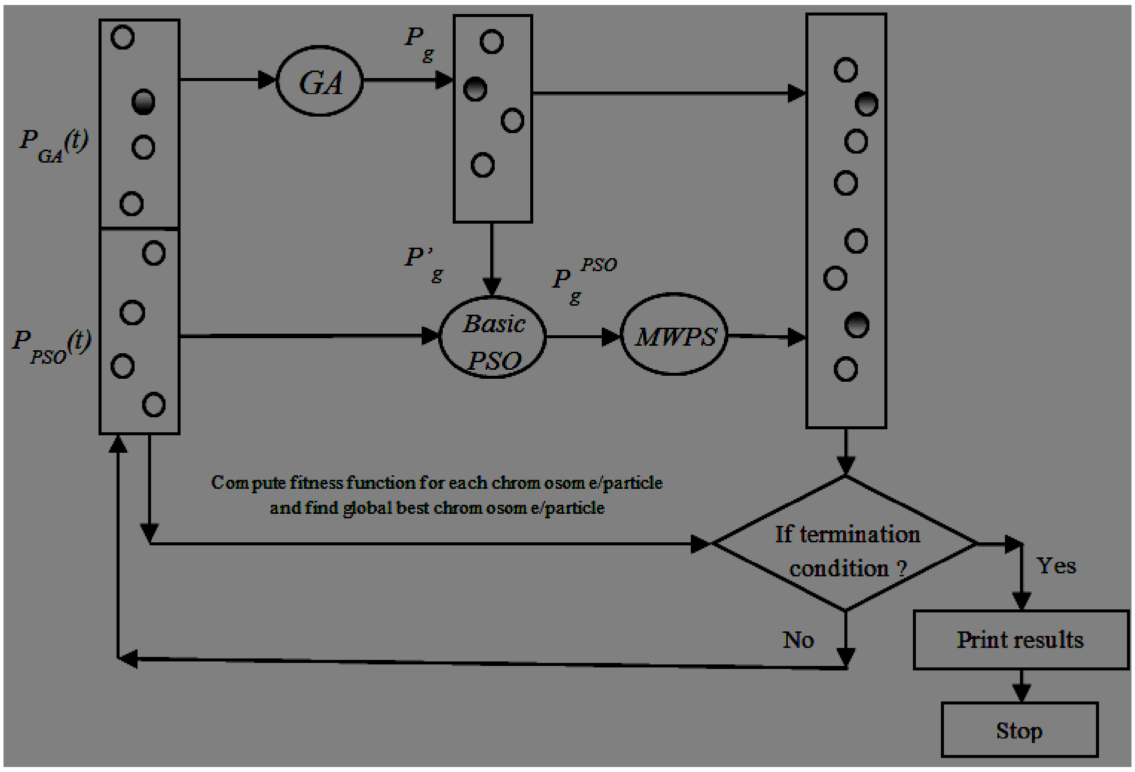
It may be noted that GA and PSO are complementary to each other. GA's slow convergence is complemented by the fast convergence of PSO. PSO's inability to escape local minima is complemented by GA's ability to find the global minima. Hence, a hybrid approach, when properly applied, utilizes the best of both approaches: a fast converging optimal solution finding algorithm.

We must note that the metaheuristic algorithms work on estimates of goodness and are randomized. Further, they have to be stopped after some generations. Hence, an optimal solution is not guaranteed. However, in the domain of soft computing, a close solution to optimal is acceptable. Hence, a near optimal solution is acceptable in this case.

The problem formulation is the same as described in equations 6.1 to 6.4. The hybrid algorithm being used is GA MWPSO, where MWPSO is the Mean Weighted Particle Swarm Optimization.

6.5.1 Proposed Algorithm

We present the flowchart and the pseudocode of the algorithm first. Subsequently, the algorithm is explained.



$P_{GA}(t)$
$P_{PSO}(t)$
GA
P_g
P'_g
Basic PSO
P_g^{PSO}
MWPS
Compute fitness function for each chromosome/particle and find global best chromosome/particle
If termination condition ?
Yes
No
Print results
Stop

Figure 6.6: Flowchart of the data placement solution

The algorithm takes NP number of population, P_c is the probability of crossover, P_m is the probability of mutation, and M_g is maximum number of generations as parameter inputs. There are three functions: Main, ApplyPSO, and ApplyGA. The three functions are described as follows.

- Main: This function creates an equal number of population for GA and PSO as dictated by NP. Time t is initialized and so are the populations. The initial fitness function and best chromosome/particle determination for GA and PSO are carried out in this function. If the termination condition is not met then GA and PSO specific operations are carried out by calling ApplyGA and ApplyPSO functions respectively. The outputs from these functions are used to update the initial/last results which are used as a parameter in the next time iteration.

Algorithm 4 GA MWPSO Algorithm

```
procedure Main($NP, P_c, P_m, M_g$)
    $NP_{GA}, NP_{PSO} \leftarrow CreatePopulation(2 * NP)$
    $t \leftarrow 0$
    $P_{GA}(t), P_{PSO}(t) \leftarrow InitializePopulation(NP_{GA}, NP_{PSO})$
    $f(x_i)_{GA} \leftarrow Fitness(P_{GA}(t))$
    $f(x_i)_{PSO} \leftarrow Fitness(P_{PSO}(t))$
    $P\_Chr_{best} \leftarrow FindBestChromosome(P_{GA}(t), f(x_i)_{GA})$
    $p\_Ptr_{best} \leftarrow FindBestParticle(P_{PSO}(t), f(x_i)_{PSO})$
    while $Termination\_Check == False$ do
        $t \leftarrow t + 1$
        $P\_Chr_{best}, f(x_i)_{GA}, P_{GA}(t) = ApplyGA(P_{GA}(t-1), P\_Chr_{best}, P_c, P_m, M_g, t)$
        $P\_Ptr_{best}, f(x_i)_{PSO}, P_{PSO}(t), velocity, position = ApplyPSO(CombinePopulation(P_{PSO}(t-1), P_{GA}(t-1)), P\_Ptr_{best}, t, P\_Chr_{best})$
    $Print(P\_Chr_{best}, f(x_i)_{GA})$
    $Print(P\_Ptr_{best}, f(x_i)_{PSO})$
procedure ApplyGA($P_{GA}(t-1), P\_Chr_{best_{t-1}}, P_c, P_m, M_g), t$)
    $P_{GA}(t) \leftarrow ApplyCrossover(P_{GA}(t-1))$
    $P_{GA}(t) \leftarrow ApplyMutation(P_{GA}(t))$
    $f(x_i)_{GA} \leftarrow Fitness(P_{GA}(t))$
    $P\_Chr_{best} \leftarrow FindBestChromosome(P_{GA}(t), f(x_i)_{GA})$
    $P\_Chr_{best} \leftarrow CompareChromosome(P\_Chr_{best}, P\_Chr_{best_{t-1}})$
    $returnP\_Chr_{best}, f(x_i)_{GA}, P_{GA}(t)$
procedure ApplyPSO($P_{PSO}(t-1), P\_Ptr_{best_{t-1}}, t, P\_Chr_{best}$)
    $velocity \leftarrow GetVelocity(P_{PSO}(t))$
    $position \leftarrow GetPosition(P_{PSO}(t))$
    $f(x_i)_{PSO} \leftarrow Fitness(P_{PSO}(t))$
    $p\_Ptr_{best} \leftarrow CompareParticle(P_{PSO}(t), P_{PSO}(t-1), f(x_i)_{PSO})$
    $P_{PSO}(t) = ApplyMW(p\_Ptr_{best}, position, velocity, p\_Ptr_{best}, P\_Chr_{best})$
    $returnP\_Ptr_{best}, f(x_i)_{PSO}, P_{PSO}(t), velocity, position$
```

- ApplyGA: This function applies crossover and mutation on the population followed by the fitness function calculation and best chromosome determination. If the present iteration's best chromosome is better than the last iteration's best chromosome, then the best chromosome is updated with the present one. The function returns the best chromosome, new GA population, and fitness value.

