A MULTIMODEL CLOUD DATA STORAGE SYSTEM HAVING AN OBJECT BASED VIEW

Thesis Submitted by

Anindita Sarkar Mondal

DOCTOR OF PHILOSOPHY (ENGINEERING)

Department of Information Technology
Faculty Council of Engineering & Technology
Jadavpur University
Kolkata, India

2023

Supervisors Details:

Name: Prof. Samiran Chattapadhyay
Designation: Professor
Institution of the Supervisor:
Department of Information Technology
Jadavpur University, Salt Lake Campus
E-mail samiranc.ju@gmail.com
Kolkata-700106
West Bengal
India

AND

Name: Prof. Anirban Mukhopadhyay
Designation: Professor
Institution of the Supervisor:
Department of Computer Science and Engineering,
University of Kalyani, Kalyani, Nadia
E-mail anirbanbuba@yahoo.com
Kalyani-741235
West Bengal
India

List of Publications

Journals

1. Anindita Sarkar Mondal, Anirban Mukhopadhyay, Samiran Chattopadhyay, "Machine Learning-driven Automatic Storage Space Recommendation for Object-based Cloud Storage System", Complex & Intelligent System Journal, DOI: 10.1007/s40747-021-00517-4, 2021

2. Anindita Sarkar Mondal, Somnath Mukhopadhyay, Kartick Chandra Mondal, Samiran Chattopadhyay, "A Double Threshold Based Power Aware Honey Bee Cloud Load Balancing Algorithm", SN Computer Science Journal, Volume: 2, Article number: 395, 2021.

3. Anindita Sarkar Mondal, Sarmistha Neogy, Nandini Mukherjee, Samiran Chattopadhyay, "Performance analysis of an efficient object-based schema oriented data storage system handling health data." Innovations System Software Engineering (ISSE), Vol: 16, Issue: 1, Pages: 63-77, 2020.

4. Anindita Sarkar Mondal, Sarmistha Neogy, Nandini Mukherjee, and Samiran Chattopadhyay, "A survey of issues and solutions of health data management systems." Innovations Syst Softw Eng (ISSE), Vol: 15, Pages: 155–166, 2019, doi:10.1007/s11334-019-00336-4

5. Anindita Sarkar Mondal, Madhupa Sanyal, Samiran Chattopadhyay, and Kartick Chandra Mondal, "Performance Analysis of Structured, Un-structured & Cloud Storage Systems", International Journal of Ambient Computing and Intelligence (IJCAI), IGI Global (SI titled: "Storage, Process and Intelligent Systems") Vol: 10, Issue: 1, Pages: 1-29, 2019, doi: 10.4018/ IJACI. 2019010101

International Conferences

1. Anindita Sarkar Mondal, Samiran Chattapadhyay, "Comparative Analysis of Load Balancing Algorithms in Cloud Computing", Proceedings of International Conference on Advanced Computing Applications - ICACA 2021, AISC series Springer, Kolkata, (Conference Proceedings), 2021

2. Anindita Sarkar Mondal, Kshitij Pant, Samiran Chattapadhyay, "DRSQ - A Dynamic Resource Service Quality Based Load Balancing Algorithm." 2nd International Conference on Computational Intelligence, Communications, and Business Analytics (CICBA 2018), Communications in Computer and Information Science, Springer, Singapore, ISSN: 18650929, Kalyani, India, (Conference Proceedings), 2018.

3. Anindita Sarkar Mondal, Madhupa Sanyal, Samiran Chattapadhyay, and Kartick Chandra Mondal, (2017) "Comparative Analysis of Structured and Un-structured Databases." In: Computational Intelligence, Communications, and Business Analytics (CICBA 2017), Springer, Singapore, vol 776, pages 226-241,(Conference Proceedings), 2017.

4. Anindita Sarkar Mondal, Samiran Chattapadhyay, (2017) "A Storage Model for Handling Big Data Variety." In: Computational Intelligence, Communications, and Business Analytics (CICBA 2017), Springer, Singapore, vol 775, pp 59-71, (Conference Proceedings), 2017.

5. Anindita Sarkar Mondal, Samiran Chattopadhyay, Sarmistha Neogy, and Nandini Mukherjee, (2016) "Object based schema oriented data storage system for supporting heterogeneous data." In IEEE International Conference on Advances in Computing, Communications and Informatics (ICACCI), Jaipur, pp. 1025-1032, 2016, (Conference Proceedings), 2016.

6. Anindta Sarkar Mondal, Samiran Chattopadhyay, "A Context Aware Big Data Analytics Service-Centric Process Modeling Approach", M. Khan et al. (eds.), Smart Cities: A Data Analytics Perspective, Lecture Notes in Intelligent Transportation and Infrastructure. (Book Chapter)

7. Kartick Chandra Mondal, Sayan Bhattacharya, Anindita Sarkar, (2017) "A Suffix Tree based Parallel Approach for Association Rule Mining and Biclustering." In proceedings of 1st International Conference on Computer, Electrical & Communication Engineering (ICCECE 2016), Kolkata, India. (Conference Proceedings)

8. Debasmita Pal, Anindita Sarkar, Kartick Chandra Mondal, (2017) "Knowledge Discovery From HIV-1-Human PPIs Assimilating Interaction Keywords." In proceedings of 1st International Conference on Computer, Electrical & Communication Engineering (ICCECE 2016), Kolkata, India. (Conference Proceedings)

9. Kartick Chandra Mondal, Sayan Bhattacharya, Anindita Sarkar, (2017) "Comparative Study of Parallelism on Data Mining." In: Mandal J., Satapathy S., Sanyal M., Bhateja V. (eds) Proceedings of the First International Conference on Intelligent Computing and Communication, ICIC2 2016, Advances in Intelligent Systems and Computing, Springer, Singapore, vol 458, Pages: 195-204 (Conference Proceedings)

10. Soumyadeep Basu Chowdhury, Debasmita Pal, Anindita Sarkar, Kartick Chandra Mondal, (2017) "Closure Based Integrated Approach for Associative Classifier." In: Mandal J., Satapathy S., Sanyal M., Bhateja V. (eds) Proceedings of the First International Conference on Intelligent Computing and Communication, ICIC2 2016, Kalyani, Nodia, Pages 225 - 235, (Conference Proceedings)

11. Kartick Chandra Mondal, Ankur Paul, Anindita Sarkar, (2017) "Brief Review on Optimal Suffix Data Structure." In: Mandal J., Satapathy S., Sanyal M., Bhateja V. (eds) Proceedings of the First International Conference on Intelligent Computing and Communication (ICIC2 2016), Kalyani, Nodia, Pages 205 - 214, (Conference Proceedings)

List of Patents

- Innovation Patent Number: 2021107061, Kartick Chandra Mondal, Somnath Mukhopadhyay, Sunita Sarkar, Samiran Chattopadhyay, Moumita Ghosh, Anindita Sarkar Mondal, Rohmatul Fajriyah, "Suffix Forest: A novel in-memory data structure for analyzing time-series data", Term of Patent: Eight years from 24 August 2021, Australian Govt. IP Australia.

List of Presentations in National/ International/ Conferences/ Workshops:

International Conference Presentation

1. Anindita Sarkar Mondal, Samiran Chattapadhyay, “Comparative Analysis of Load Balancing Algorithms in Cloud Computing”, Proceedings of International Conference on Advanced Computing Applications - ICACA 2021, AISC series Springer, Kolkata, (Conference Proceedings), 2021

2. Anindita Sarkar Mondal, Samiran Chattapadhyay, (2017) "A Storage Model for Handling Big Data Variety." In: Computational Intelligence, Communications, and Business Analytics (CICBA 2017), Springer, Singapore, vol 775, pp 59-71, (Conference Proceedings), 2017.

3. Anindita Sarkar Mondal, Samiran Chattopadhyay, Sarmistha Neogy, and Nandini Mukherjee, (2016) "Object based schema oriented data storage system for supporting heterogeneous data." In IEEE International Conference on Advances in Computing, Communications and Informatics (ICACCI), Jaipur, pp. 1025-1032, 2016, (Conference Proceedings), 2016.

4. Kartick Chandra Mondal, Sayan Bhattacharya, Anindita Sarkar, (2017) "Comparative Study of Parallelism on Data Mining." In: Mandal J., Satapathy S., Sanyal M., Bhateja V. (eds) Proceedings of the First International Conference on Intelligent Computing and Communication, ICIC2 2016, Advances in Intelligent Systems and Computing, Springer, Singapore, vol 458, Pages: 195-204 (Conference Proceedings)

PROFORMA – 1

"Statement of Originality"

I **ANINDITA SARKAR MONDAL** registered on **18/06/2019** do hereby declare that this thesis entitled **"A MULTIMODEL CLOUD DATA STORAGE SYSTEM HAVING AN OBJECT BASED VIEW"** 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 **07** %.

Signature of Candidate:

——————————
(Anindita Sarkar Mondal)
Date :

Certified by Supervisors:
(Signature with date, seal)

1.————- —————
(Samiran Chattopadhyay)

2.————- —————
(Anirban Mukhopadhyay)

PROFORMA – 2

“CERTIFICATE FROM THE SUPERVISORS”

This is to certify that the thesis entitled **“A MULTIMODEL CLOUD DATA STORAGE SYSTEM HAVING AN OBJECT BASED VIEW”** submitted by **Ms. ANINDITA SARKAR MONDAL**, who got her name registered on **18/06/2019** for the award of Ph.D. (Engg.) degree of Jadavpur University is absolutely based upon her own work under the supervision of **Prof. Samiran Chattapadhyay, Department of Information Technology, Jadavpur University, Kolkata** and **Prof. Anirban Mukhopadhyay, Department of Computer Science and Engineering, University of Kalyani, Kalyani, Nodia** and that neither her thesis nor any part of the thesis has been submitted for any degree/diploma or any other academic award anywhere before.

1. ————————————-
Signature of the Supervisor
and date with Office Seal

2. ————————————-
Signature of the Supervisor
and date with Office Seal

ACKNOWLEDGEMENT

First of all, I wish to express my profound regards to my supervisor, **Prof. Samiran Chattapadhyay**, Professor, Department of Information Technology, Jadavpur University, and **Prof. Anirban Mukhopadhyay**, Professor, Department of Computer Science and Engineering, University of Kalyani, for their valuable guidance, scholarly inputs and consistent encouragement I received throughout the research work. No word would be sufficient to express my gratitude towards them. This feat was possible only because of the unconditional support provided by them. I consider it as a great opportunity to do my doctoral programme under their guidance and to learn from their research expertise.

I would like to express my deepest appreciation towards **Prof. Nandini Mukherjee**, Department of Computer Science and Engineering, Jadavpur University, and **Prof. Sarmistha Neogy**, Department of Computer Science and Engineering, Jadavpur University for their invaluable guidance supported me in completing this thesis. They were a great inspiration on the path of my research. Words are inadequate to express a deep sense of gratitude to members of the doctoral committee, Jadavpur University, Kolkata.

I gratefully acknowledge the research project "Remote Health: A Framework for Healthcare Services using Mobile and Sensor-Cloud Technologies" in ITRA research scheme under Prof. Nandini Mukherjee, Department of Computer Science and Engineering, Jadavpur University who is PI of the project. It helps me to find out my research challenge in real domain. I would like to thank School of Mobile Computing and Communication department, Jadavpur University, for supporting the lab work. I also wants to thank all the associated technical and nontechnical members involved in the laboratory who helps to finish the tasks. I also wants to thank to the RUSA 2.0 research scheme as the research funding agency.

All of them helped me directly or indirectly during this memorable journey. I also acknowledge all Cloud Storage System researchers as well as data scientists whose works inspired me to undertake this research. At last, I must express my sincere heartfelt gratitude to all the members of the Department of Information Technology of Jadavpur University who helped me directly or indirectly during this course of work.

The thesis would not be possible without the technical and moral support of my husband, Dr. Kartick Chandra Mondal who have shown condense in me and supported me in completing my thesis work. My Parents and In-Laws have always been there to show their blessing and inspiration to complete this thesis work. My little son Kaayaan Mondal is the source of my strength to keep going when I feel like giving up. Above all, I owe it all to Almighty God Lord Sri Krishna for granting me the wisdom, health, and strength to undertake this research task and enabling me to its completion.

Anindita Sarkar Mondal

Abstract

As the digital data generation grew, there has been an enormous increase in the demand for storing such data. This gave rise to the paradigm of cloud storage system. Side by side, cloud storage system is no longer limited to tasks of data storing or data accessing; data management became an integral part of such cloud storage systems. Data processing or knowledge extraction related jobs are also dependent on the cloud storage system for data accessing or data storage purposes. However, to build up such a cloud storage system, one needs to consider certain points, such as virtual machines as resources, storage model for deciding the storage architecture, load balancing technologies for supporting the multi-tenancy, security for protecting the data, monitoring unit to observe the health status of the system and supporting the pay as you go model.

The cloud storage system developer faces a lot of challenges and difficulties when they want to consider all these aspects and their internal relationship. The foremost challenge is the characteristics presents in the cloud storage system. Since, this data is not only big in volume, rather, it also has other complex data characteristics (e.g., value, variety, velocity, veracity). To store and access such unstructured or semi-structured data, enough resources are necessary. These resources can be any hardware (e.g., data rack) or software (e.g., model, algorithm, tool).

In this thesis, the main focus is to develop a novel cloud storage system where the issue of big data variety property is handled. A series of frameworks, algorithms, and models are presented to reach this objective. Different concepts of cloud storage systems such as the association of multiple storage devices, object-based storage models, and unique URI to communicate with the storage systems are considered here. Besides, the facilities of popular NoSQL database models and machine learning technologies are incorporated within the storage system to make the storage system more acceptable and applicable. The service quality parameters are also examined to judge the performance of the involved resources, the significance of resources in workload handling, and load balancing algorithms for handling the multi-tenancy.

The architecture of the big data management system along with the workflow model, their issues and the corresponding solutions of the big data management system, popular cloud storage system viz., google cloud platform, Amazon S3, Windows Azure, Rackspace, and Openstack Swift with the storage model of SQL, NoSQL, and cloud storage system are discussed in detail.

The proposed novel cloud storage system is named as Object based Schema oriented Cloud Storage system or RSoS System, in short. Three different data types are used as replicas of big data variety properties. Three databases are used as storage resources viz., column-family oriented NoSQL database (Cassandra), document-oriented database (MongoDB) and distributed data file system (HDFS). A storage model is presented using the concept of object-based cloud storage system for the RSoS System, which follows the account-container-object-database-schema structure. The task of this storage model is to store (and retrieve) data to (from) the corresponding storage location/device. Three query elements are considered: reading, writing, and deleting. Data service operations to communicate with the proposed cloud storage system. In order to assess the performance of the proposed cloud storage system, query execution time of data insertion, deletion, extraction, and aggregation are considered as performance metrics with Amazon S3, and Windows Azure. To do the experiment, three different data formats are used viz., health sensor data (structured

data, medical images (unstructured data), and patient's personal data (document format). A classification engine framework is designed by using a feature selection algorithm and a classifier to predict the storage space of the input dataset at the storage time. To judge the performance of the classification engine framework, namely accuracy, precision, and recall are used. Proposed RSoS system has run on top of Amazon cloud services for observing the behaviour of the RSoS system.

Load balancing algorithms in the cloud environment have been further explored. Two load balancing algorithms are proposed, namely Dynamic Resource Service Quality Based Load Balancing Algorithm (DRSQ), and Double Threshold Based Power Aware Honey Bee Cloud Load Balancing Algorithm (DTPAHBF). The purpose of the DRSQ is to find out the efficient resource based on the service quality parameters (e.g., maximum CLOCK_SPEED, minimum LOAD in last 15 minutes, minimum number of processes, minimum number of RUNNING processes, and maximum Number of CPU CORES) for handling the arriving requests and select the appropriate resources for evenly distributing the workload. DTPAHBF load balancing algorithm assists to reduce the energy consumption of virtual machines in addition to equal load distribution among the available resources.