- ApplyPSO: This function applies position and velocity for each particle in the population followed by the fitness function calculation and best particle determination. If the present iteration's best particle is better than the last iteration's best particle, then the best particle is updated with the present one. The function returns the best particle, new PSO population, fitness value, velocity, and position.

NP is the number of population. At the beginning of the algorithm, the population is doubled, such that half of the population is assigned to the GA and the other half to PSO. For both GA and PSO, the algorithm iteratively computes in parallel to execute for initialization, computation of fitness function, and determining the best chromosome or particle. This computation continues till the termination condition is met. In each iteration the following stochastic updates are made.

- GA: Crossover and mutation operations are performed on population from the last iteration to obtain a new population. The best chromosome from the new population is determined and compared to the best chromosome from the last iteration's population. The best among these two chromosomes is stored. The tournament selection process is used to select the new population.
- PSO: Each particle's position is compared with the position of all the particles to improve its position. The new position is computed by calculating the velocity of each particle.

Mean Best Position ($mbest$) is used to update particle position, which is defined as the average center of gravity of each particle positions. it is defined as follows:

$$mbest(t) = [m_1(t), m_2(t), ..., m_n(t)] \tag{6.5}$$

$$m_j(t) = \frac{1}{M}\sum_{i=1}^{M} P_{i,j}^{best}(t) j - 1, ...n \quad and \quad M = 2NP \tag{6.6}$$

Where, $P_{i,j}^{best}(t) = [P_{i,1}^{best}(t), P_{i,2}^{best}(t), ..., P_{i,n}^{best}(t)]$ is best position for particle x_{ij}. In the algorithm $P_{i,j}^{best}(t)$ is described by p_Ptr_{best}

Fitness function $f(x_i)$ is calculated, where $i = 1, 2, ...2.NP$ and compared with P_t^{best}. A comparison is made whether $f(x_i)$ is greater than $P_{i,j}^{best}$ and make $P_{i,j}^{best} = f(x_i)$. P_g^{PSO} is updated as $max(P_g^{PSO}, P_{i,j}^{best})$. Subsequently, the particle position is updated using the basic PSO formula as described in the equation below. A new local and global best solution (P_{ij}^{best} and P_g^{PSO} is found. Alternately, if P_j^{best} is the best then next iteration follows using the same steps. In the algorithm, $f(x_i)$ is denoted by $f(x_i)_{PSO}$ and P_g^{PSO} is denoted by $P_{PSO}(t)$.

The best position of a particle is updated as follows:

$$P_{ij}^{best}(t) = \eta P_{ij}^{best}(t) + (1-\eta)P_g^{PSO}(t), \quad where \quad \eta = rand(0,1) \tag{6.7}$$

The new particle position is calculated as follows:

when $rand(0,1) > 0,5$,

$$x_{ij} = Pij^{best}(t) - \beta.|mbest(t) - x_{ij}|.ln(\frac{1}{u}) \quad where \quad u = rand(0,1) \tag{6.8}$$

else,

$$x_{ij} = Pij^{best}(t) + \beta.|mbest(t)) - x_{ij}|.ln(\frac{1}{u}) \tag{6.9}$$

Where β is the model coefficient

6.6 Empirical Observations

The algorithm's performance is validated with the experiment described in this section and the empirical observation that follows.

Algorithm 4 has been implemented in C++, which was executed on an x86 based processor of speed: 2.10 GHz, running G++ and Ubuntu OS platform.

Keeping in mind that a metaheuristic based algorithm has been used, the problem considers different sets of random fuzzy numbers using triangular membership function for each runs.

Availability of data for given storage and network usage budget is considered as the evaluation criteria. We have n storage nodes, with each node storing a fraction of the data. If r out of n nodes are used for access, a total of $\binom{n}{r}$ combination of nodes is possible for access. Hence, a greater number of $\binom{n}{r}$ will translate to greater viability that the data will be accessed by the system.

Considering the above evaluation criteria, the performance of the algorithm is measured on the following metric.

- Probability of Access: The main point of observation is to note whether the data stored in the distributed system is accessible with minimum budget. We need to figure out, how many viable possibilities are available out of the total number of possibilities. More the number of such viable possibilities, the greater is the availability of the data. Such greater availability of data ensures the better run time performance of the tasks that need these data. The probability of access measures the availability of data in the distributed system.

Table 6.3: Essential Parameters

Parameters	**Value**
Population size	150
Maximum generations	100
Probability of crossover	0.85
Probability of mutation	0.15
Beta coefficient	1.5
Number of storage nodes	10
Number of access nodes	5
Number of runs	50
Storage cum Network budgets (T)	$\frac{9}{7}, \frac{11}{9}, \frac{13}{11}, \frac{15}{13}$

Results and Discussions

From Figure 6.7, we can make the following observations:

- At T= $\frac{11}{9}$, the best performance is observed. The probability of access reaches 60% at $\binom{10}{4}$ and reaches 100% at $\binom{10}{6}$. Thus with only 22% overhead, we obtain a 100% probability of access with 210 combinations ($\binom{10}{6} = 210$) of retrieval nodes.

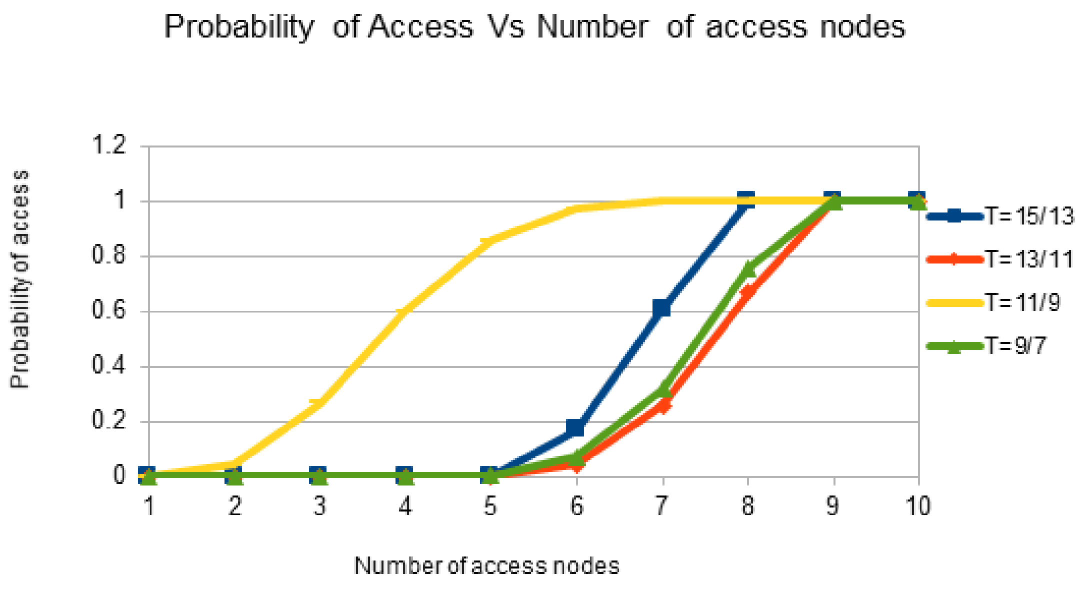


Figure 6.7: Probability of Access

- At $\binom{10}{9} = 10$ combinations of retrieval nodes, every budget has a probability of access at 100%.

In summary, it can be agreed that for full availability, there are 210 possible combinations in which data can be placed. This is possible at a budget of T= $\frac{11}{9}$, i.e. about 22% overhead, which is the second lowest value considered. At the lowest budget of T=$\frac{9}{7}$, i.e. with about 12% overhead, probability of access of 100% is obtained at $\binom{10}{9}$ or 10 combinations; while a decent probability of access at about 80% is possible at $\binom{10}{8}$ or 45 combinations.

6.7 Conclusions

Storage, network usage and reliability are three interconnected factors that contribute to the performance of the scientific workflow execution in a distributed system. Savings in storage space makes room for more dependent data to be colocated, thereby reducing migration and hence network usage. High reliability of data retrieval assures limited use of precious internet resources to download a backup copy of the data. Network usage is an inevitability in distributed systems due to storage space constraints. Hence, a data placement that results in minimal network usage is desirable.

In this work, the spread and allocation pattern was studied and the best combination was established through a simulation study. Further, it was established that both objectives of data and network usage can be jointly optimized with high system reliability in terms of data retrieval. Empirical observations suggest that high data availability is possible with a very low budget through the data placement algorithm proposed in this work.

Chapter 7

Conclusion

This work has been aimed at studying data management in distributed systems, especially with the data placement and prefetching part of data management. The scientific workflow is considered in most cases as the typical workload on distributed systems. However, many of the concepts created are independent of the workload being deployed.

In chapter 2, necessary background concepts and terminologies have been discussed, with a detailed reference to different important works in literature, This creates the foundation on which the novel works presented in the subsequent chapters can be understood.

In chapter 3, three independent works are presented which describe at length, the concepts to be used in later chapters in the form of two novel works. The concept of data interdependency between files in a distributed system is discussed, which will become important in chapters 4 and 5. This concept is discussed in conjunction with a problem with the classic definition and a solution to the problem. We have been able to describe that when replication is in use, additional information in this regard is required by the system for proper replica placement. Another related concept to interdependency has been discussed. It is the study on whether files should be placed as a whole as in traditional systems or by splitting them into fragments. It has been established in this work that placing files by fragmenting them instead of the present practice of placing the entire file is a better placement solution. The third work is aimed at extending a famous network simulation tool to study the performance of a distributed system in terms of simply network and storage. We have extended NS-3 simulator such that it is able to simulate network characteristics of the Hadoop Distributed File System.

In chapter 4, the cold start problem concerning data interdependency has been addressed. The solution lies in content based analysis. It has been shown that similarity based on topic in the content of text documents can be used for finding the file interdependency with quite good results.

In chapter 5, a data placement problem concerning prefetching is addressed, especially when dependency graphs are being used. The problem of identifying the insignificant dependencies which do not predict a files usage trend is addressed. For the solution, graph theoretic and social network analysis concept of edge betweenness centrality has been utilised. We have been able to obtain good results, when enough historic file usage data is available.

Finally, the problem of optimizing multiple parameters like lowering storage and network cost, while maintaining high data reliability, has been addressed. For this purpose metaheuristics based, statespace search has been used by hybridizing two algorithms to find an efficient solution to the problem. Through the empirical observations from the solution of the optimization problem, we have been able to prove that data placement is possible with low storage and network cost simultaneously while maintaining high reliability. Based on this finding, high reliability, low storage and network cost data placement scheme has been proposed for erasure coding based methods.

In summary, we have studied the problem of data placement in a distributed system under various conditions and have proposed appropriate solutions to the problems that came up from these conditions with varying amounts of success. While solution to many problems in the domain of distributed storage management has been addressed in this work, as discussed above, certain limitations do remain. In our work on simulation of the network characteristics of HDFS, the work is limited to HDFS only. A general purpose simulation solution would be more useful. Such a simulator could be extended into the specific distributed storage system other than HDFS. It has been generally shown that splitting a file into sub-components and placing them increases their localization; however, the splitting was made randomly which results in sub-optimality. In the work on localized placement of text files based on their topics, advanced insights obtained through the analytical power of LDA described in Chapter 4 has not been utilised to its full potential. Finally, the social network analysis based prefetching works when abundant data is available. The sparsity, while being an overall problem with data analytics, remains a severe limitation on the use of such approach in data placement in distributed systems.

We have plans to continue our research in data placement in distributed systems in future; part of which being to address the limitations discussed above. Source codes in NS-3 uses C++; hence, allowing for the use of object oriented paradigm's concept of inheritance. We plan to build a general purpose distributed storage simulator in NS3. The specific storage system, like HDFS, which needs to be simulated can be extended by the new classes inheriting the general purpose classes of the to be built simulator for the distributed storage system. We plan to develop an intelligent system which can split and place the fragments of the file that has been split in the distributed system for maximum performance gain through localization. In relation to the data sparsity problem, we plan to generate additional inter-dependency data between the files through the use of provenance analysis of data. We further hope to obtain better text file placement through content analysis through advanced text mining techniques. Hopefully, these and other similar efforts in future will further improve the state-of-the-art in data placement in distributed systems, a goal that we set out while working on the different works described in this thesis.

Bibliography

[1]A. Rowstron and P. Druschel, “Storage management and caching in past, a large-scale, persistent peer-to-peer storage utility,” in *Proc. 8th ACM Symposium on Operating Systems Principles*, pp. 188–201, October 21-24, 2001.

[2]F. Dabek, M. F. Kaashoek, D. Karger, R. Morris, and I. Stoica, “Wide-area cooperative storage with CFS,” *ACM SIGOPS Operating Systems Review*, vol. 35, no. 5, pp. 202–215, 2001.

[3]B. Pawlowski, C. Juszczak, P. Staubach, C. Smith, D. Lebel, and D. Hitz, “NFS version 3: Design and implementation.,” in *Proc. Summer USENIX Conference*, pp. 137–152, June 6-10, 1994.

[4]R. Sandberg, D. Goldberg, S. Kleiman, D. Walsh, and B. Lyon, “Design and implementation of the sun network filesystem,” in *Proc. Summer USENIX Conference*, pp. 119–130, June 11-14, 1985.