Contents

List of Figures

List of Tables

CHAPTER 1

INTRODUCTION

In this thesis, a novel object-based cloud storage system is developed to support newer requirements arising for big data storage and management. The main objective of the proposed system is to address big data variety issue jointly with big data volume issue. The proposed mechanism enhances data access performance measures.

1.1 Context

In this era, we are moving towards a digital world that makes our lives easier. IoT devices and web applications have become a part and parcel of the new world. The data generated by such billions of devices are known as big data. Big data is said to follow 5V properties viz., volume, veracity, variety, velocity and value. Cloud has become a standard choice to handle such big data. Cloud is used to store and manage this huge set of data. Natuarlly, we refer to them as cloud storage system [1].

Historically, data were stored in the form of punch cards whose capacity was 0.08KB [2] in 1980s. Before 1956, the storage capacity of storage devices (magnetic tapes) was limited to a few KB (around 231 KB). In 1956, hard disk drive was invented and storage capacity increased to 3,750 KB. But they were huge in size and heavy in weight (nearly 1 ton). Evolution of storage system in a timeline is shown in Figure 1.2. When virtualization became possible in around 1970, research took place in a new direction [3]. With the advancement of internet services, virtualization technique has become more effective. In 1972, the first virtual machine was launched as a working framework by IBM [4]. In 1990, cloud computing was considered as empty space between the user and the provider [5]. After 1997, the concept of cloud changed drastically Professor Ramnath Chellapa of Emory University defined cloud computing as a "computing paradigm, where the boundaries of computing will be determined by economic rationale, rather than technical limits alone" [6]. This concept made it more popular in the IT industry. The first successful development made by the salesforce [7] in 1999, introduced cloud computing infrastructure as a space where other organizations could sell their software to their customers. From the storage perspective, a successful development took place when Amazon [8] launched Amazon Web Service with AWS S3 as cloud storage system in 2006. Cloud storage system allows to store a big array of data in a secure manner and also prevents data loss or corruption. It became so popular that other big organizations moved towards the development of cloud storage systems. In 2007, IBM, Google, and several other organizations invested more money on such research projects where high-speed processing and management of huge dataset was possible. In 2007, Netflix [9] launched such a storage system for supporting their streaming video services. In 2011, Apple launched the iCloud [10] to store or share personal photos, music and documents. In 2012, Oracle introduced oracle cloud [11] with three basic architecture layers, viz. Infrastructure as a Service, Platform as a Service, and Software as a Service.

Also, at almost the same time, in 2012, the incorporation of the pay-as-you-go model in cloud computing created a new milestone [12]. Through this model, customers have to pay only for the services that they consume and not not for the system utilities. Though this model was invented in 2000 by Jürgen Gehr for running his solar home system [13], this concept made cloud computing very popular to the service providers as well as service consumers from the business perspective. The timeline of cloud computing is shown in

Figure 1.1.

cloud service models are categorized as Infrastructure as a Service (IaaS), Platform as a Service (PaaS), and Software as a service (SaaS). Figure 1.3 shows the tasks of the cloud service providers and customers in these three models. The target of the Infrastructure as a Service (IaaS) [14] developer is only to provide hardware components like storage disks, cpu, monitors, network bandwidth etc. to the customers who rent them in such a manner that customers need not bother about hardware problem such as disk corruption, limitation of disk space etc. The IaaS customer needs to install their required operating system, set up their personal network, and the applications as per their requirements. The Amazon S3 [8], and Rackspace [15] act as IaaS service providers. Platform as a Service (PaaS) [16] developer provides a hardware-software framework to the software developer so that the software developers just install their required application to build up the software. The maintenance of the framework such as the operating system, hardware components, and network sections are provided by the PaaS developer. Windows Azure [17] provides the PaaS related services. A complete software product is provided to the users as Software as a Service (SaaS) [18]. Even SaaS users do not need to install any application to support the software but there has to be an internet service by which they can take the facilities of the software.

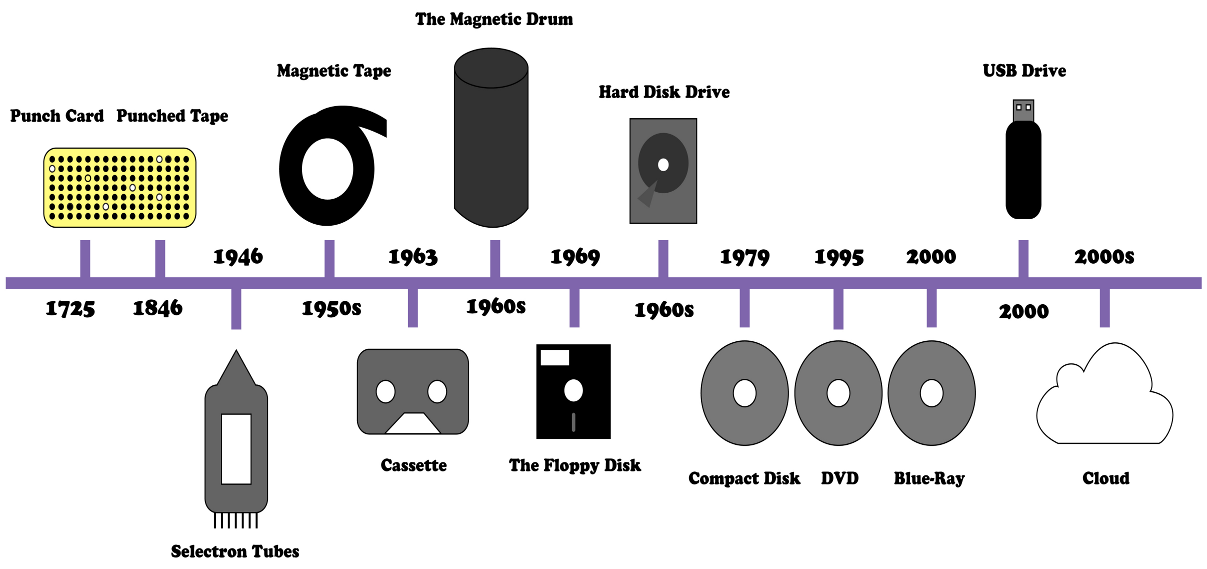


Figure 1.2: The timeline of data storage system from 1725 to 2000 [adopted from [20]]

Three basic physical models are involved in supporting storage systems shown in Figure 1.4, viz., (i) Direct Attached Storage (DAS), (ii) Storage Area Network (SAN), and (iii) Network Attached Storage (NAS). DAS [22] represents a storage system where hardware storage devices are directly connected with a server or a workstation without any internet connection. These storage devices can be connected with the host internally or externally. SAN [23] is composed of a special type of high-speed network, hosts, storage devices, storage elements, and switches, attached with servers. SAN is used for sharing real-time data between the servers. SAN enhances the performance of applications by protecting and

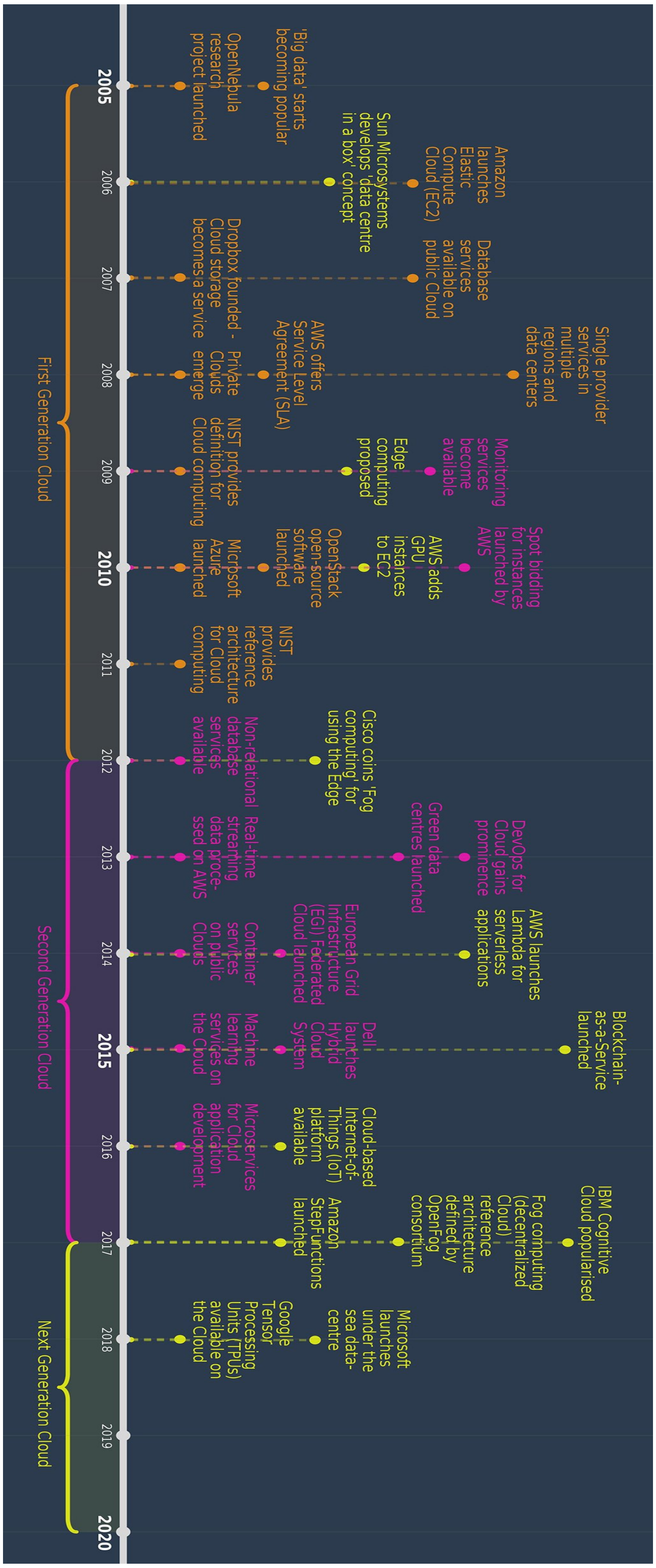

2005
2006
2007
2008
2009
2010
2011
2012
2013
2014
2015
2016
2017
2018
2019
2020
'Big data' starts becoming popular
OpenNebula research project launched
Amazon launches Elastic Compute Cloud (EC2)
Sun Microsystems develops 'data centre in a box' concept
Database services available on public Cloud
Dropbox founded - Cloud storage becomes a service
Single provider services in multiple regions and data centers
AWS offers Service Level Agreement (SLA)
Private Clouds emerge
Monitoring services become available
Edge computing proposed
NIST provides definition for Cloud computing
Spot bidding for instances launched by AWS
AWS adds GPU instances to EC2
OpenStack open-source software launched
Microsoft Azure launched
NIST provides reference architecture for Cloud computing
Cisco coins 'Fog computing' for using the Edge
Non-relational database services available
DevOps for Cloud gains prominence
Green data centres launched
Real-time streaming data proce-ssed on AWS
AWS launches Lambda for serverless applications
European Grid Infrastructure (EGI) Federated Cloud launched
Container services on public Clouds
Blockchain-as-a-Service launched
Dell launches Hybrid Cloud System
Machine learning services on the Cloud
Cloud-based Internet-of-Things (IoT) platform available
Microservices for Cloud application development
IBM Cognitive Cloud popularised
Fog computing (decentralized Cloud) reference architecture defined by OpenFog consortium
Amazon StepFunctions launched
Microsoft launches under the sea data-centre
Google Tensor Processing Units (TPUs) available on the Cloud
First Generation Cloud
Second Generation Cloud
Next Generation Cloud

Figure 1.1: The timeline of cloud computing from 2005 to 2020 [adopted from [19]]

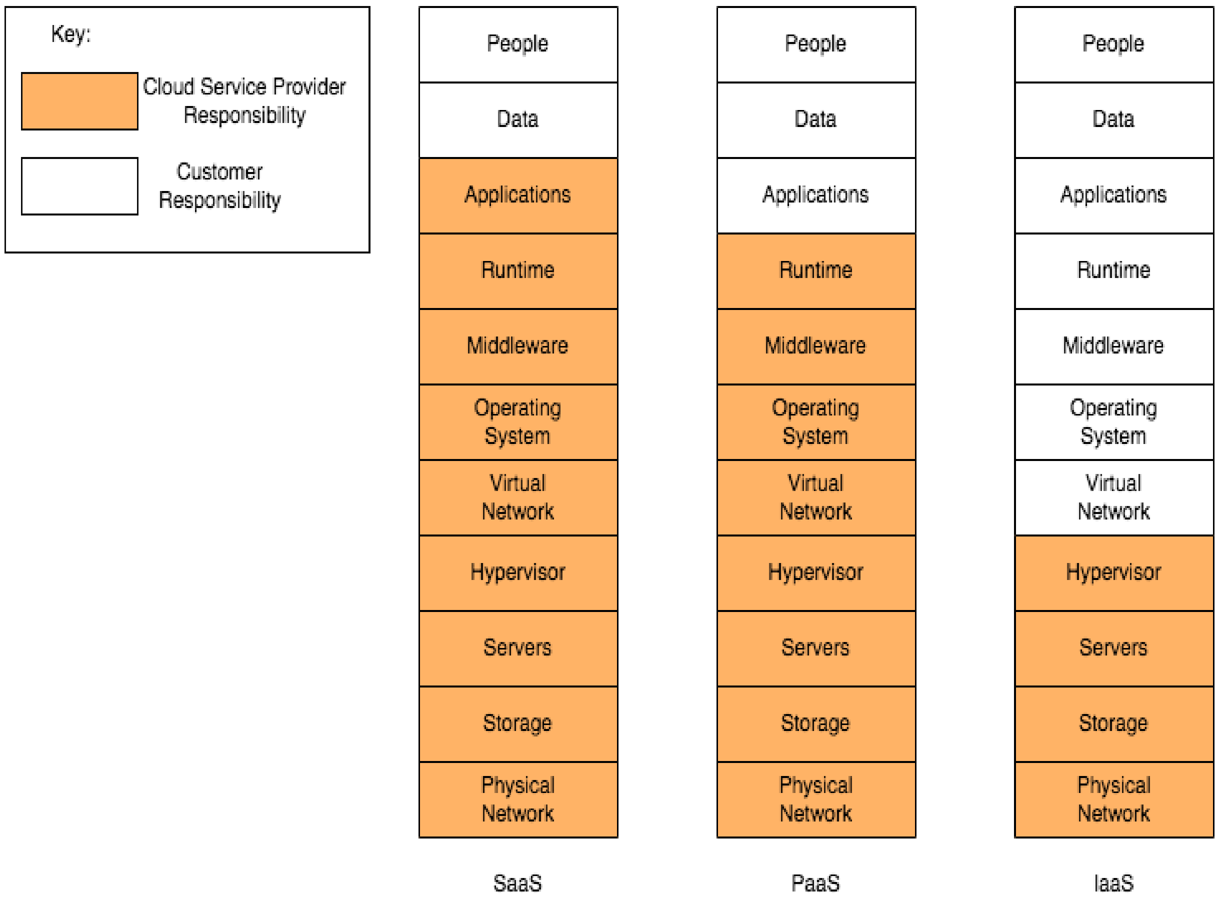


Figure 1.3: The task distribution of service models in between cloud service provider and customers of cloud computing [adopted from [21]]

managing the data in storage devices in an effective manner. NAS [24] storage devices are connected with several workstations via the network. Here, data is stored by following the file level architecture. It is used for sharing data or applications among connected workstations. With the advancement of cloud computing, cloud storage system model has changed dramatically. Through virtualization techniques, cloud computing encapsulates or hides the complex hardware infrastructure that is built up with storage devices, network connection between the storage devices, and the hypervisor that hosts software applications [25]. Developers do not need to bother about the hardware components and their corresponding network system for supporting data storage and data sharing. Also, they do not need to consider other related tasks like hardware storage device monitoring, scaling the resources according to the data requirements, etc. They focus on the storage models by creating the mirrors of virtual disks [26]. The three most popular cloud storage models are instance, volume, and object, as shown in Figure 1.5. In the instance storage model, the storage system acts as a conventional virtual disk. In general, it is built on DAS storage system but it is not very reliable. NAS is more reliable. This storage model is more popular on such applications where regular replication is needed in multiple locations. Amazon EC2 [27] and openstack [28] follows a similar storage model. To maintain data, volume cloud storage model [26, 29] divides the data into two parts: data files, and database tables. Files stores the user data and the other stores OS and application data. It is also known as block storage and is used in a SAN. For storing data files in a SAN, they are divided into several blocks and each block has a unique identifier. It maintains the information of running virtual machines through system files. OpenStack's Cinder [30] uses volume storage concept. Object storage [31] is highly reliable and very efficient for storing, archiving, backup and managing huge volumes of static and unstructured data. NAS allows sharing of data among different data centres placed in different geographical regions using a single namespace. It looks like a file. It supports eventual consistency. Amazon S3 [8] follows object storage architecture.