[5]A. Muthitacharoen, R. Morris, T. M. Gil, and B. Chen, “Ivy: A read/write Peer-to-Peer file system,” *ACM SIGOPS Operating Systems Review*, vol. 36, no. SI, pp. 31–44, 2002.

[6]B. Wilcox-O’Hearn, “Experiences deploying a large-scale emergent network,” in *International Workshop on Peer-to-Peer Systems*, February 24-25 2002.

[7]B. Cohen, “Incentives build robustness in bittorrent,” in *Workshop on Economics of Peer-to-Peer Systems*, June 4-5, 2003.

[8]P. J. Braam and P. Schwan, “Lustre: The intergalactic file system,” in *Ottawa Linux Symposium*, pp. 50–54, July 13-16, 2002.

[9]P. Braam, “The lustre storage architecture,” arXiv, 2019.

[10]F. Schmuck and R. Haskin, "[gpfs]: A shared-disk file system for large computing clusters," in *Proc. Conference on File and Storage Technologies*, pp. 231–244, January 28-30, 2002.

[11]S. S. Vazhkudai, X. Ma, V. W. Freeh, J. W. Strickland, N. Tammineedi, and S. L. Scott, "Freeloader: Scavenging desktop storage resources for scientific data," in *Proc. ACM/IEEE Conference on Supercomputing*, pp. 56–66, November 12-18, 2005.

[12]J. Kubiatowicz, D. Bindel, Y. Chen, S. Czerwinski, P. Eaton, D. Geels, R. Gummadi, S. Rhea, H. Weatherspoon, W. Weimer, *et al.*, "Oceanstore: An architecture for global-scale persistent storage," *ACM SIGOPS Operating Systems Review*, vol. 34, no. 5, pp. 190–201, 2000.

[13]S. Ghemawat, H. Gobioff, and S.-T. Leung, "The Google File System," in *19th ACM Symposium on Operating Systems Principles*, pp. 29–43, October 23-26, 2003.

[14]R. Hasan, Z. Anwar, W. Yurcik, L. Brumbaugh, and R. Campbell, "A survey of Peer-to-Peer storage techniques for distributed file systems," in *Proc. International Conference on Information Technology: Coding and Computing*, pp. 205–213, April 4-6, 2005.

[15]A. Oram, *Peer-to-Peer: Harnessing the power of disruptive technologies.* California, USA: O'Reilly Media, Inc., 2001.

[16]B. Carlsson and R. Gustavsson, "The rise and fall of Napster-An evolutionary approach," in *Proc. 6th International Computer Science Conference on Active Media Technology*, pp. 347–354, December 18-20, 2001.

[17]M. Ripeanu, "Peer-to-Peer architecture case study: Gnutella network," in *Proc. 1st International Conference on Peer-to-Peer Computing*, pp. 99–100, August 27-29, 2001.

[18]S. Adler, "The slashdot effect: An analysis of three internet publications," *Linux Gazette*, vol. 38, no. 2, pp. 623–626, 1999.

[19]D. Hughes, G. Coulson, and J. Walkerdine, "Free riding on gnutella revisited: the bell tolls?," *IEEE Distributed Systems*, vol. 6, no. 6, pp. 1–18, 2005.

[20]M. Yang, Z. Zhang, X. Li, and Y. Dai, "An empirical study of free-riding behavior in the Maze P2P file-sharing system," in *Proc. International Workshop on Peer-to-Peer Systems*, pp. 182–192, February 24-25, 2005.

[21]W. J. Bolosky, J. R. Douceur, D. Ely, and M. Theimer, "Feasibility of a serverless distributed file system deployed on an existing set of desktop PCs," *ACM SIGMETRICS Performance Evaluation Review*, vol. 28, no. 1, pp. 34–43, 2000.

[22]A. Adya, W. J. Bolosky, M. Castro, G. Cermak, R. Chaiken, J. R. Douceur, J. Howell, J. R. Lorch, M. Theimer, and R. P. Wattenhofer, "Farsite: Federated, available, and reliable storage for an incompletely trusted environment," *ACM SIGOPS Operating Systems Review*, vol. 36, no. SI, pp. 1–14, 2002.

[23]P. E. Crandall, R. A. Aydt, A. A. Chien, and D. A. Reed, "Input/output characteristics of scalable parallel applications," in *Proc. ACM/IEEE conference on Supercomputing*, pp. 59–89, November 12-17, 1995.

[24]I. Foster, C. Kesselman, and S. Tuecke, "The anatomy of the grid: Enabling scalable virtual organizations," *The International Journal of High Performance Computing Applications*, vol. 15, no. 3, pp. 200–222, 2001.

[25]A. Chervenak, I. Foster, C. Kesselman, C. Salisbury, and S. Tuecke, "The data grid: Towards an architecture for the distributed management and analysis of large scientific datasets," *Journal of Network and Computer Applications*, vol. 23, no. 3, pp. 187–200, 2000.

[26]A. Rajasekar, M. Wan, and R. Moore, "MySRB and SRB-components of a data grid," in *Proc. 11th IEEE International Symposium on High Performance Distributed Computing*, pp. 301–310, July 24-26, 2002.

[27]E. Deelman and Y. Gil, "NSF workshop on the challenges of scientific workflows," pp. 1–2, May 1-2, 2006.

[28]T. Oinn, M. Addis, J. Ferris, D. Marvin, M. Senger, M. Greenwood, T. Carver, K. Glover, M. R. Pocock, A. Wipat, *et al.*, "Taverna: A tool for the composition and enactment of bioinformatics workflows," *Bioinformatics*, vol. 20, no. 17, pp. 3045–3054, 2004.

[29]B. Ludscher, I. Altintas, C. Berkley, D. Higgins, E. Jaeger, M. Jones, E. A. Lee, J. Tao, and Y. Zhao, "Scientific workflow management and the Kepler system," *Concurrency and Computation: Practice and Experience*, vol. 18, no. 10, pp. 1039–1065, 2006.

[30]I. Taylor, M. Shields, and I. Wang, "Distributed P2P computing within TGriana: A galaxy visualization test case," in *Proc. International Parallel and Distributed Processing Symposium*, pp. 8–18, May 15-19, 2003.

[31]E. Deelman, G. Singh, M.-H. Su, J. Blythe, Y. Gil, C. Kesselman, G. Mehta, K. Vahi, G. B. Berriman, J. Good, *et al.*, "Pegasus: A framework for mapping complex scientific workflows onto distributed systems," *Scientific Programming*, vol. 13, no. 3, pp. 219–237, 2005.

[32]P. Mell, T. Grance, *et al.*, "The NIST definition of cloud computing." url: https://nvlpubs.nist.gov/nistpubs/legacy/sp/nistspecialpublication800-145.pdf. (accessed 2022-03-08).

[33]I. Foster, Y. Zhao, I. Raicu, and S. Lu, "Cloud computing and grid computing 360-degree compared," in *Proc. Grid computing Environments Workshop*, pp. 1–10, November 12-16, 2008.

[34]D. E. Williams, "Virtualization with Xen (TM): Including XenEnterprise, XenServer, and XenExpress," Amsterdam, Netherlands: Elsevier, 2007.

[35]Y. Goto, "Kernel-based virtual machine technology," *Fujitsu Scientific and Technical Journal*, vol. 47, no. 3, pp. 362–368, 2011.

[36] A. Gulati, A. Holler, M. Ji, G. Shanmuganathan, C. Waldspurger, and X. Zhu, "VMware distributed resource management: Design, implementation, and lessons learned," *VMware Technical Journal*, vol. 1, no. 1, pp. 45–64, 2012.

[37] A. Zahariev, "Google App Engine." url: http://cse.tkk.fi/en/publications/B/5/papers/Zahariev_final.pdf. (accessed 2022-03-08).

[38] "Amazon Web Services." url http://aws. amazon. com/es/ec2. (accessed 2022-03-08).

[39] D. Liu, H. Zhu, and I. Bayley, "Applying algebraic specification to cloud computing–A case study of Infrastructure-As-A-Service gogrid," in *Proc. of the 7th International Conference on Software Engineering Advances*, pp. 1–4, November 18-23 2012.

[40] G. Munasinghe and P. Anderson, "Flexiscale-next generation data centre management." url: http://spring2008.ukuug.org/talk_abstracts.html. (accessed: 2022-03-08).

[41] M. Copeland, J. Soh, A. Puca, M. Manning, and D. Gollob, *Microsoft Azure.* New York, USA: Apress, 2015.

[42] A. Manchar and A. Chouhan, "Salesforce CRM: A new way of managing customer relationship in cloud environment," in *Proc. 2nd International Conference on Electrical, Computer and Communication Technologies*, pp. 1–4, June 26-27 2017.

[43] Y. Zhou, "SAP business by design," in *Proc. IEEE 25th International Conference on Data Engineering*, pp. 1760–1760, March 29-April 2, 2009.

[44] D. Borthakur *et al.*, "HDFS architecture guide," *Hadoop apache project*, vol. 53, pp. 1–14, 2008.

[45] A. Yousefpour, C. Fung, T. Nguyen, K. Kadiyala, F. Jalali, A. Niakanlahiji, J. Kong, and J. P. Jue, "All one needs to know about fog computing and related edge computing paradigms: A complete survey," *Journal of Systems Architecture*, vol. 98, pp. 289–330, 2019.

[46] J. Gedeon, F. Brandherm, R. Egert, T. Grube, and M. Muhlhauser, "What the fog? Edge computing revisited: Promises, applications and future challenges," *IEEE Access*, vol. 7, pp. 152847–152878, 2019.

[47] A. J. Ferrer, J. M. Marques, and J. Jorba, "Towards the decentralised cloud: Survey on approaches and challenges for [m]obile, [a]d hoc, and [e]dge computing," *ACM Computing Surveys (CSUR)*, vol. 51, no. 6, pp. 1–36, 2019.

[48] B. Zaghdoudi, H. K.-B. Ayed, and I. Riabi, "Ad hoc cloud as a service: A protocol for setting up an ad hoc cloud over manets," *Procedia Computer Science*, vol. 56, pp. 573–579, 2015.

[49] N. Jones, "How to stop data centres from gobbling up the world's electricity?," *Nature*, vol. 561, no. 7722, pp. 163–167, 2018.

[50] P. Domingues, P. Marques, and L. Silva, "Resource usage of Windows computer laboratories," in *Proc. International Conference on Parallel Processing Workshops*, pp. 469–476, June 14-17, 2005.

[51] A. Chien, B. Calder, S. Elbert, and K. Bhatia, "Entropia: Architecture and performance of an enterprise desktop grid system," *Journal of Parallel and Distributed Computing*, vol. 63, no. 5, pp. 597–610, 2003.

[52] M. J. Litzkow, M. Livny, and M. W. Mutka, "Condor-A hunter of idle workstations." url: https://research.cs.wisc.edu/htcondor/doc/icdcs1988.pdf. (accessed: 2022-03-08).

[53] D. P. Anderson, "BOINC: A system for public-resource computing and storage," in *Proc. 5th IEEE/ACM International Workshop on Grid Computing*, pp. 4–10, November 8, 2004.

[54] J. Barranco, Y. Cai, D. Cameron, M. Crouch, R. De Maria, L. Field, M. Giovannozzi, P. Hermes, N. Hoimyr, D. Kaltchev, *et al.*, "LHC@ Home: A BOINC-based volunteer computing infrastructure for physics studies at CERN," *Open Engineering*, vol. 7, no. 1, pp. 379–393, 2017.

[55]G. A. McGilvary, A. Barker, A. Lloyd, and M. Atkinson, "V-BOINC: The virtualization of Boinc," in *Proc. 13th IEEE/ACM International Symposium on Cluster, Cloud, and Grid Computing*, pp. 285–293, May 13-16, 2013.

[56]K. Graffi, D. Stingl, C. Gross, H. Nguyen, A. Kovacevic, and R. Steinmetz, "Towards a P2P cloud: Reliable resource reservations in unreliable P2P systems," in *Proc. IEEE 16th International Conference on Parallel and Distributed Systems*, pp. 27–34, December 8-10, 2010.

[57]K. Chard, S. Caton, O. Rana, and K. Bubendorfer, "Social cloud: Cloud computing in social networks," in *Proc. IEEE 3rd International Conference on Cloud Computing*, pp. 99–106, October 24-26, 2010.

[58]D. Neumann, C. Bodenstein, O. F. Rana, and R. Krishnaswamy, "STACEE: Enhancing storage clouds using edge devices," in *Proc. 1st ACM/IEEE Workshop on Autonomic Computing in Economics*, pp. 19–26, June 14, 2011.

[59]L. Wang, J. Tao, M. Kunze, D. Rattu, and A. C. Castellanos, "The Cumulus project: Build a scientific cloud for a data center," in *Proc. 1st workshop on Cloud Computing and its Applications*, pp. 1–7, October 22-23, 2008.

[60]D. Nurmi, R. Wolski, C. Grzegorczyk, G. Obertelli, S. Soman, L. Youseff, and D. Zagorodnov, "The Eucalyptus open-source cloud-computing system," in *Proc. 9th IEEE/ACM International Symposium on Cluster Computing and the Grid*, pp. 124–131, May 18-21, 2009.

[61]B. Sotomayor, R. S. Montero, I. M. Llorente, and I. Foster, "Capacity leasing in cloud systems using the open nebula engine." url: https://citeseerx.ist.psu.edu/document?repid=rep1type=pdf doi=35451f05ae33a8d3160dd3b11dada3968ce99566. (accessed 2022-03-08).

[62]E. Pinheiro, W.-D. Weber, and L. A. Barroso, "Failure trends in a large disk drive population," in *Proc. 5th USENIX Conference*

on File and Storage Technologies, pp. 17–29, February 13-16, 2007.

[63]D. Ford, F. Labelle, F. Popovici, M. Stokely, V.-A. Truong, L. Barroso, C. Grimes, and S. Quinlan, "Availability in globally distributed storage systems," in *Proc. 9th USENIX Conference on Operating Systems Design and Implementation*, pp. 61–74, October 4-6, 2010.

[64]S. Lu, B. Luo, T. Patel, Y. Yao, D. Tiwari, and W. Shi, "Making disk failure predictions smarter!," in *Proc. 18th USENIX Conference on File and Storage Technologies*, pp. 151–167, February 24-27, 2020.