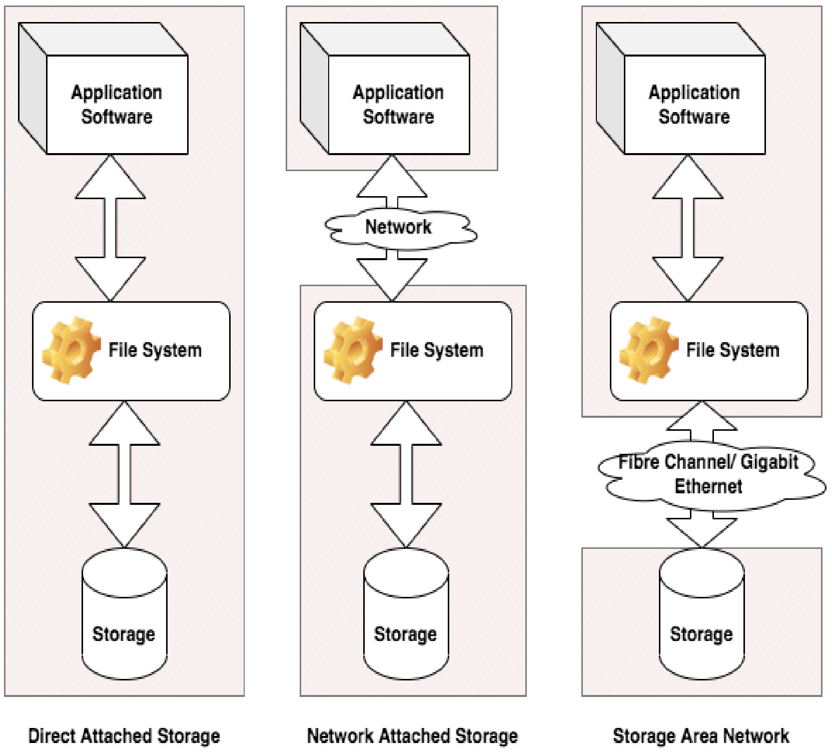


Figure 1.4: Diagram of three physical models, DAS vs. NAS vs. SAN [adopted from [32]]

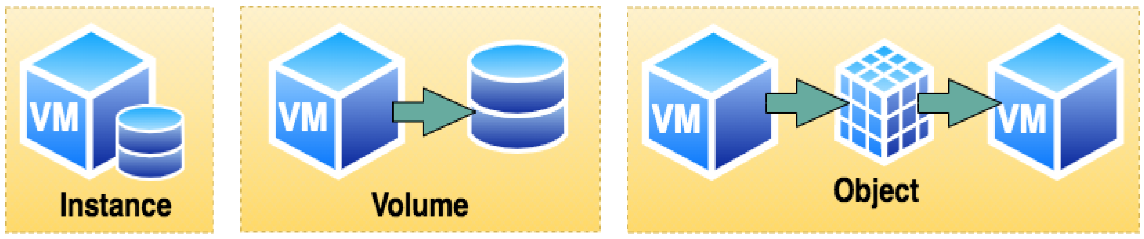


Figure 1.5: Diagram of three cloud storage models, instance vs. volume vs. object [adopted from [26]

Applications which run in a system generate data and customers communicate with the running applications to generate more data. The generated data is processed in downstream applications. Customers may want to extract knowledge from data, which may be used to keep track of performance of the systems. In 1997, the term big data was introduced in an article "Application-controlled demand paging for out-of-core visualization" by Michael

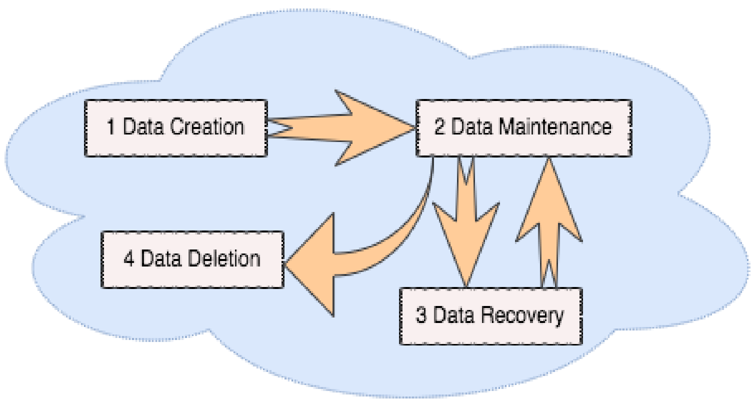


Figure 1.6: Cloud Data Life Cycle [adopted from [37]

Cox and David Ellsworth. According to this definition, local disk and memory are not enough for storing big data. In 2000, Francis Diebold, in "Big Data' Dynamic Factor Models for Macroeconomic Measurement and Forecasting" [33], said that big data would be useful in the domains of physical, biological, and social sciences. Doug Laney characterized big data in terms of 3 dimensions, also known as 3V: Volume, Variety, and Velocity in the article "3D Data Management: Controlling Data Volume, Velocity, and Variety" in 2001 [34]. In 2005, Tim O'Reilly argued in the article "What is Web 2.0?" that big data cannot be handled by traditional tools [35]. In 2005, yahoo introduced Hadoop [36] for processing big data of around petabytes in size. Hadoop was a key turning point in handling big data. In 2006, when Amazon S3 [8] was launched for storing data in a cloud. This had been another turning point in the storage of big data. Gradually, cloud computing became more popular to the customers and the size of data stored in the cloud increased immensely. Several tools became available for managing big data. In 2012, when oracle cloud [11] was introduced with three cloud computing service modules IaaS, PaaS, and SaaS, then the relationship between customers and service providers with respect to big data management got more organized and fruitful. Big data and cloud computing started to act as complementary to each other.

As big data is going to be stored using a cloud computing platform is known as cloud data [37, 38]. The life cycle of the cloud data shown in Figure 1.6 indicates data uploaded to and deleted from the cloud storage system. There are four stages in the life cycle of the cloud data [37] as discussed below: (i) Data Creation: When data is stored in a cloud storage system one or more replicas are created for supporting data backup policy. (ii) Data Maintenance: In this stage, data is going to be used by certain applications, stored for later use, and the replicas are to be maintained for later use. (iii) Data Recovery: Due to the previous data maintenance stage, the modified data or replicas can be lost. At this stage, various methods are used to recover the datas which is lost for the storage failure. This is the stage where data consistency is achieved. (iv) Data Deletion: This is the last stage where the storage space reclamation mechanism is run to delete the cloud data which is no longer used in future and also to delete those data which were not used for a long time in past. In such a way, the storage system makes empty space for new data.

Cloud computing technology is responsible to process and store big data. Due to the volume and faster growth nature of big data, it is not possible to store them in a local drive. On the other hand, each and every data item helps to reach an analytical decision. Cloud storage systems, data warehouse, data mart, etc. all are cloud provided data storage-related services. Also, data analyzer needs such powerful machine that it can process this big data.

Cloud storage is a cloud computing service model where data are managed, protected, and backed up remotely, and it is accessed via internet. A system which provides all such data storage-related services is known as a cloud storage system. There are three main types of cloud storage systems [shown in Figure 1.7]: (i) Block Storage System: This storage system is built up with SAN. Here, data is used to store in big volumes by forming blocks. The block storage system divides the large volumes into multiple nodes. It provides better performance due to the low I/O latency. (ii) File Storage System: The file storage system follows the hierarchy of directory and file for storing the data. File storage supports the storage of different data types like video, image, and text under a single node. It also supports the accessing of the same data file by multiple users. It is highly scalable, so the resources are managed in a better way. (iii) Object Storage System: Through this storage system, data in any format, viz., structured, semi-structured, or unstructured, are stored in nodes by forming objects. Each object is associated with metadata. By customizing the metadata, we can control the data arrangement in the storage nodes. This concept simplifies data storage, access, and analysis jobs. It is highly available, scalable, and durable compared to the other two storage systems.

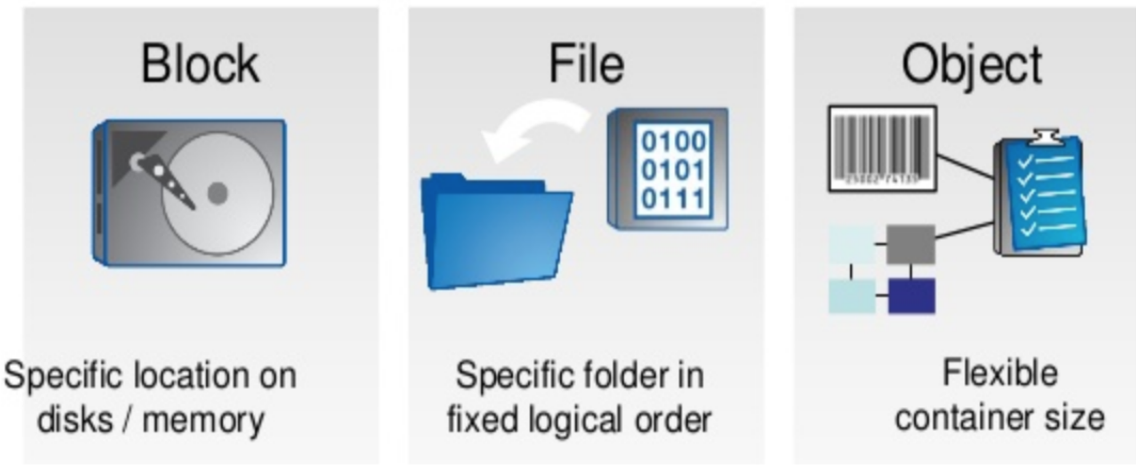

Block
Specific location on disks / memory
File
0100
0101
0111
Specific folder in fixed logical order
Object
Flexible container size

Figure 1.7: Object Based, file based, and block based cloud storage system [adopted from [39]

From the business perspective, cloud storage is divided into three cloud storage models. A diagrammatic representation of public cloud storage model, private cloud storage model, and hybrid cloud storage model is shown in Figure 1.8. (i) public cloud storage model: These cloud storage services are provided by cloud service providers. Here, customers can be anyone: individual persons or organizations. The cloud service provider is responsible for deploying, maintaining, and protecting the storage system. Users need to pay for the utilities that they are using. Dropbox and google drive are examples of public cloud storage

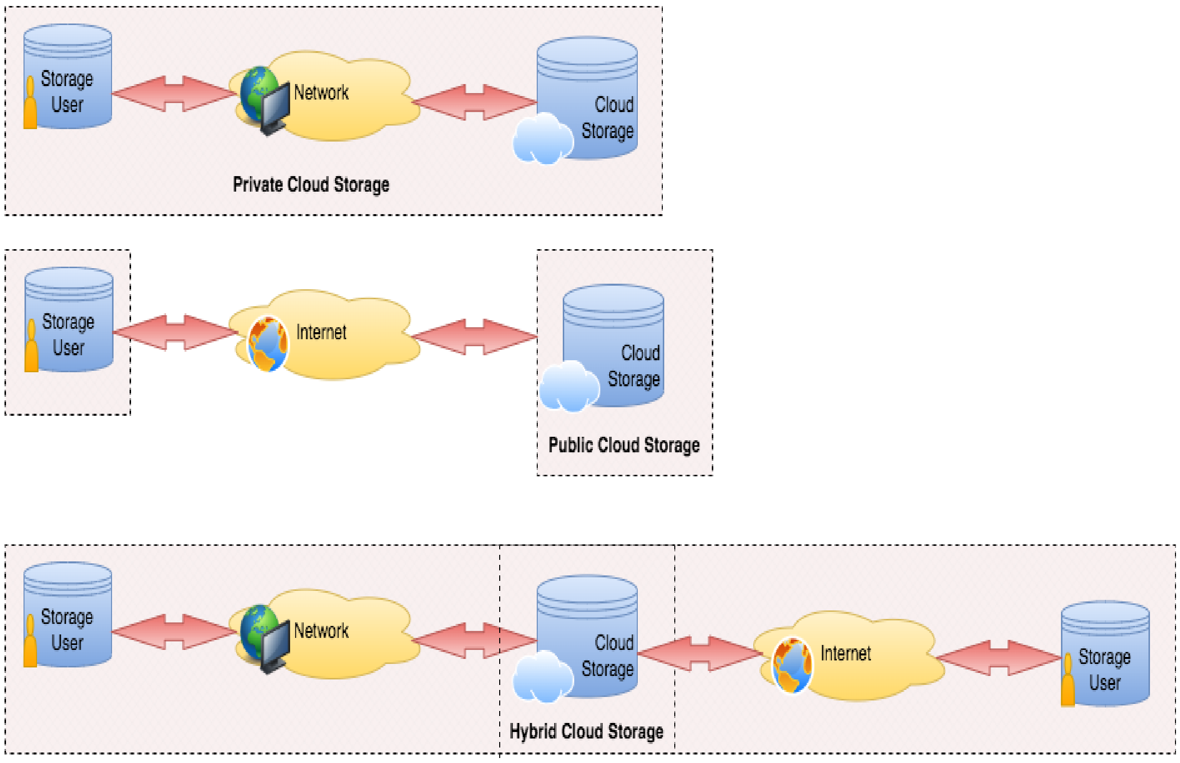


Figure 1.8: Public cloud storage model, private cloud storage model, and hybrid cloud storage model [adopted from [40]

models. (ii) Private Cloud Storage Model: Through this storage model, the storage services are restricted within an organization. The provided storage system is maintained by the in-house employees. They follow the cloud storage technology to build up the storage system but external parties have no permission to store or retrieve data. This storage model is better than the public cloud storage model with respect to data security but both the storage model have the same flexibility and scalability. This storage model is popular to maintain an organization's private data. (iii) Hybrid cloud storage model: This cloud storage model is designed by combining the facilities of both the public cloud storage model and the private cloud storage model. It is cheaper than the private cloud storage model but more secure than the public cloud storage model.

1.2 Thesis Territory

This is the era of "digital age". To maintain a smart lifestyle, people are using more computing devices and generate huge data that is not only big in size but also, complex in nature. Such data can be of any format (i.e., structured, unstructured, and semi-structured). The value of such data changes very frequently. Cloud computing applications compute using this huge and complex data. Before computation, this big data needs to be stored.