[65]S. Kadekodi, K. Rashmi, and G. R. Ganger, "Cluster storage systems gotta have heart: improving storage efficiency by exploiting disk-reliability heterogeneity," in *Proc. 17th USENIX Conference on File and Storage Technologies*, pp. 345–358, February 25-28, 2019.

[66]J. Basak and R. H. Katz, "Significance of disk failure prediction in datacenters," arXiv, 2017.

[67]S. S. Arslan, "A reliability model for dependent and distributed mds disk array units," *IEEE Transactions on Reliability*, vol. 68, no. 1, pp. 133–148, 2018.

[68]G. Wang, L. Zhang, and W. Xu, "What can we learn from four years of data center hardware failures?," in *Proc. 47th Annual IEEE/IFIP International Conference on Dependable Systems and Networks*, pp. 25–36, June 26-29, 2017.

[69]Q. Lin, K. Hsieh, Y. Dang, H. Zhang, K. Sui, Y. Xu, J.-G. Lou, C. Li, Y. Wu, R. Yao, *et al.*, "Predicting node failure in cloud service systems," in *Proc. 26th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, pp. 480–490, November 4-9, 2018.

[70]C. Huang, H. Simitci, Y. Xu, A. Ogus, B. Calder, P. Gopalan, J. Li, and S. Yekhanin, "Erasure coding in windows azure storage," in *Proc. USENIX Annual Technical Conference*, pp. 15–26, June 13-15 2012.

[71] D. Beaver, S. Kumar, H. C. Li, J. Sobel, P. Vajgel, *et al.*, "Finding a needle in haystack: Facebook's photo storage.," in *Proc. 9th USENIX Symposium on Operating Systems Design and Implementations*, pp. 1–8, October 4-6, 2010.

[72] C. Huang, M. Chen, and J. Li, "Pyramid codes: Flexible schemes to trade space for access efficiency in reliable data storage systems," *ACM Transactions on Storage*, vol. 9, no. 1, pp. 1–28, 2013.

[73] C. Suh and K. Ramchandran, "Exact-repair mds code construction using interference alignment," *IEEE Transactions on Information Theory*, vol. 57, no. 3, pp. 1425–1442, 2011.

[74] K. V. Rashmi, N. B. Shah, and P. V. Kumar, "Optimal exact-regenerating codes for distributed storage at the msr and mbr points via a product-matrix construction," *IEEE Transactions on Information Theory*, vol. 57, no. 8, pp. 5227–5239, 2011.

[75] S.-J. Lin and W.-H. Chung, "Novel repair-by-transfer codes and systematic exact-mbr codes with lower complexities and smaller field sizes," *IEEE Transactions on Parallel and Distributed Systems*, vol. 25, no. 12, pp. 3232–3241, 2014.

[76] D. Leong, A. G. Dimakis, and T. Ho, "Distributed storage allocations," *IEEE Transactions on Information Theory*, vol. 58, no. 7, pp. 4733–4752, 2012.

[77] D. Leong, A. G. Dimakis, and T. Ho, "Distributed storage allocation for high reliability," in *Proc. IEEE International Conference on Communications*, pp. 1–6, May 23-27, 2010.

[78] D. Leong, A. G. Dimakis, and T. Ho, "Distributed storage allocation problems," in *Proc. Workshop on Network Coding, Theory, and Applications*, pp. 86–91, June 15-16, 2009.

[79] J. Liu, H. Shen, H. Chi, H. S. Narman, Y. Yang, L. Cheng, and W. Chung, "A low-cost multi-failure resilient replication scheme for high-data availability in cloud storage," *IEEE/ACM Transactions on Networking*, vol. 29, no. 4, pp. 1436–1451, 2020.

[80]S.-Q. Long, Y.-L. Zhao, and W. Chen, “Morm: A multi-objective optimized replication management strategy for cloud storage cluster,” *Journal of Systems Architecture*, vol. 60, no. 2, pp. 234–244, 2014.

[81]D. Boru, D. Kliazovich, F. Granelli, P. Bouvry, and A. Y. Zomaya, “Energy-efficient data replication in cloud computing datacenters,” *Cluster Computing*, vol. 18, no. 1, pp. 385–402, 2015.

[82]M.-K. Hussein and M.-H. Mousa, “A light-weight data replication for cloud data centers environment,” *International Journal of Engineering and Innovative Technology*, vol. 1, no. 6, pp. 169–175, 2012.

[83]D. Yuan, Y. Yang, X. Liu, and J. Chen, “A data placement strategy in scientific cloud workflows,” *Future Generation Computer Systems*, vol. 26, no. 8, pp. 1200–1214, 2010.

[84]M. Y. Eltabakh, Y. Tian, F. Ozcan, R. Gemulla, A. Krettek, and J. McPherson, “Co-Hadoop: flexible data placement and its exploitation in hadoop,” *Proc. VLDB Endowment*, vol. 4, no. 9, pp. 575–585, 2011.

[85]J.-x. Wu, C.-s. Zhang, B. Zhang, and P. Wang, “A new data-grouping-aware dynamic data placement method that takes into account jobs execute frequency for hadoop,” *Microprocessors and Microsystems*, vol. 47, pp. 161–169, 2016.

[86]J. Wang, P. Shang, and J. Yin, “Draw: A new data-grouping-aware data placement scheme for data intensive applications with interest locality,” in *Cloud Computing for Data-Intensive Applications*, pp. 149–174, Springer, 2014.

[87]N. Mansouri and M. M. Javidi, “A new prefetching-aware data replication to decrease access latency in cloud environment,” *Journal of Systems and Software*, vol. 144, pp. 197–215, 2018.

[88]K. A. Kumar, A. Quamar, A. Deshpande, and S. Khuller, “SWORD: Workload-aware data placement and replica selection for cloud data management systems,” *The VLDB Journal*, vol. 23, no. 6, pp. 845–870, 2014.

[89]N. Mansouri, M. M. Javidi, and B. Mohammad Hasani Zade, "Using data mining techniques to improve replica management in cloud environment," *Soft Computing*, vol. 24, no. 10, pp. 7335–7360, 2020.

[90]T. Hamrouni, S. Slimani, and F. B. Charrada, "A data mining correlated patterns-based periodic decentralized replication strategy for data grids," *Journal of Systems and Software*, vol. 110, pp. 10–27, 2015.

[91]S. Tong and D. Koller, "Support vector machine active learning with applications to text classification," *Journal of Machine Learning Research*, vol. 2, pp. 45–66, 2001.

[92]G. M. Fung and O. L. Mangasarian, "Multicategory proximal support vector machine classifiers," *Machine learning*, vol. 59, no. 1-2, pp. 77–97, 2005.

[93]G. Ingersoll, T. S. Morton, and D. Farris, "Taming text: how to find, organize, and manipulate it," New York, USA: Simon and Schuster, 2012.

[94]D. Antons and C. F. Breidbach, "Big data, big insights? Advancing service innovation and design with machine learning," *Journal of Service Research*, vol. 21, no. 1, pp. 17–39, 2018.

[95]C. Hopp, D. Antons, J. Kaminski, and T. O. Salge, "The topic landscape of disruption research—a call for consolidation, reconciliation, and generalization," *Journal of Product Innovation Management*, vol. 35, no. 3, pp. 458–487, 2018.

[96]S. Kaplan and K. Vakili, "The double-edged sword of recombination in breakthrough innovation," *Strategic Management Journal*, vol. 36, no. 10, pp. 1435–1457, 2015.

[97]D. Antons, A. M. Joshi, and T. O. Salge, "Content, contribution, and knowledge consumption: Uncovering hidden topic structure and rhetorical signals in scientific texts," *Journal of Management*, vol. 45, no. 7, pp. 3035–3076, 2019.

[98] R. Agrawal, R. Srikant, *et al.*, "Fast algorithms for mining association rules," in *Proc. 20th International Conference on Very Large Data Bases*, pp. 487–499, September 12-15, 1994.

[99] N. D. Almalis, G. A. Tsihrintzis, N. Karagiannis, and A. D. Strati, "FoDRA—a new content-based job recommendation algorithm for job seeking and recruiting," in *Proc. 6th International Conference on Information, Intelligence, Systems and Applications*, pp. 1–7, July 6-8, 2015.

[100] M. Amami, R. Faiz, F. Stella, and G. Pasi, "A graph based approach to scientific paper recommendation," in *Proc. international conference on web intelligence*, pp. 777–782, August 23-26, 2017.

[101] C. Wang and D. M. Blei, "Collaborative topic modeling for recommending scientific articles," in *Proc. 17th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pp. 448–456, August 21 - 24, 2011.

[102] D. Ofer, N. Brandes, and M. Linial, "The language of proteins: NLP, machine learning & protein sequences," *Computational and Structural Biotechnology Journal*, vol. 19, pp. 1750–1758, 2021.

[103] X. Chen, X. Hu, X. Shen, and G. Rosen, "Probabilistic topic modeling for genomic data interpretation," in *Proc. IEEE International Conference on Bioinformatics and Biomedicine*, pp. 149–152, December 18-21, 2010.

[104] F. Glover, "Future paths for integer programming and links to artificial intelligence," *Computers & Operations Research*, vol. 13, no. 5, pp. 533–549, 1986.

[105] E. Goldberg David and H. Henry, "Genetic algorithms and machine learning," *Machine Learning*, vol. 3, no. 2, pp. 95–99, 1988.

[106] M. Dorigo, M. Birattari, and T. Stutzle, "Ant colony optimization," *IEEE Computational Intelligence Magazine*, vol. 1, no. 4, pp. 28–39, 2006.

[107] J. Kennedy and R. Eberhart, “Particle swarm optimization,” in *Proc. International Conference on Neural Networks*, pp. 1942–1948, November 27-December 1, 1995.

[108] T. Mekhaznia and M. E. B. Menai, “Cryptanalysis of classical ciphers with ant algorithms,” *International Journal of Metaheuristics*, vol. 3, no. 3, pp. 175–198, 2014.

[109] M. Angelova and T. Pencheva, “Genetic operators’ significance assessment in multi-population genetic algorithms,” *International Journal of Metaheuristics*, vol. 3, no. 2, pp. 162–173, 2014.

[110] A. Aliano Filho, H. de Oliveira Florentino, and M. Vaz Pato, “Metaheuristics for a crop rotation problem,” *International Journal of Metaheuristics*, vol. 3, no. 3, pp. 199–222, 2014.

[111] S. Fidanova, P. Marinov, and M. Paparzycki, “Multi-objective ACO algorithm for wsn layout: performance according to number of ants,” *International Journal of Metaheuristics*, vol. 3, no. 2, pp. 149–161, 2014.

[112] J. Bobadilla, F. Ortega, A. Hernando, and A. Gutierrez, “Recommender systems survey,” *Knowledge-Based Systems*, vol. 46, pp. 109–132, 2013.

[113] A. Sang and S. K. Vishwakarma, “A ranking based recommender system for cold start & data sparsity problem,” in *Proc. 10th International Conference on Contemporary Computing*, pp. 1–3, August 10-12, 2017.

[114] R. Obeidat, R. Duwairi, and A. Al-Aiad, “A collaborative recommendation system for online courses recommendations,” in *Proc. International Conference on Deep Learning and Machine Learning in Emerging Applications*, pp. 49–54, August 26-28 2019.

[115] N. Nassar, A. Jafar, and Y. Rahhal, “A novel deep multi-criteria collaborative filtering model for recommendation system,” *Knowledge-Based Systems*, vol. 187, pp. 104–121, 2020.

[116] W. Krisdhamara, B. Pharmasetiawan, and K. Mutijarsa, “Improvement of collaborative filtering recommendation system to

resolve sparsity problem using combination of clustering and opinion mining methods," in *International Seminar on Application for Technology of Information and Communication*, pp. 94–99, September 21-22, 2019.

[117]K. K. Fletcher, "A method for dealing with data sparsity and cold-start limitations in service recommendation using personalized preferences," in *Proc. IEEE International Conference on Cognitive Computing*, pp. 72–79, June 25-30, 2017.

[118]C. L. Streeter and D. F. Gillespie, "Social network analysis," *Journal of Social Service Research*, vol. 16, pp. 201–222, 1993.

[119]S. Wasserman, K. Faust, *et al.*, "Social network analysis: Methods and applications," Cambridge, UK: Cambridge university press, 1994.

[120]J. Coleman, "The mathematics of collective action," Abingdon-on-Thames, UK: Routledge, 2017.

[121]R. Burt, "Toward a structural theory of action," New York, USA: Academic Press, 1982.

[122]P. Bonacich, "Factoring and weighting approaches to status scores and clique identification," *Journal of Mathematical Sociology*, vol. 2, no. 1, pp. 113–120, 1972.

[123]L. C. Freeman, "Centrality in social networks conceptual clarification," *Social Networks*, vol. 1, no. 3, pp. 215–239, 1978.

[124]J. Zhang and Y. Luo, "Degree centrality, betweenness centrality, and closeness centrality in social network," in *Proc. 2nd International Conference on Modelling, Simulation and Applied Mathematics*, pp. 300–303, March 26-27, 2017.

[125]S. Brin and L. Page, "The anatomy of a large-scale hypertextual web search engine," *Computer Networks and ISDN Systems*, vol. 30, no. 1-7, pp. 107–117, 1998.

[126]S. Sukrat and B. Papasratorn, "An architectural framework for developing a recommendation system to enhance vendors' capability in c2c social commerce," *Social Network Analysis and Mining*, vol. 8, no. 1, pp. 1–13, 2018.

[127] D. H. Lee and P. Brusilovsky, "Improving personalized recommendations using community membership information," *Information Processing & Management*, vol. 53, no. 5, pp. 1201–1214, 2017.

[128] A. Triantafillidou and G. Siomkos, "The impact of facebook experience on consumers' behavioral brand engagement," *Journal of Research in Interactive Marketing*, vol. 12, no. 2, pp. 164–192, 2018.

[129] A. Corbellini, C. Mateos, D. Godoy, A. Zunino, and S. Schiaffino, "An architecture and platform for developing distributed recommendation algorithms on large-scale social networks," *Journal of Information Science*, vol. 41, no. 5, pp. 686–704, 2015.