We consider that cloud storage system is the base platform. The objective of a cloud storage system is to organize such big data in the devices in such a manner that at the time

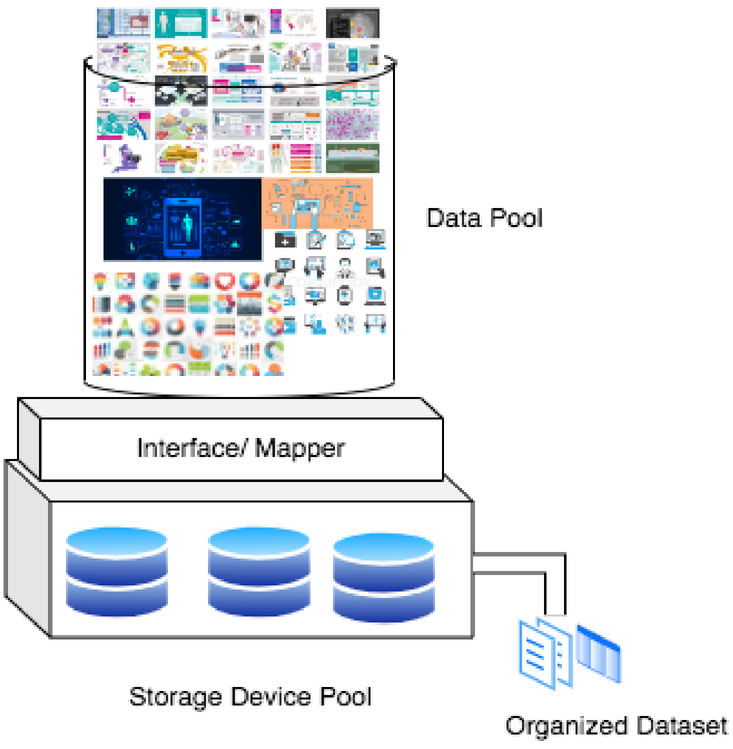


Figure 1.9: Scenario of the thesis

of computation the needed data can be retrieved as fast as possible and similarly store the generated result efficiently for further use.

According to Figure 1.9, data enters into the system in the form of a very wide pipeline and comes out from the system through a comparatively narrower pipeline. The entered raw data are collected from many input sources. For example, in the health care domain, big data sources are, hospitals, clinics, pathological laboratories, researchers and many other health-related devices. This data has to pass through some components before it is transferred as an organized data. In the process the volume of the ingested data may decrease. This organized data is then used for computation in a fast and efficient manner. Big data management system helps to reach this goal. More detailed insight is available in Chapter 2.

The cloud storage system is a part of the big data management system where data is stored in an organized manner. Here, we are dealing with big data. So, the traditional technologies e.g., RDBMS and excel file systems are not enough for storing the huge data. The characteristics of big data, such as volume, variety and velocity must be taken into account at the design time of the storage system. As already mentioned, big data does not follow any specific structure. Side-by-side a lot of resources and a large number of clients are associated with the cloud storage system. Therefore, these issues must be addressed when we design a cloud storage system.

The prime components of a cloud storage system keeping in mind the services to be provided may be as follows: storage model, storage devices, monitor, load balancing, and protection as shown in Figure 1.10. *Storage Model* describes the storage architecture in

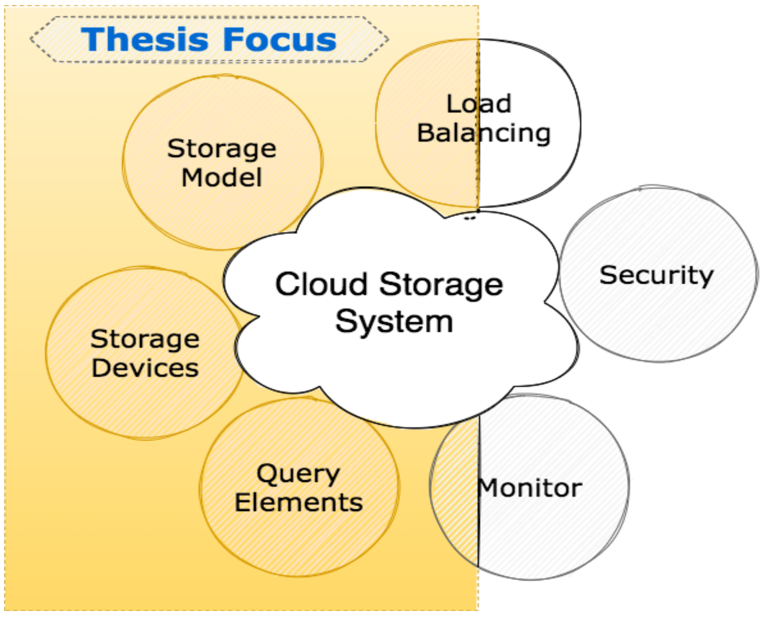


Figure 1.10: Focused Area of the Thesis

the storage system. The examples of storage models are the object-based, block-based, and file-based cloud storage models. The cloud storage system developer adds the contents with the basic storage models to fulfil the system demand. An analysis regarding the involved resources is described through the *Storage Device* component. These storage devices are nothing but a set of virtual machines. According to the system requirement, these devices are set up. *Monitor* unit is responsible to control or detect the health status of the system. This unit helps to build up a cost-efficient, service optimized, effective storage system. Multitenancy is typically associated with cloud applications. These applications have a a huge workload. This workload is balanced by using a *Load Balancing* component.

Query Element consists of a set of communication process (e.g., REST API, HTTP protocol, URI) by which cloud storage service consumers can interact with the cloud storage system. The involved resources and stored data of the cloud storage system are needed to be protect from unauthorized access or data loss by resource corruption. Through the *Security* component, the necessary actions are taken for protecting data and resources of a cloud storage system.

The objective of this thesis is to develop a cloud storage system by considering two big data characteristics, namely volume and variety primarily for health data applications. To achieve this target storage system, we focus on the following central components: the storage model, storage devices, and Query elements Additionally, a few tasks of load balancing and monitoring components are also considered. Query elements consist of a set of instructions by which consumers instruct the provider about their requirements and what type of jobs (e.g., read, write, and delete) they want to perform. Monitor components understand the consumers need and make a link between consumer requests and available resources to support the consumers' demand. As the number of consumers increase, load on the system is increased. For supporting such load, load balancing and monitoring components are used.

1.3 Research Questions of the Thesis

In accordance with the boundaries of the thesis, some research questions have been posed. These research questions are considered as the building blocks of the thesis and our target is to explore the answers to these questions in this thesis.

Question 1: How have the characteristics of digital data have influenced the design of the storage system?
As mentioned earlier, big data has popular 3V properties namely volume, variety, and velocity. For designing a storage system, one needs to understand the characteristics of the big data and their system requirements.

Question 2: How are the cloud computing solutions employed to support big data management?
The study [41, 42, 43] said that the cloud storage system are capable enough for managing the volume of big data. However, a big data storage system should be scalable, reliable, and flexible. For that reason, cloud computing is a much better solution for big data rather than the traditional data centres.

Question 3: How to enhance the time efficiency of a storage system?
The efficiency of a storage system is measured by some storage service properties, such as service response time, the maximum amount of data that can be stored at a time, the number of services that can be processed at a time, and the range of the service cost. Among them, service time is very crucial as its increase implies degradation of productivity. It is important to measure and improve how fast the data related operations (store, retrieve and delete) can be performed on a storage system. The storage system architecture and the involved components are directly related to the storage system service time.

Question 4: How to automate management of a storage system to reduce the human intervention?
A lot of machine learning technologies can be explored to find out how a storage system can be managed without human intervention.

Question 5: How easily can the users interact with the storage system?
The storage system needs a set of queries by Clients communicate with the storage system using a set of queries (write query, read query, and delete query). Hence, the cloud storage system design time needs to take into consideration the storage server URI and protocols for transferring data between the users and the storage server.

Question 6: How can the storage system manage the task load in an effective way?
A cloud storage system is accessed by URI. By using this URI, multiple clients can request for database services (i.e., retrieve, store, and delete) to the same storage system server at the same time. This multitenancy feature [44] is used to support requests of multiple users who hit the same resource. A set of load balancing algorithms are key to the issue of multitenancy. Resource distribution in the storage system may be another way of handling concurrent multiple requests.

1.4 Aims and Objective

The aims and objectives of this thesis are as follows.

- To study characteristics of health data and the associated data management system.
- To empirically investigate cloud storage system different data storage models that can handle health data volume and variety properties from the perspective of a data management system.
- To investigate the load balancing techniques for distributing data storage service requests among associated resources of a cloud storage system.
- To explore the machine learning techniques for predicting the data type of a given health data item.
- To design and develop a prototype of a cloud storage system for health data that can handle the big data variety property.

1.5 Contribution

The contributions of this thesis are as follows.

Contribution 1: An architectural diagram of a big data management system
In this contribution, we explore the evolution of big data using cloud computing. Our aim is to provide a basis architecture of a typical big data management system. The proposed architecture comprises four primary components: data source unit, data storage unit, data accessing unit, and data processing unit. Each unit of this system has individual requirements, (i) *Data source unit* refers to the sources from where the data is generated (e.g., digital device, organization, person etc.), (ii) *Data storage unit* is for describing the big data storage elements (e.g., virtual machines, hard disks) and their features (flexibility, reliability, accessibility). (iii) *Data accessing unit* consists of the procedures by which a big chunks of data is retrieved from the remote server, support multitenancy and so on. (iv) *Data Processing Unit* is associated with the data analyzing process. Through the data service pathway, the data is circulated among the components. A detailed discussion of this architecture is presented in Chapter 2 and which has also been published in [45].

Contribution 2: Characterizing the big data variety property with respect to the cloud storage system
Big data is the combination of structured, unstructured, and semistructured data. From the database design perspective, they follow different storage mechanism. It is not possible to segregate big data on the basis of the data type because they act as a single unit during data processing. This contribution talks about *storage model* that supports big data variety property. This storage model is expressed in .xml format. Initially, we have developed a simple and basic XML based structure which has

been later represented through a hypergraph. The detailed explanation is provided in Chapter 3 and this proposal is published in [46, 47].

Contribution 3: A framework for the cloud storage system that supports big data variety property
We combined the big data variety property and cloud storage system to design a cloud storage system. The name of this novel cloud storage system is "Object based Schema oriented Cloud Storage system" (RSoS). All associated components of RSoS and their relationships, and a set of algorithms are described to formalize RSoS. REST API methods are used to communicate with clients remotely. RSoS is shown to handle big data variety property through a unified platform. A detailed explanation is made in Chapter 5 and these are published in articles [46] and [48].

Contribution 4: Sketch of the storage space structure of the cloud storage system that supports big data variety property
The cloud storage system follows an object storage model to store data. These storage models could be object storage, file-based storage mode or block-based storage. Different cloud storage providers consider these models as the base and generate the storage space structure for organizing the stored data. In this contribution, our target is to develop a storage space structure that uses the object storage model as the underlying data model. The multi-layer view of the *object storage space* structure depicts how data is organized in the proposed RSoS system. The storage space is represented in terms of account-container-object-database-schema format. The detailed description is made in Chapter 5 and the related articles are [46] and [48].

Contribution 5: Interaction between cloud storage system and clients
The cloud storage server is placed remotely. Side by side, the intention of the cloud storage system is to support the database related services (e.g., store, retrieve, and delete). We explore to find out the most effective communication mechanism between the server and the users for performing database related operations. Three query elements, namely write, read, and delete are developed for RSoS using JSON and REST API. The detailed explanation is made in Chapter 5 and the corresponding article is published in [48].

Contribution 6: Design of resource-aware load balancing algorithms for cloud computing domain
We investigated how to manage task load by utilizing the involved resources (e.g., CPU, memory and network). The intent is to maximize the throughput and to minimize response time. Nowadays, researchers also focus on minimizing energy consumption and cost-efficiency to enhance the overall system parameters. In this contribution, we have designed two algorithms: one is an efficient load-balancer and the other is energy aware. The latter not only balances the workloads of the multitenants but also preserves the energy of the resources. The detailed description is provided in Chapter 4 and the related articles are published in [49, 50], and [51].

Contribution 7: Intelligent framework for allocation of storage space in a cloud storage system
Till now, cloud storage systems is used as a repository of data where analytics may

be performed using various machine learning techniques. We attempt to apply machine learning techniques for predicting the storage space of the cloud storage system where numerous storage resources are associated with it. Our framework utilizes feature selection algorithms and classifiers. The detailed description is made in chapter 6 and the related article is published in [52].

1.6 Thesis Organization

Chapter 2 surveys the state of the art big data management systems including NoSQL databases to identify the challenges and probable solutions. All big data architectures are described with an intent to have a comprehensive understanding of storage models needed to handle big data in an efficient manner.

Chapter 3 presents a storage model that is designed to handle big data variety property and describes the design of a unified platform that can handle multiple types of data. Two different data formats are considered, viz., tuple structured data and a document structured data. It is the first step towards designing a cloud storage system in the healthcare domain by focusing on the big data variety property.

Due to increasing demand for cloud computing services, the number of tenants increases and so increases the number of service access requests. But the number of available resources is limited and hence, load balancing becomes important. Load balancing techniques are involved in the distribution of the multi-tenants multi-service requests.Chapter 4 studies the load balancing algorithms for cloud computing systems. Also, we discuss the resource characteristics which are important for load balancing. Two novel load balancing algorithms are presented. Dynamic Resource Service Quality Based Load Balancing Algorithm (DRSQ) and Double Threshold Based Power Aware Honey Bee Cloud Load Balancing Algorithm (DTPAHBF).

Chapter 5 introduces a novel storage architecture on top of the chosen storage model. The proposed cloud storage system is named as Object based Schema oriented Cloud Storage system (RSoS). This storage system follows account-container-object-database-schema oriented storage architecture to store the dataset. Here, an object is considered as an abstracted box that contains differently structured data. Object storage space and hypergraph data model are used to place the data in proper storage space so that big data variety issue can be handled. A comparative analytical study is discussed with Amazon S3, and Windows Azure.

Chapter 6 describes how the proposed RSoS system can be made smarter and more intelligent. In the previous chapter, clients are made responsible for specifying the storage space for a particular data. Smart RSoS system helps overcome this drawback by introducing a classification engine framework in the RSoS architecture. The classification engine uses feature selection and classification to predict the correct storage space required to store data that is being written by a write query. We have also discussed a study on performance analysis to identify the correct combination of machine learning techniques to be employed in the classification engine.

Chapter 7 concludes with a summary of the works done. I have also pointed out the probable research directions which may be taken up in future.

CHAPTER 2

BACKGROUND STUDY

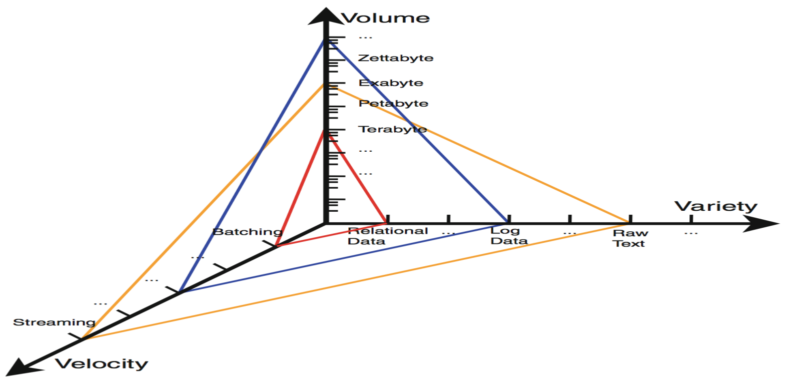


Figure 2.1: Volume, Variety, and Velocity properties of big data [adopted from [38]]

In this chapter, we discuss all terms, state of the art techniques topics related to this thesis. This chapter also brings out the basic intent of the present work.

2.1 Big Data

Big Data is huge in volume and possesses some unique characteristics. These characteristics include variation in data format, trust worthiness and ability to change values very frequently.

The term big data was coined in in 1999 by an American computer scientist, and entrepreneur John R. Mashey in his article "Big Data and the Next Wave of InfraStress Problems, Solutions, Opportunities" [53]. Francis presented an oveview of big data in his article "Big Data" Dynamic Factor Models for Macroeconomic Measurement and Forecasting in 2003 [33]. He presented big data as an explosion of data in terms of quantity that had arisen in physical, social, and biological sciences. Big data is defined by the three V's viz., volume, velocity, and variety by an industry analyst Doug Laney in "3D Data Management: Controlling Data Volume, Velocity, and Variety" at 2001 [54]. Figure 2.1 According to the McKinsey Global Institute report in 2011, cloud computing and machine learning technologies used big data [55].

The importance of big data is never ignored because of it contains an ocean of hidden knowledge. To make better decisions and take corresponding actions at the right time, one needs to extract the knowledge hidden in big data. To explain the importance of big data, Chris Lynch, Vertica Systems, said - "Big Data is at the foundation of all the megatrends that are happening today, from social to mobile to cloud to gaming."

Number of big data properties kept on increasing. The 5V properties of big data are mentioned below.

- Volume: This property discusses the massive size of the big data. Big data size is nearly petabyte.
- Variety: This property determines different types of big data. It may be structured, semi-structured, or unstructured. It also refers to the various sources from which

big data may be generated. To make a reasonable decision, a data scientist needs to consider more than one type of data generated from different sources.

- Velocity: It determines the flow of big data. Anyone can estimate the system's health by analyzing the generated continuous dataset in real-time. For this reason, this property has more importance than others.
- Veracity: This property is about trustworthiness and accuracy of big data.
- Value: This property is about the usefulness of the data for processing.

It is observed that data generated from any sphere of life such as finance, media, entertainment, transportation, administration, health, industry possess the 5V properties. Data are generated at a speed that is double compared to that of 2013. As Eric Schmidt, Google, said, "There were 5 exabytes of information created between the dawn of civilization through 2003, but that much information is now created every 2 days."

An international Data Corporation (IDC) report shows that the expected compound annual growth rate (CAGR) for health data is 36 percent; the CAGR for financial data is 26 percent; and the CAGR for media-entertainment data is 25 percent. This study says that health data growth rate is much higher than others. Health data includes medical records, patient profile, financial data, administration data, research data, and digital device-generated data. The media and entertainment dataset consists of music, movie, news, series, and advertisement audio and video. Financial service data consists of banking-related data, such as customer portfolios, banking transactions, and credit and debit information. Of them, some are structured, some are unstructured, and some are semi-structured.

Handling big data poses several critical challenges which are enumerated below.

- Requirement of ever increasing storage spaces. The high of data growth of data needs more and more storage space.
- Big Data integration: A data scientist needs to consider big data from different sources for data analysis.These data are heterogeneous in nature.
- Real time data processing: Some applications (e.g., share market, health system, banking industry) need that processing is done.
- Data Reliability: The correctness of processing depends upon the hidden information of big data. The conflict occurs when As the same data are collected from separate sources, there could be conflicts.
- Data Load: The amount of information hidden in big data is often much less. It is difficult to figure out which data are important for the purpose of analytics.
- Data Security: Big data repositories are accessed over internet by very large number of sources and client. Thus, it is important to keep this data safe.
- Data Service Response time: Cloud storage systems should be fast in terms of data access and data storage.

A data scientist needs to work with several technologies for handling big data. For example, the data scientist may need to have knowledge in one or more computer programming languages (e.g., R, python, java), statistics, data mining, artificial intelligence and so on. Afew such important technologies are introduced in the following.

- NoSQL Database: Such database are used to store big data. NoSQL database is chosen over the traditional database system becuase of its efficacy in storing huge volume of heterogeneous data that are processed in a distributed manner.
- HDFS: HDFS (Hadoop Distributed File System) is an open source specification of a distributed file system. Such a file system is used as the physical storage of data to store big data. Data in HDFS are stored in distributed clusters of low-cost hardware component. Fault-tolerance and consistency are guaranteed in HDFS.
- Apache Spark: It is an open-source unified analytic engine for real-time big data processing. Its in-memory cluster computing property makes data processing task faster.
- Splunk: It is a tool that is used to analyze the data present in the log file and convert them into a human-readable format in real-time. It consists of three components, Forwarders for collecting data, Indexers for storing data and Search Heads for analyzing and visualizing data. In addition, it helps the user by presenting system performance, failure conditions, data patterns, business matrix, and data visualization.
- Hadoop MapReduce: MapReduce is a software framework that allows executing an application in a parallel and s=distributed manner. The framework consists a mapper that divides input data into small chunks and a reducer that shuffles the generated data from the mapper and generate a new set of data.
- Hadoop YARN: YARN (Yet Another Resource Negotiator) is a resource manager for big data processing tasks. It has two modules in two ways: resource management monitor and a job scheduler. The first one monitors the applications running in clusters. If the assigned task fails, it is restarted. job scheduler launches the application on available resources based on query parameters such as capacity etc.
- Apache Kafka: Kafka is an open-source distributed message streaming platform that uses publish and subscribe mechanism. There is a time gap between message consumption and deletion to publish the messages. Besides real-time streaming data pipelines, it also supports messaging (e.g., sending notifications to consumers), storage (i.e., storing the generated messages), and stream processing. Kafka is composed of three components: Producer, Broker, and Consumer. The producer is responsible for generating messages, The Broker is a software that maintains and manages message passing. The Consumer is responsible to consume messages.

2.2 Health Data

In this thesis, health data is considered for experiments. The health care system acts as the source of health data. Health care system is multidimensional and consists of more than one component, such as doctors, technical staff, researchers, nurses, health administrators, and patients which make health data really enormous in size and complex in nature.

From market research [56], it is estimated that by 2022, the volume of health data will reach 34.27 billion bytes. This health data can be divided into categories based on their nature (i.e., data source, purpose, and so on).

- Administrative Data:
- Clinical Trial Data: This data is produced during research CONDUCTED BY scientists, doctors, and OTHER researchers. It consists of the data related to new drug analysis, treatment methods, medical tool analysis, disease diagnosis, and so on. It is important to know how to distribute the available resources between the patients and new researches.
- Electronic Health Record: It is the set of information generated during the treatment procedure of individual patients, such as lab report, medical history, personal details, post-treatment, and outcome.
- Medical Insurance Data: This data consists of patient admission, patient release, cost of treatment, and medical diagnosis data. It helps to identify which set of patients are readmitted to hospitals and the disease concerned. It also helps to identify expenditure for treatment, overuse, and reuse of the insurance. People are also interested to know which types of diseases are costlier to insurance companies.
- Patient Survey: Health care organization wants to know whether patients are satisfied with their services and the improvements needed to create a better health care system through patient surveys. Also, a government wants to know the health conditions of its citizens, which is done through patient surveys.

These data are in structured, unstructured, and semi-structured formats. According to the Healthcare Information and Management Systems Society (HIMSS),[57] Personal details of a patient may be structured data. Doctors' written hand prescriptions, etc. belong to the group of unstructured data.

Only an expert data analyst can extract hidden knowledge from this health data. For health data analysis purposes, $300 million per year is spent [56]. Until now, all generated health data cannot be analyzed due to the lack of infrastructure but are stored as historical datas. Predictive analytics reduces the fault of the health care system by 210.7 million per year. Also, 57% of health organizations believe that predictive analytics will reduce the cost by more than 25%.

As health data is one type of big data, it follows all challenges related to big data. Moreover, as health service is one of the basic necessity services, data scientists face some more difficulties.

- Big Data Repository: The repository where health data is placed needs to have a faster data access and storage facility because it may involve questions of life and death.

- Health Data diagnosis: The chances of providing better and more fruitful health services will be increased by how much and faster data scientists can help in the process of diagnosis.
- Fault Tolerance: The loss of a patient's health reports will be catastrophic to the patient's treatment procedure. Therefore, the system where the data is put should be super fault-tolerant.
- Data Security: Health data holds some private information like patients' personal details, hidden diseases which they do not want to disclose, medical insurance-related data of a patient, health billing, and so on. These are essential and secure information.

2.3 Big Data Management System

Big data management system is one that helps data scientists to work on big data in an efficient manner. It combines strategies and technologies adopted by the organization to store, process, and visualize the big data.

2.3.1 Data Service Pathway

The data service pathway describes the movement of data from source to consumer through a big data management system. See Figure 2.2. It consists of four entities: (i) Data Source, (ii) Data Manager, (iii) Data Warehouse, and (iv) Service Consumer. IoT devices, disks, or memory act as data sources. The generated data formats are text, image, video, or audio. The Data Manager is responsible for collecting, integrating, preprocessing, and storing the generated data in Data Warehouse. The data storage structures are variable, i.e., they can be documents, rows, columns, or tree-based. The *service consumers* retrieve the data from the *data warehouse* as per their requirements. Service consumers can also control other entities, like a proper selection of data sources, presenting the service requirements to the data manager, such as expenditure amount, access time, data, and storage duration.

For managing big data, a *data manager* faces a few challenges, such as (i) lac og availability of storage (ii) the ratio of data volume between valuable data and the not so useful data (iii) data concurrency and consistency, (iv) supporting data variety issues by providing different types of data storage units. Also, there are other requirements. For example, such a data manager cannot use an unlimited amount of resources to support the system or needs to assign the same amount of resources for the same operation. The management system should be secured, reliable and flexible enough to satisfy the users' service demands.

2.3.2 Visualization

According to big data variety property, one type of storage system is not suitable for storing all data types. A good storage model solves this problem, as mentioned earlier, by deciding

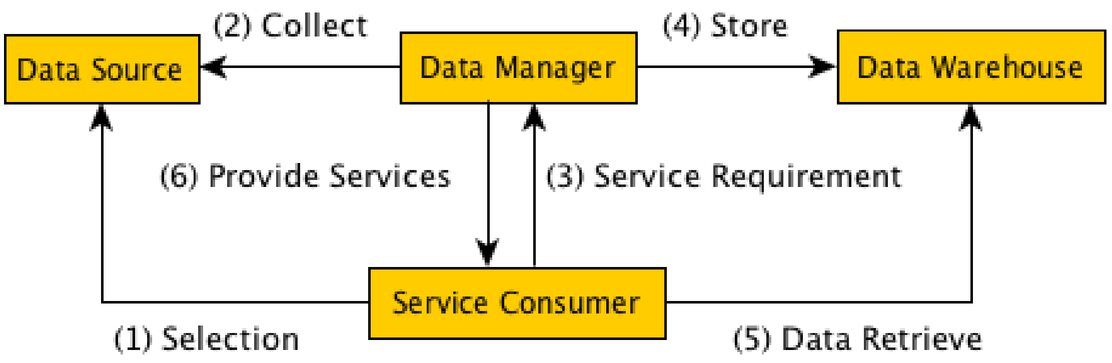


Figure 2.2: Graphical representation of Data Service Pathway

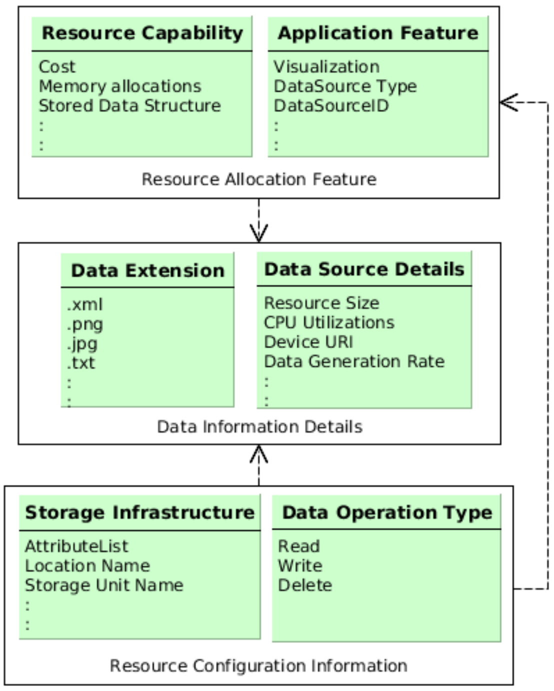


Figure 2.3: A UML Representation of the Components and their relationship

the allocation of storage resources (or databases) based on the data type. It can even customize the storage structure of each storage resource. These storage modeling-components not only helps in the way storage resources are allocated or configured but also helps to give a structural representation of the data storage units. Storage resources are nothing but the set of data storage devices controlled by available databases like Cassandra, SQL, MongoDB, Neo4j, and HDFS. The components of a storage model and their relationships are shown in Figure 2.3.

2.3.3 Issues and Solutions

Data Complexity:

Data complexity is about the the characteristics which make one data similar or dissimilar to other data. For example, health data is collected from different sources. Also, the collected data are different in their structure. Some are tuple-oriented, some are file-based, some are graph-oriented, some are web-based representation, some are image structure and so on. To put all these differently structured data in a single data modelis a challenging task. Two approaches are considered here as probable solutions.

Mapping-based data storage: In this storage architecture, a single data model and interface is used to support different data model of the various database for storing and accessing the collected data. Figure 2.4 shows the multi-layer architecture of mapping-based data integration approach. A client communicates via put, get, and delete queries. In the data representation layer, data is arranged according to its context, i.e., value, entity, relationship, and metadata. Then, this data is mapped with NoSQL databases' data model and stored in the corresponding database.

There are different mapping approaches for storing and accessing data in databases. According to object-data store mapping, [58, 59], data are represented as objects. Each object has an entity, a value. The entities are related. During data storage, data elements (e.g., a snapshot of the aggregated data, metadata of their aggregation relationship and contents) are mapped and stored in the particular NoSQL databases (e.g., Apache Cassandra, Couchbase, Oracle NoSQL, DynamoDB, MongoDB and Redis). For this purpose, authors in [58] use the concept of NoSQL abstract data model (NoAM) [60]. It is designed using the common behavior of different NoSQL databases.

In Save, [61], MetaLayer is used to perform the mapping between several languages with a common language. MetaLayer acts in two steps: (i)It manipulates the collected data objects according to their structure. (ii) Performs mapping to store these data parts according to their type. The collected data object representation is organized in three parts: (i) Set: defines the collected data, (ii) Struct: Each object of the collected data, and (iii) Attribute: the value of each object. The mapping operation maps the set part with document-oriented NoSQL database MongoDB, struct part with hash-based NoSQL database Redis and attributes part with column-oriented database HBase.

Ontology based data access: Ontology-based data access approach tries to resolve the dissimilarity between different NoSQL data storage systems [62, 63, 64]. Authors use semantic webs to integrate different NoSQL data stores like document-oriented data stores (e.g., Mongodb) and column-family data stores (e.g., Cassandra, HBase). This technique mainly works in three ways, as shown in Figure 2.5. (i) It generates the local ontology by considering the corresponding schemaless NoSQL data stores, (ii) It designs the global ontology by considering all local ontologies (iii) SPARQL queries written by end-user is translated into the corresponding set of NoSQL data queryies This translation takes the help

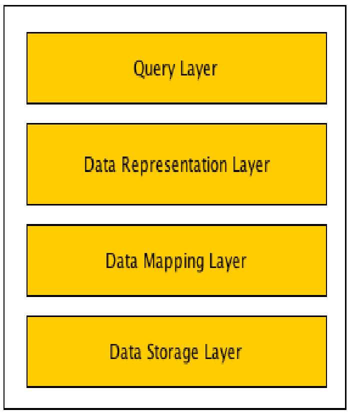


Figure 2.4: Architecture of Mapping-based data storing approach

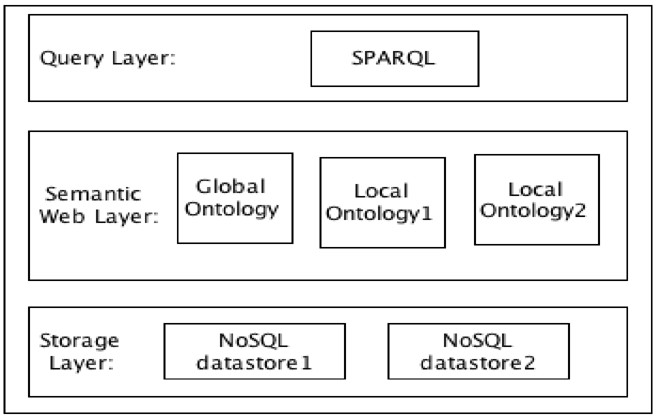


Figure 2.5: Architecture of ontology-based data access approach

of global ontology and local ontologies to access data from the corresponding NoSQL data storage system. This approach helps to enhance data quality and data consistency.