[130] R. N. Calheiros, R. Ranjan, A. Beloglazov, C. A. De Rose, and R. Buyya, "Cloudsim: a toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms," *Software: Practice and Experience*, vol. 41, no. 1, pp. 23–50, 2011.

[131] R. Buyya and M. Murshed, "Gridsim: A toolkit for the modeling and simulation of distributed resource management and scheduling for grid computing," *Concurrency and Computation: Practice and Experience*, vol. 14, no. 13-15, pp. 1175–1220, 2002.

[132] T. Issariyakul and E. Hossain, "Introduction to network simulator 2 (NS2)," in *Introduction to network simulator NS2*, pp. 1–18, New York, USA: Springer, 2009.

[133] T. R. Henderson, M. Lacage, G. F. Riley, C. Dowell, and J. Kopena, "Network simulations with the ns-3 simulator," *SIGCOMM Demonstration*, vol. 14, no. 14, p. 527, 2008.

[134] H. Rinne, "The Weibull distribution: a handbook," New York, USA: CRC press, 2008.

[135] T. White, "Hadoop: The definitive guide," California, USA: "O'Reilly Media, Inc.", 2012.

[136] W. McCormick, S. Deutsch, J. Martin, and P. Schweitzer, "Identification of data structures and

relationships by matrix reordering techniques.” url-https://apps.dtic.mil/sti/tr/pdf/AD0754012.pdf. (accessed 2022-03-08).

[137]W. T. McCormick Jr, P. J. Schweitzer, and T. W. White, “Problem decomposition and data reorganization by a clustering technique,” *Operations Research*, vol. 20, no. 5, pp. 993–1009, 1972.

[138]X. Liu, D. Yuan, G. Zhang, J. Chen, and Y. Yang, “SwinDeW-C: A‘ peer-to-peer based cloud workflow system,” in *Handbook of Cloud Computing*, pp. 309–332, New York, USA: Springer, 2010.

[139]H. Bhattacharya, S. Chattopadhyay, and M. Chattopadhyay, “Problems with replica placement using data dependency in scientific cloud workflow,” in *Proc. 5th International Conference on Emerging Applications of Information Technology*, pp. 1–4, January 12-13, 2018.

[140]H. Bhattacharya, M. Chattopadhyay, and S. Chattopadhay, “A case for splitting a file for data placement in a distributed scientific workflow,” in *Proc. IEEE 12th Annual Information Technology, Electronics and Mobile Communication Conference*, pp. 1058–1063, October 27-30, 2021.

[141]H. Bhattacharya, S. Chattopadhyay, and M. Chattopadhyay, “NS3 Based HDFS data placement algorithm evaluation framework,” in *Proc. International Conference on Computer, Electrical & Communication Engineering*, pp. 1–8, December 22-23, 2017.

[142]A. S. Foundation, “HDFS logical cluster architecture.” url: https://hadoop.apache.org/docs/stable/hadoop-project-dist/hadoop-hdfs/HdfsDesign.html. (accessed 2022-03-08).

[143]H. Bhattacharya, A. Bhattacharya, S. Chattopadhyay, and M. Chattopadhyay, “LDA topic modeling based dataset dependency matrix prediction,” in *Proc. International Conference on Computational Intelligence, Communications, and Business Analytics*, pp. 54–69, July 27-28, 2018.

[144]D. M. Blei, A. Y. Ng, and M. I. Jordan, “Latent dirichlet allocation,” *The Journal of Machine Learning Research*, vol. 3, pp. 993–1022, 2003.

[145] I. Porteous, D. Newman, A. Ihler, A. Asuncion, P. Smyth, and M. Welling, "Fast collapsed gibbs sampling for latent dirichlet allocation," in *Proc. 14th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pp. 569–577, August 24 - 27, 2008.

[146] Q. Le and T. Mikolov, "Distributed representations of sentences and documents," in *Proc. International Conference on Machine Learning*, pp. 1188–1196, June 21-26, 2014.

[147] J. D. M.-W. C. Kenton and L. K. Toutanova, "BERT: Pre-training of deep bidirectional transformers for language understanding," in *Proc. of NAACL-HLT*, pp. 4171–4186, June 2-7, 2019.

[148] G. Maheshwari, P. Trivedi, H. Sahijwani, K. Jha, S. Dasgupta, and J. Lehmann, "Simdoc: Topic sequence alignment based document similarity framework," in *Proc. Knowledge Capture Conference*, pp. 1–8, December 2-3, 2017.

[149] V. Rus, N. Niraula, and R. Banjade, "Similarity measures based on latent dirichlet allocation," in *Proc. International Conference on Intelligent Text Processing and Computational Linguistics*, pp. 459–470, March 24-30, 2013.

[150] D. Greene and P. Cunningham, "Practical solutions to the problem of diagonal dominance in kernel document clustering," in *Proc. 23rd International Conference on Machine Learning*, pp. 377–384, June 25 - 29, 2006.

[151] H. Bhattacharya, A. Bhattacharya, M. Chattopadhyay, and S. Chattopadhyay, "Determining threshold for partitioning a dependency graph in replica prefetching in distributed systems," in *Proc. 6th International Conference on Research in Computational Intelligence and Communication Networks*, p. Article in Press, Springer, 2021.

[152] M. Girvan and M. E. Newman, "Community structure in social and biological networks," *Proc. National Academy of Sciences*, vol. 99, no. 12, pp. 7821–7826, 2002.

[153]B. Leem and H. Chun, "An impact of online recommendation network on demand," *Expert Systems with Applications*, vol. 41, no. 4, pp. 1723–1729, 2014.

[154]I. Pietri and R. Sakellariou, "Scheduling data-intensive scientific workflows with reduced communication," in *Proc. 30th International Conference on Scientific and Statistical Database Management*, pp. 1–4, July 9-11, 2018.

[155]H. Bhattacharya, S. Chattopadhyay, M. Chattopadhyay, and A. Banerjee, "Storage and bandwidth optimized reliable distributed data allocation algorithm," *International Journal of Ambient Computing and Intelligence*, vol. 10, no. 1, pp. 78–95, 2019.

[156]K. Bilal, S. U. Khan, L. Zhang, H. Li, K. Hayat, S. A. Madani, N. Min-Allah, L. Wang, D. Chen, M. Iqbal, *et al.*, "Quantitative comparisons of the state-of-the-art data center architectures," *Concurrency and Computation: Practice and Experience*, vol. 25, no. 12, pp. 1771–1783, 2013.

[157]D. Wong, K. T. Seow, C. H. Foh, and R. Kanagavelu, "Towards reproducible performance studies of datacenter network architectures using an open-source simulation approach," in *Proc. IEEE Global Communications Conference*, pp. 1373–1378, December 9-13, 2013.

[158]L. Sahoo, A. Banerjee, A. K. Bhunia, and S. Chattopadhyay, "An efficient GA-PSO approach for solving mixed-integer nonlinear programming problem in reliability optimization," *Swarm and Evolutionary Computation*, vol. 19, pp. 43–51, 2014.

Hindol Bhattacharya