Resource Optimization:

It is a technique by which available resources are distributed between the set of organizations to optimize system parameters. The growth of big data entails an increase in the cloud resource demands for storage, analysis, and so on. It is challenging to meet the expectations and requirements of all organizations. As we know, the capacity of any cloud service provider is limited. In such cases, cloud service providers consider different approaches to use their resources optimally. In this survey, we shall present two such popular approaches.

Pay-as-you-go: Every user does not want to invest a huge cost for managing their data. In this model, consumers act as the controller. They specify their demand for the resources. Amazon Simple Storage Service (S3) [8] provides an elastic storage system to the user as a server on-demand service for storing frequently used large data. It also provides life-cycle support like durability, backup, recovery, and content distribution of the stored data. Microsoft Azure [65] provides window server-based cloud platform where customers can deploy their applications. In this model, customers only pay for the resources used for deploying the application or storing the data.

The cloud computing platform helps the service consumer to reserve the resources in the form of virtual machines. Initial reservation helps to reduce the total computing time by avoiding initial setup time. Users who need more resources can achieve it via on-demand services. Amazon EC2 [27] supports this reserved resource methodology. Here, users can assign resources on a primary basis for starting the cloud computing job. If the service consumer needs extra resources, he can assign them by paying only the additional cost.

Data Size Minimization: Big Data naturally contain redundant data. A massive amount of unnecessary storage spaces are required to store raw data. In healthcare, the patient's data needs to be protected from disaster. Allover, the data needs to be arranged and stored so that it uses few resources with backup.

The data management system must be efficient enough so that it can use a minimum amount of resources to store data. MapReduce techniques [66] help minimize resource usability by merging the data using the notion of key-value pairs. The authors in [67] proposed a 3-stage approach for end-to-end set similarity joins to support balancing workload among different nodes. This is done to minimize the need for replication and control the main memory data size in each node. Llama [68] designed a technique to compress the data size for reducing storage requirements.

Data Concurrency:

The main difficulty in big data management is that it is necessarily distributed distributed in nature. Simultaneously, multiple users update data to a different ends. In this situation, there is a high probability of data override or loss of data. So data management becomes challenging for supporting data concurrency by synchronizing the data copy of all nodes with the updated data copy so as to maintain data consistency. We present two most fundamental approaches for this job: MVCC [69] and OCC[70].

Multi-version Concurrency Control (MVCC): In a multi-user database environment, the shared database can be manipulated by multiple users simultaneously using locking. Here, the read operation must wait for the write operation to finish or vice-versa. Multi-version concurrency control allows users to write and read operations on the same data simultaneously without locking the data file.

This method considers multiple versions of data. Figure 2.6 represents the architecture of MVCC. At the transaction time, the write operation updates the records and saves the new version. This new version is identified by the transaction id, a single number that number with the number of transactions. For the read operation, there is confusion about which version is needed to be read. Transaction id of the record version is compared with the Transaction id of the read operation. If the read operation is started after the last write operation committed, it returns the record's latest version. This technique stores every updated data version by using transaction id as an identifier. It helps to return the old copy of the data record. PostgreSQL[1] supports MVCC using two transaction IDs and a row-based data record version. The two transaction IDs are (i) creation transaction id: The last updated data row version is signed by this id. and (ii) expiration transaction id: Before the last modified data row version is identified by this id. Blobseer [71] uses metadata to serialize the different versions of the distributed data blob (i.e., a set of data content act as a unit and is named as a data blob), which is generated after the write operation. CouchDB[2] allows a client to consistently read and write on the same document by supporting MVCC. Short-running transactions do not wait to finish long-running transactions in NuoDB[3] to support MVCC. Other technologies also support the

[1]http://www.onlamp.com/pub/a/onlamp/2001/05/25/postgresql_mvcc.html
[2]http://guide.couchdb.org/draft/consistency.html
[3]http://dev.nuodb.com/techblog/2013/03/12/mvcc-part-1-an-overview

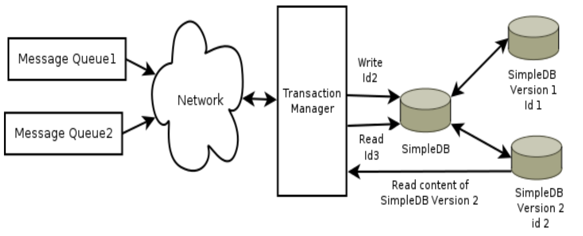


Figure 2.6: Architecture of MVCC

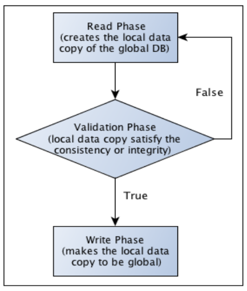

Read Phase (creates the local data copy of the global DB)
False
Validation Phase (local data copy satisfy the consistency or integrity)
True
Write Phase (makes the local data copy to be global)

Figure 2.7: Strategy of OCC

MVCC are Virtuoso[4], Cassandra, IBM DB2[5], DynamoDB[6].

Optimistic Concurrency Control (OCC): This type of concurrency control supports a multi-user, shared database system. This technique is helpful in dominant query systems where transaction conflict occurs rarely. In the case of locking, it locks the whole data even in a read operation. This may lead to a deadlock. OCC avoids the locking of the data. This procedure helps to provide a deadlock-free database system.

Figure 2.7 shows the strategy of OCC for data transaction jobs. The transaction has two operations: read and write. Loss of integrity does not occur for read operations. However, when it is going to return the data content of the read query to the application, all write phases have to be completed. Writing passes through three phases: read, validation, and write [72]. Firstly, data is read, a local copy of the data is made, and all data writing is done on the local data copy. After reading the data, the transaction process goes through the validation phase. Here system checks the consistency of local data copy. The read operation checks the similarity of the current data with the local data copy, and the write operation checks the validity of the updated local data copy. If the validation phase returns true, the writing phase is entered where local data copy is made global. The global database is rewritten with the local data copy for a write operation, and the local data copy is returned to the application for a read operation. If not validated, rollback is performed, and the operation is restarted. OCC improves the data service performance by avoiding the waiting time for data locking. Microsoft SQL Server[7] supports optimistic concurrency control to support data transaction.

Data Scalability:

Nowadays, the volume of big data is increasing proportionally with the number of devices. Also, historical big data needs to be kept in storage because no data can be deleted. At the same time, a data storage system is also affected by a data-related operations like backup, recovery, and fault tolerance. Data scalability is required to handle the growth of data and applications. The challenge is how to make a big data

[4]http://virtuoso.openlinksw.com/
[5]http://www.ibm.com/software/data/db2
[6]http://aws.amazon.com/DynamoDB/
[7]https://msdn.microsoft.com/en-us/library/aa0416cz(v=vs.110).aspx

management system scalable. Two crucial approaches are discussed in the following.

Scale Up/ Vertical Scaling: Such scaling adds more hardware (e.g., RAM, CPU, network) in a single machine. Vertical scaling aims to increase the power of hardware to achieve the desired performance Suppose an application needs more memory for handling increasing data size. In that case, the load balancer adds memory to a single machine, or if it wants to make faster data access, the load balancer adds more processing units as shown in Figure 2.8. This technique applies to read-heavy applications like a blog website, where most people like to read an article rather than a comment. Single address space with multi-threading is used for performing tasks. Message passing and data sharing are done by passing the data file references. There are limitations to adding hardware in a single machine.

Amazon Relational Database Service (Amazon RDS)[8] provides a web service to manage the multi-user accessible relational database in the cloud. Each hardware component is handled individually to make web services more cost-effective and efficient for the data service (e.g., recovery, backup) by following techniques of vertical scaling. Object-relational database system PostgreSQL[9] considers scale up methodology to increase the power of the server for supporting increasing transactional workload. In-memory document-oriented data storage system, D-Store [73] adds more memory in a single server to decrease the cost/size ratio. MySQL[10] supports vertical scaling for providing multi-user, in-memory storage space.

Scale Out/ Horizontal Scaling : Scale-out provides a much better solution based on cloud computing and by adding new nodes in a distributed manner. These nodes act as web servers with storage devices, networks, and processors, forming clusters. For handling scalability, the load balancer adds machines rather than hardware in a single machine, as shown in figure 2.8. During the processing time, data files are shared via the network. There is a low operational impact for adding nodes in the cluster to handle the increasing volume of the data because the structure of the new node is the same as that of the first node. This technique is more beneficial for write-heavy applications like online money transfers, where more write transactions, rather than read, are involved.

Different technologies support horizontal scaling to achieve specific goals like resource cost, data recovery, data processing time, etc. CouchDB [74] is a document oriented database system where data is stored in .json format. It supports horizontal scaling where replicated data is stored in distributed web server to maintain high performance and fault tolerance. For better data services like minimizing data query time from a large dataset, HDFS [75] is used. HDFS reduces processing time by distributing data nodes with the processor. VoltDB [76] provides relational database management system for big data. Real-time data analytics enhances data service performance. For this reason, data is partitioned across machines in a cluster to support parallel data processing. Other

[8]https://aws.amazon.com/rds/
[9]http://www.postgresql.org/
[10]https://www.mysql.com/

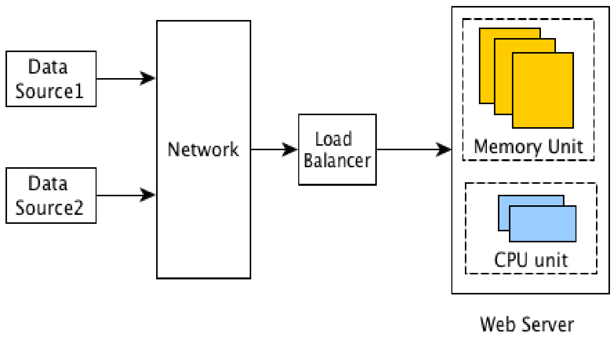


Figure 2.8: Architecture of Vertical Scaling

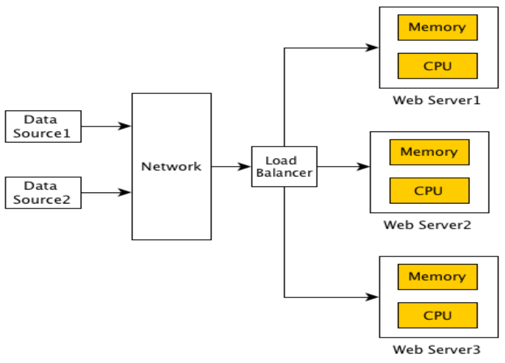


Figure 2.9: Architecture of Horizontal Scaling

database systems like Cassandra[11], HBase[12] follow the horizontal scaling technique.

Data Reliability:

It deals with data integration, data completeness, data accuracy, and data consistency of the collected data. Also, big data is collected from different sources. So, there is a huge that may lead to unreliable data. The issues regarding reliability are discussed below.

Statistical Parameter-based : These parameters are used to measure data reliability.

- **Accuracy:**.
  Accuracy is the number of correct values/number of total collected values.
- **Completeness:** It measures whether data values are present in the database corresponding to the actual real-world data. It is defined as:
  number of not null values/number of total values
- **Consistency:** It determines the integrity of the data present in the database. It is defined as:
  number of consistent data values/number of total values.

Data in the database system are more reliable to a data analyzer if optimum of the parameters discussed earlier are optimum. Therefore, the main focus of reliability approaches is to increase the values of those parameters. These approaches are heuristic or knowledge-based algorithms[77].

Data-Driven : This strategy focuses on the update of data to increase the parameters of the data reliability. This approach follows different techniques as mentioned below, (a)Update the data with the more actual real world data, (b)Follow a fixed set of integrity constraint rule to design the database. (c) Synchronize data of the same objects stored in different databases. (d) Integrate data obtained from

[11]http://cassandra.apache.org/
[12]http://hbase.apache.org/

various heterogeneous sources. (e) Select data source so as to increase the parameter values. (f) Discard the data that cannot reach the quality needed for a data analyzer Through using statistical techniques. However, this technique offers a temporary solution to data reliability.

Amazon supports data reliability to maintain financial needs and customer trusts [78]. In this case, data reliability depends on the state of the application. For example, healthcare needs high-quality, very reliable data. Amazon has designed Simple Storage Service (S3) to support data reliability. Dynamo manages the state of the services, which requires high reliability.

Process-Driven : It examines the processes which are responsible for generating the storage data. To avoid erroneous data collection and processing, processes responsible for data generation, data update, and data processing needs to be controlled. Process redesigning helps to improve the process quality by adding new techniques. It attacks the root cause of data reliability issue and hence, results in which applies to long-term data reliablity.

Hadoop uses MapReduce [66] for processing large sets of data and generating reliable data that the processors use. MapReduce can automatically detect any hardware failure, which helps maintain the data quality. Hadoop helps in the bioinformatics field for processing and knowledge extraction from large datasets of about petabytes size. [79]. This paper uses HBase and MapReduce techniques to achieve data reliability.

Availability:
Availability means "readiness for use". For real-time applications, a continuous flow of data is needed. It must not stop due to system failure. Llama [68] provides continuous data services by rescheduling a task on another machine if one map task fails.

Recovery:
Due to some reasons, if the system crashes, the stored data are not lost. It must be recovered by allowing/providing data replication on different nodes. GFS [80] allows data recovery by performing data replication. Data partitioning is a technique to support data availability. GFS supports the data partitioning technique. PNUTS [81] supports data availability via redundancy at different levels (data, metadata, serving components, etc.).

Security:
Data Security helps to protect data from corruption and unauthorized access. Amazon S3 [82] controls user accessibility by providing client authentication.

2.4 Evolution of Storage Systems

A database's logical structure is determined by the data models, which specify the organization of data in the database. These models provide a layer of abstraction to the database and describe how data is present and how they are interlinked.

Structured Data [83] refers to the data that are essentially present as the field-rows-column structure and can be easily manipulated by algorithms and queries. Some common examples include data in excel sheets and relational data in a relational database. Semi-structured data [84] are not precisely organized as relational data but have some structured format like XML tags that enable analysis algorithms to read data easily. JSON (JavaScript Object Notation) data is a common example of semi-structured data. However, most of the data present today is unstructured data [85] which does not follow any format, for example, prescriptions, cliical notes, videos, and other multimedia content.

2.4.1 SQL Data Model

The logical structure of a database defines the organization of data. The earliest data model was hierarchical data model e.g., IBM Information System which was followed by hierarchical database [86]. Further evolution of databases resulted in foundation of relational model [87] where data is represented as tuples or rows which aggregate to form a relation and such systems are called Relational Database Management System (RDBMS). RDBMS uses structured query language (SQL) as its data query and data manipulation language. The storage architecture of RDBMS helps in maintaining the entity relationship [88]. However, the organization of data in the form of field is very stringent, and rigid that leads to many limitations.

PostgreSQL (Example of Relational Model RDBMS): Data are stored as tuples in relations (tables) in PostgreSQL. Structurally, it comprises different modules or components which communicate amongst themselves and function to form a complete working unit of the database [89].

As shown in Figure 2, the components are Client Process (Client Application + Client Interface), Server Process (Server+ Postmaster), and back-end or data Storage. The end-user communicates with the client process via the client application. For example, the user might write SQL queries or procedural methods to access data in the application. One of the main components of the client process is the Interface. It converts procedural code to SQL queries that the server can understand. Then comes the server process. Two essential components are present here. First is the Postmaster, which maps a client process to a server process. Next is the Multi-version Control System, which ensures a proper locking mechanism while any server tries to access the innermost component.

2.4.2 NoSQL Data Model

NoSQL (Not Only Structured Query Language) not only supports storing of data but also supports durability, reliability, availability, and scalability [90]. Furthermore, the NoSQL database follows CAP (Consistency, Availability, and Partition Tolerance) rather than following the ACID property.

Considering the NoSQL databases, they have better management of semi-structured, and unstructured data [91]. There are four types of NoSQL databases, (a) Key-Value: In this NoSQL database, data are stored in groups. A group is identified by a unique identifier known as the key. Amazon S3, and Azure follows this type of data storage structure to store a large volume of data such as Redis [92] (b) Document: A set of data groups with variable attributes are stored by forming a document. This document is identified by key value and presented in XML, JSON, or BSON format. CouchDB, and MongoDB are examples of document NoSQL databases. (c) Graph: In a network-based system, an entity's instance is connected with another instance of another entity, and this connection has explicit meaning to the storage data. In this situation, the graph database stores the data by holding information about how one instance is connected with another. OrientDB, and Neo4j are the most popular graph-based NoSQL databases. (d) Column-family: In this storage, data are stored column-wise rather than in terms of horizontal tuples. This concept makes data operations (i.e., access, storing) faster. Cassandra, and HBase are two examples of Column-family NoSQL databases.

- MongoDB (Example of Document Model Document-oriented Database): In MongoDB [93, 93, 94, 95], complex documents are stored as arrays, hash tables etc. that are supported by JSON Documents. MongoDB stores documents as data in a binary representation of JSON called BSON. An index is used for ordering the documents as a collection in MongoDB, and a unique id is automatically created for a particular index. Index helps in sorting the documents [94, 95]

  Several documents are integrated into a collection. Collections can be compared to tables in RDBMS, documents to rows, and fields to columns. Data are split across numerous shards. The application that needs to access data in a shard connects to underlying MongoDB processes. However, small collections need not be split across shards. As given in the Figure 2.10, the structural components of MongoDB are:

  1. Shard: Multiple shards are present, which hold parts of data. Read and write operations are done on the target shard, i.e., the primary shard. A replica set is present as a backup to the primary shard. All the changes need to be distributed gradually across replicated secondary shards.
  2. Config Server: Multiple config Servers are present, storing metadata about what data is stored in a shard.
  3. Routers: The client directs queries to routers. A router consults with the config server and redirects to the desired shard.
  4. Mongo Client Library: Clients, which are a part of an application, issues a query to be routed via the client library.

- OrientDB (Example of Multi-Model Document-oriented Database): OrientDB is the first multi-model NoSQL database whose engine supports four kinds of data model: document model, graph model, key-value model, object-oriented data model [96]. Generally, multi-model databases have an abstraction layer with a set of APIs to handle multiple models. It limits performance. But in OrientDB, the engine provides direct support to the data model. It combines the graph concept in its representation of data as documents. "LINKS" are between documents that cannot be found in document-oriented databases like MongoDB. The graphical representation of data in

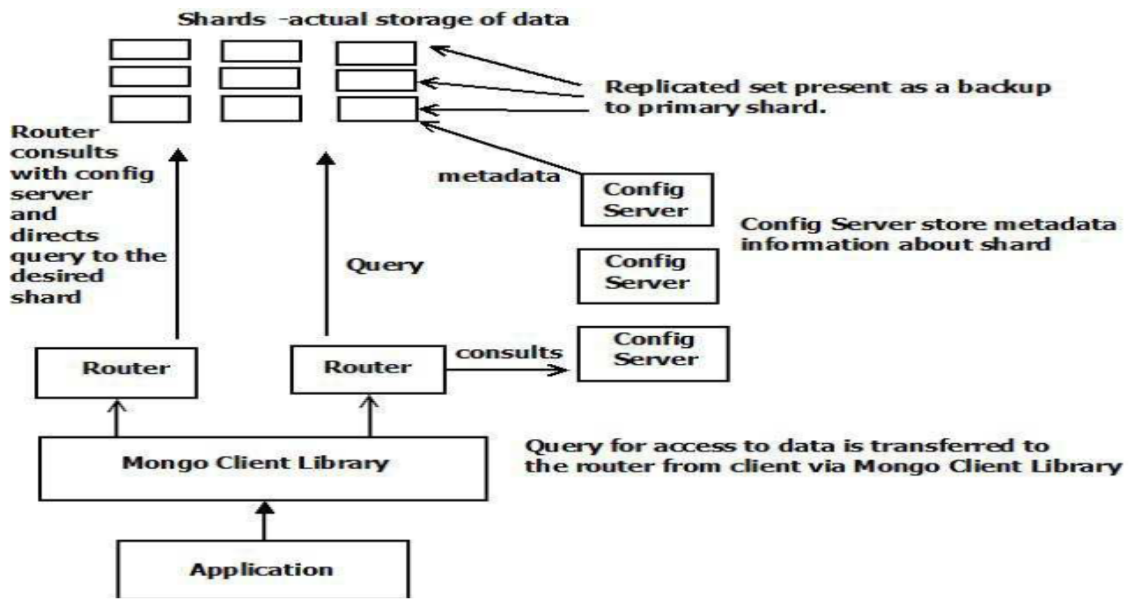

Shards -actual storage of data
Replicated set present as a backup to primary shard.
Router consults with config server and directs query to the desired shard
metadata
Config Server
Config Server store metadata information about shard
Query
Config Server
Config Server
Router
Router
consults
Mongo Client Library
Query for access to data is transferred to the router from client via Mongo Client Library
Application

Figure 2.10: MongoDB System Storage Structural Components

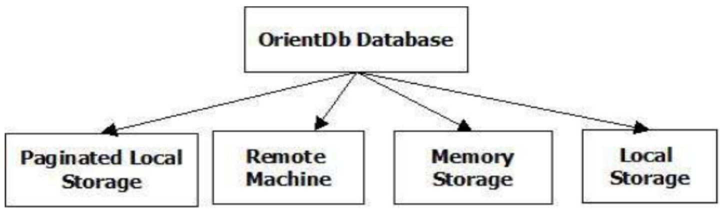


Figure 2.11: Data Storage supported by OrientDB

OrientDB is almost similar to that of a graph database. One subtle difference is the property of Inheritance which is applied while defining a "class"(analogous to tables in RDBMS). The class extends Vertex class "V" or edge class "E".

As shown in Figure 2.11, the OrientDB database supports four kinds of storage, namely Paginated Local Storage(PLocal), Remote Storage, Memory Storage, and Local Storage. PLocal comprises Clusters, Write Ahead logs (WAL), Indexes and Index Containers, and File mapping. A cluster is a logical portion of disk space to store records/data. Clusters are split into pages. Hence the name "Paginated". cpm files in the cluster map the cluster location of file data to its actual physical location. WAL is used to record operations/activities. Indexes are necessary to store the file, and file mapping maps the file name to the file id. Remote storage supports Data storage in remote machines. Memory storage is where data is stored in memory, and Local Storage is disk storage which PLocal has replaced.

- Neo4j (Example of Graph Model Graph-oriented Database): Neo4j [97, 98] is essentially an open-source NoSQL database implemented in Java. It follows the Graph Data Model and maintains data in that way, even at the storage level. On top of it , an additional Cache is maintained for implementation of Node-Relationship. Disk is organized into record-based storage assigned to every data structure (node, rela-

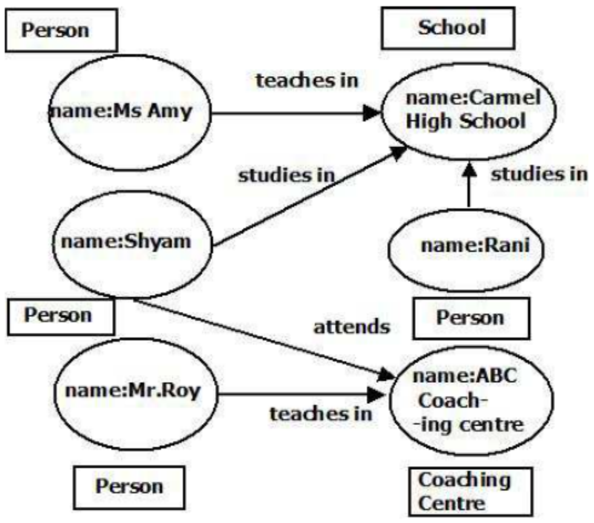


Figure 2.12: Components of Property-Graph model

tionship, and property). Each node and relation are identified by ids. The size of a block depends on the type of data structure stored. The data represented in the graph storage system mainly comprises the following components as shown in Figure 2.12.

- Node: It contains labels, and key-value pairs. Pointers are there, which point to the first relationship, and the first property block.
- Property: It stores information related to a node. For example if "Sachin" be the node, its properties may include run, average, noOfMatches etc.
- Relationship: It defines the kind of association. Along with it, pointers are present. These point to the start, and end the node; There are pointers to the previous and the next relation of the start and end node.
- Label: In Figure 2.12, these are represented as rectangular boxes. Each node is associated with a label that must specify the creation time. Nodes belonging to different labels may have relationships (edges) between them. Along with it, pointers are present. These points to the start, and end node; first property block; pointer to the previous and next relation of the start and end node.
- Caches: They Implement nodes or relationships.

2.4.3 Cloud Storage Systems

Cloud storage systems [99, 100, 101, 102] store data in remote hosts, and it is expected to be available all the time. A third-party vendor manages data in cloud storage. There are innumerable cloud storage vendors like Amazon, Microsoft, and others. Each provides its cloud storage services. Cloud-based storage systems are used to store structured, semistructured, and unstructured data.

Every cloud storage has its own data manipulation mechanism, authentication mechanism, data handling and encryption techniques. Data handling can be done using a Cloud

Console Platform (Graphical User Interface), REST (REpresentational State Transfer) APIs (Application Programming Interfaces), or by designing its API to communicate with the cloud. The available storage systems are divided into three parts based on storage architecture.

- Bucket Based Object Oriented Data Storage System: A bucket acts as a data storage container for storing data files as an object.
  - Amazon S3: Amazon S3 [82, 8] follows this type of architecture to store data. Using the rest API based web service. data of the bucket is edited, retrieved, or deleted. A bucket name is associated with an URL for performing data service-related read, write, and delete operations. For example, if the bucket name is reader, than the URL will be http://reader.s3.amazonaws.com. For a specific region, the bucket name is unique, which acts as a data service controller and permits the account that can use this bucket of architecture to store data. For a specific region, the bucket name is unique, which acts as a data service controller and permits the account that can use this bucket.

    In this container, data contents are stored as objects in Amazon S3. This object is identified by a unique identifier known as an object key. It contains metadata. Bucket Oriented Object Based Data Storage System stores data objects as flat files simplifying data organization. This simple methodology helps to support data scalability by adding nodes. However, this type of storage system is unsuitable for multitasking relational-based online data analysis tasks.
  - CACSS: Authors of CACSS [103] were concerned about the interoperability among cloud storage vendors. For this reason, they used a bucket as the primary container to store data objects. Here bucket name is unique, and the object index is the composite value of the bucket name and object key. Google Cloud Platform provides cloud storage named Google Cloud Storage. This storage system considers reliability, group-based access controls, backup, and restore functionalities. The storage architecture is bucket-object oriented. RESTful API is used to communicate with this storage system using JSON, or XML API. The GET verb is used to retrieve the data object; the POST verb is used for storing new data object with its metadata and the DELETE is used for deleting an object with its metadata.
  - Google Cloud Storage System: It provides an object storage solution where the data are stored in the form of objects present in buckets [104, 105, 106]. Each bucket is associated with a unique predefined key that is used for the protection of data. Every data in the bucket is identified uniquely by the URL in the form of "https://storage.googleapis.com/bucket_name/object_name". The file can be uploaded manually and through JAVA client Google APIs, and REST APIs.
- Account Container Based Object Oriented Data Storage System: Using this storage technique data are placed in a tree-based address space. This tree follows from the account, and container of the data object. The container holds a specific storage policy for storing data in the particular container, and the data operations (i.e., update, replicate, delete) corresponding to the stored data object follows this policy. The

client request hits the account, and if the storage policy is justified, the request is responded to with the corresponding data object.

- OpenStack Swift: Swift [107, 108, 109] supports account container based object storage system by building rings. The account, container, and data object have a separate ring. Mapping is done based on zone, partition, replication, and devices. The ring is responsible for providing swift services. Rest API (GET, PUT, DELETE) is used to communicate with the data storage system. Metadata holds the information of the data object. The minimum number of replication is 2.
- Rackspace: Rackspace [15, 110, 111, 112] supports account container based object data storage architecture similar to OpenStack Swift and file based storage architecture. Web applications communicate with Rackspace using RESTful APIs. Content Delivery Network (CDN) [113] is used to make the tasks of storage and data file sharing of the object storage system faster. Here, a dedicated container is needed, which is assigned in CDN, and then the stored file is operated by the generated web-ready URL.

  Rackspace stores an data object by making three copies of the data object, making it more reliable and available to the user. Through object storage architecture, large data files, and media files can be stored. The user needs to pay only for the storage spaces which he uses.
- Windows Azure:Azure [114, 65, 115] supports four types of data storage services: Table storage, File Storage, Queue Storage, and Blob Storage. Blob storage services of Azure are known as object storage. It focuses on availability, scalability, and durability properties. Azure supports an auto-partitioning system for automatically adding resources to handle massive data traffic. It gives users the facility for payment only when the resources are allocated to them. After the creation of the account, all storage services are accessible. The data object is placed in the container for blob or object storage service. By blob or object storage service, Azure stores only unstructured data (e.g., media files, documents). Azure also has a table storage service for structured data.

• Distributed Object Oriented Data Storage System: In this storage system, specific computing units build the cluster, and data are stored as objects [116, 117]. Flat file organization is used to locate the data storage space. The advantages of this type of storage system are that it is highly available, reliable, and does not have a single point failure. More than one Object Storage Device (OSD) is used as a storage disk, and a single monitoring unit configures or manages the data distribution in the devices.

  - Ceph: It uses Reliable Autonomic Distributed Object Store (RADOS) for supporting distributed object storage features [118, 119]. For deciding the storage space of data object into ceph cluster, CRUSH algorithm is used. The cluster node is placed in a different zone. The number of replication copies of the data object is more than one and distributed in such a manner that no duplicate copy of the data object is stored in the same cluster.
  - Sheepdog supports durability by dividing the virtual volume of data into multiple copies, and each copy is stored in several servers [120, 121]. The object

Table 2.1: Performance comparison between four database systems

Cases	PostgreSQL	MongoDB	OrientDB	Neo4j
Initial creation of schema	100ms	150ms	538ms	NaN
Insertion of one data item	81ms	97ms	52ms	580ms
Insertion of N records together	144ms	2ms	9ms	135ms
Query with one filtering condition only	110ms	33ms	25ms	35ms
Query all records	81ms	1ms	27ms	27ms
Delete only one record	111ms	48ms	44ms	105ms
Delete all records	108ms	1ms	39ms	76ms

storage location is decided by hashing technology. The client can add the external disk of any volume with a sheepdog gateway to handle the increased data volume of the collected data. Data recovery is made in case a server crashes. Here, zookeeper [122] is used as a monitoring unit. According to the pgbench, the sheepdog is better than ceph.

2.4.4 Comparative Study

Big Data is a combination of structured and unstructured data. Due to the rise in unstructured data, NoSQL databases have become a common storehouse of such data. For experimentation, four different NoSQL databases have been chosen. (a) MongoDB stores document-oriented dataset. (b) Neo4j follows the graph model. (c) OrientDB, which follows a hybrid model, is a bridge between the document database and graph database. (d) An extension of the relational database management system. Performance comparison is shown in Table 2.1.

We have evaluated the performance of the four databases on seven different criteria:

- Time required for creating Table or Collection or Class or initial creation of schema; 2. Insertion of one data item;
- Insertion of N records together;
- Query with one filtering condition only;
- Query all records (total count=21); 6. Delete only one record;
- Delete all records.

The above cases have been executed in the four databases using their respective query language. The results have been represented graphically in the form of 2D lines. Time in milliseconds is given along the Y-axis, and along the X-axis, the name of the database is given. For PostgreSQL, the standard Structured Query Processing Language is used, and for Neo4j, Cypher Query Language is used (Panzarino, 2014). Let us now look into the case studies.

Case 1: The initial creation of the outline structure of Table/Collection/Class/Label:
This covers the initial design of tables (or equivalents) using their respective query language. Table 2.1 shows the time taken for each database. It is evident from the table that OrientDB takes much greater time than others in this case:

1. **PostgreSQL:** The SQL query that is required to create a table is “create table student_db (name text, school text, class int, roll int, age int)” where “student_db” stands for the name of the table created to accommodate data along with columns names: school, class, roll, and their respective data types (e.g., text, int);
2. **MongoDB:** To create a database for MongoDB, the query used is “use student_db”. Here “student_db” is the database name, which is the actual container of collections that will hold data. To create a collection, query is used as db. CreateCollection “Student” where Student is the collection name that will hold the documents or records. It is worth mentioning that if no collection is created separately, MongoDB creates a collection automatically and names it the same as the name of the database;
3. **OrientDB:** In the case of OrientDB, first, a class needs to be created which extends either predefined Vertex class V or edge class E. The query used is “create class student extends V” where “student” is the name of the class. The time taken for creating the class is 439 msec. After creating the class, properties need to be created, which are class attributes. The properties can be made using a query as well as by using options that OrientDB Studio provides. The following query is used to create properties in this article: “create property Student.name string” where “Student” is the name of class extending vertex class “name” and type is string. The time recorded in this case is 62 msec. In this way, other properties like school, class, age, and roll each of string type have also been created; “Neo4j:” For Neo4j, separate creation of labels is not required since the label can be specified while creating the nodes.

Case 2: Insertion of First Data or Record:
Insertion of first data always takes greater time than the subsequent insertions due to initialization and memory allocation. Case 2 analyses the first insertion of a record in each of the databases. The execution time is shown in Table 2.1. Neo4j takes a much greater time in this case to insert the first data:

1. **PostgreSQL:** The SQL query “insert into student_dbvalues(‘Ram’, ‘DPS’, 9, 40, 13)” inserts a row or tuple in the table student_db. The entire relation or row represents the data in the database.
2. **MongoDb:** The query used to insert data is “db.student_db.save ([“name”:“Shyam”, “school”:“CPS”, “class”:9, “roll”:20, “age”:12])”. Also, insert() function could have been used instead of save(). In this case save() and insert() works in the same way. However, when id is passed as a parameter in save(), it performs update if document already exists and inserts if not. insert() will never perform an update operation and it will throw an error.
3. **OrientDB:** To create a record in OrientDB, class names always need to be mentioned. The query “create vertex Student set name=‘Ram’, school=‘DPS’, class=9,

roll=40, age=13" creates data in JSON structure with key and value pair. Since the Student class extends the Vertex class, the data created will be of Vertex type. Graph model features are also available in the studio, and the same data can be visualized as a node from the graph editor.

4. **Neo4j:** The Cypher Query Language "create(ram: Studentname:"Ram", school:"DPS", class:9,roll:40,age:13)" is used to create a node named "Ram" which belongs to label Student. The key: value pair denotes the properties of the node.

Case 3: Insertion of N records together:
This case is used to simultaneously analyze the time taken to insert n-records, n is set to 4. More or less, the same query structure is used as for case 2 with the only exception of a customized javascript function that is used to insert data in OrientDB. Table 2.1 lists the time taken for insertion of 4 records. PostgreSQL takes much greater time than the other three databases to insert a collection of records.

1. **PostgreSQL:** The query to insert 4 tuples in student_db is "insert into student_db(name, school, class, roll, age) values('abc', 'CPS', 9, 20, 12), ('def', 'JPS', 10, 10, 16), ('ghi', 'HPS', 10, 50, 15), ('jkl', 'IPS', 8, 70, 14)" where the value in each "()" denotes the tuple or record to be inserted;
2. **MongoDB:** save() function can be used to insert 4 document records like "db.student_db .save(["name":"rita","school":"JPS","class":10,"roll":10,"age":16, "name":"mita", "school":"HPS","class":10,"roll":50,"age":15, "name":"sita","school":"IPS","class":8, "roll":70,"age":14,"name":"amy","school":"CPS","class":9,"roll":20,"age":12])". In this case, insertMany() function can also be used to insert multiple data.
3. **OrientDB:** In this experiment, a javascript function is used to create multiple records. The input parameters are specified, taking input and storing it in variables. The variables are passed as values in the key-value pair generated for every document record.
4. **Neo4j:** Like in previous case, the same Cypher Query is used: "create (shyam:Student na me:"Shyam",school:CPS,class:9,roll:20,age:12),(rita:Studentname:"Rita",school:"JP S",class:10,roll:10, age:16), (mita:Student name:"mita", school:"HPS", class:10, roll:50, age:15), (sita:Student name:"sita", school:"IPS", class:8, roll:70, age:14)" where the value in each "()" denotes the node to be created. Since the name of the label is mentioned only once, all nodes belong to the same label "Student".

Case 4: Query with one filtering condition only:
Similar to Case 3, more records have been inserted into the database. The total count of the papers here is 21. Table 2.1 shows the time taken for selection based on one filtering condition. It has been seen that PostgreSQL takes a much greater time than others:

1. **PostgreSQL:** Here, the query is filtered on the basis of "names". The query "select * from student_db where name='Ram'" retrieves the required dat.
2. **MongoDB:** In this database, find() function is used to retrieve records. For example, "db.student_db.find (name:"Ram")" is the corresponding query for the required operation.

3. **OrientDB:** In case of OrientDB, the query is the same as that in PostgreSQL. The query is "select * from Student where name='Ram'".
4. **Neo4j:** The Cypher Query Language to filter with condition must include the name of the label to uniquely identify the node. The query is: "match(Ram:Student) where id(Ram) in [177] returns Ram" filters on the basis of id (assigned at the time of creation) which uniquely identifies the node.

Case 5: Query all records (Total record count is 21)
This case analyses the time taken to query all records present in the database. Table 2.1 lists the time taken in this case for selection of all records and PostgreSQL takes a much greater time than others:

1. **PostgreSQL:** SQL Query is very similar to that in Case 4 but only exception is that it does not have any filtering condition i.e., "select * from student_db";
2. **MongoDB:** MongoDB uses find() function for case 5 in the form of "db.student_db.find()";
3. **OrientDB:** OrientDB provides three kinds of view of data. It uses the query "select * from Student" where one can view the raw data in document-based format along with graphical representation and tabular view;
4. **Neo4j:** For a graph database, a variable x is defined which is required to retrieve all the nodes e.g., "match(x:Student) return x".

Case 6: Delete Only One Record:
This case analyses the time taken to find a particular record and to delete that record. Table 2.1 lists the time taken for deletion of 1 record. PostgreSQL takes much greater time than others for deletion:

1. **PostgreSQL:** SQL query for deleting one record is "delete from student_db where name='Ram'";
2. **MongoDB:** For selective deletion we need to pass the name and value pair as parameters to remove() function like "db.student_db. remove(name:"Ram")";
3. **OrientDB:** The query in this case is relatively simple by just mentioning the condition in a where clause as in case of SQL e.g., "delete vertex Student where name='Ram'";
4. **Neo4j:** The delete query for Neo4j is "MATCH (n name:'Ram') DETACH DELETE n", where Detach Delete is used to delete a node irrespective of edges connecting them.

Case 7: Delete All Records:
This case demonstrates the use of a query to delete all records without deleting a table, collection, or Class. Table 2.1 lists the time to delete all records from the database. PostgreSQL takes a much greater time than other databases:

1. **PostgreSQL:** Query in this case is same as in case 6 except the filtering condition. For example, the query coul be: "delete from student_db";

2. **MongoDB:** The same remove() function is used to delete all the records. Only difference with the previous case is use of empty which signify that all records needed to be deleted i.e., “db.student_db.remove()”;
3. **OrientDB:** Deletion of class Student deletes all records. However, the class still remains and the corresponding query is: “delete vertex Student;”.
4. **Neo4j:** The Cypher query “MATCH (n:Student) DELETE n” deletes all the nodes with their properties.

From the experimental analysis, we see that the initial setup cost of Neo4j is more, yet the cost of select, insert, and delete operations is the costliest for PostgreSQL. MongoDB and OrientDB seems to be the most effective even when for structured data. From the experiments, we can see that Neo4j has a slight edge over PostgreSQL.

Chapter 3

Storage Model

3.1 Introduction

Nowadays, billions of digital things generate a massive amount of data, known as big data. Big data storage is difficult due to its volume, velocity, and variety properties [123]. Diverse types of stakeholders and their applications are associated with a data storage system. During data processing, these applications receive and generate a vast amount of data in different formats (e.g., text, document, image, video, audio, etc.). The responsibility of the database manager is to provide data storage support so that the application can efficiently run its job. Going beyond the traditional data storage system, the storage manager use different cloud data storage systems [124, 125]. In this case, allocation of storage resources is performed based on their demand. This technique helps to support scalability, reliability, and durability but fails to support data variety.

3.2 Research Challenges and Solutions

There are a variety of data storage systems. The services provided by these storage systems vary from one another. Therefore, selecting a storage unit is a challenge and the selection depends on the purpose. Some storage systems provide services according to the resource capabilities (e.g., feature, performance, cost, etc.) like an automated approach [126]. Some platforms convert raw data to the application-specific data structure like MVaaS [127]. It is even more challenging to manage a storage systems that attempts to handle the variety property of big data.

We aim to find out a reasonable solution to overcome such challenges.

- We have proposed a storage model to allocate storage resources at run time. Storage resources are assigned based on users' demand. Two features are considered for such an assignment: resource capability, and application feature.
- We aim to make the storage system flexible enough to support application demands. Therefore, our proposed storage model provides such facilities where users can design the storage resource architecture according to their needs.
- We demonstrate the applicability of the proposed storage model using a single user-defined data storage request. This request holds a set of data types in any sequence. The purpose of this request is to store multiple types of data through a single interface.

3.3 Background

As mentioned earlier, there are multiple types of data storage systems. Every system has a set of functionality for storing the collected data that differ from system to system. As an example, a column family-based data storage system, Cassandra, stores the data in key-value-based table form [128]. Document oriented database, MongoDB, stores data in json based document format [129].

It is seen that XML-based model makes the storage system efficient to support applications' needs. In [101], the author automatically selects jobs of the cloud storage service based on resource capabilities using an XML schema. T information related to resource capabilities is provided by cloud users. Author of paper [130], selects infrastructure-based services (e.g., bandwidth, CPU cost, latency, etc.) by considering the relation between functional and nonfunctional configurations provided by users (e.g., CPU type, memory size, costs, regional availability). Cloud service selection (e.g., web service) is based on cloud capabilities and human requirements. All these storage models focus on the resource capability for selecting storage services. The author in [127] proposes a framework where data storage structure depends on ways to make applications faster. We aim to support the variety property of incoming data by storing them in user-assigned storage spaces. For this reason, allocation of storage resources is done based on the type of the incoming data.

User willingness has a significant impact on the services selection process. Author in [131] makes a survey of the Chinese market about the users switching from one cloud storage service to another cloud storage service. For this reason, he considers some nonfunctional factors, related to human nature like risk, trust, switching costs, and social influences. In this work, we also consider the user willingness, but it depends on the input data type and the corresponding database system. Author in [132] thinks that the opinion of users regarding getting services depends on the quality of service. This selection procedure is accomplished by ranking cloud service s after opinion mining of users and applying multi-attribute decision-making models on quantitative (user's review) and qualitative data (QoS parameter). However, this model is not good for our purpose as the selection of storage services is dependent on quality and not on type of the input data.

3.4 Modelling Components

The first task is to collect storage-related information. Such information is collected from a set of components mentioned in Figure 3.1. These storage modeling-related components are considered for allocation and configuration. These components also provide a structural representation of data storage units. The relationship between the components is presented in Figure 3.1, which describes how one component affects another component. We have mentioned a set of components that directly affects the allocation of resources and helps to sketch a structural representation of such allocations. In this case, storage resources are nothing but the set of data storage devices controlled by available databases, like, Cassandra, SQL, MongoDB, Neo4j, HDFS, etc.

Resource Allocation Feature: These features are about requirements which enable us to allocated resources. For example, storage cost, throughput, storage space size, storage data information needed for the application, etc., are either user-defined or application-defined. Designing an optimized and friendly data storage system depends on specific points like storage space, usage pattern, storage cost, storage response time, etc., [133, 134, 135].

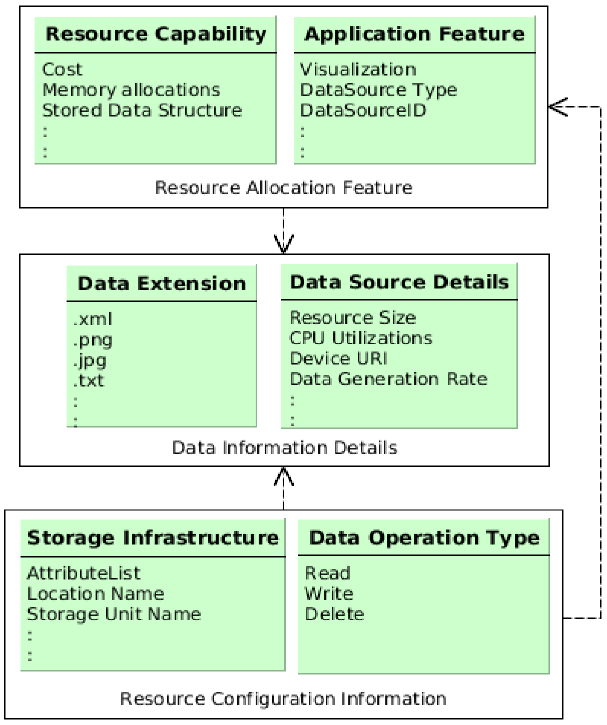


Figure 3.1: A Diagrammatic Representation of Information Of The Modeling Components And their Relationship Using UML

Property values of this component depend upon the data Information details described below. Such information is divided into two parts:

1. Resource Capability: Under this sub-component, resource structured related requirements are mentioned. It mentions the lower-level architecture-related information of the storage resources, like storage memory size, retrieval time, etc.

2. Application Feature: This sub-component define the usage patterns. Different applications or different users require various representation of the raw data. Users prefer to view the generated data in graph or file format. Applications prefer to receive the raw data in a data format suitable for these applications so that computation cost becomes low. [136].

Data Information Details: Without a complete knowledge about the collected data, design of a storage system cannot be very effective. This component mentions properties which help to understand the collected data, like data type, format, source types, generation frequency, etc. This information helps to decide which data storage device units are suitable for storing such data. Based on given information type, information about data may be categorized in two ways.

1. Data Source Details: Source-related information dramatically contributes to understanding the characteristics of the collected data. Data source information helps to

understand the data generation frequency, data volume, generated data types, etc. Depending on this feature, appropriate resources must be allocated by the data storage system [137].

2. Data Extension: In the case of big data, the storage system has to store a variety of data. [138]. Data extension helps distinguish one type of data from other type of data. Each data extension has a specific meaning which helps us to understand the nature of the data. Also, the output generated by these applications are stored using a specific data extension. Conversely, every data storage device cannot store any type of data. However, data extension type helps to decide the corresponding data storage device units.

Resource Configuration Information: For storing data, a storage system needs to arrange the resources in a proper manner. Supporting the data operations like reading, writing, and deleting are also part of a data storage system that stores raw data. Therefore, resource configuration plays a crucial role in organizing data so as to minimize the cost of future computation. Storage resource configuration focuses on specific points like data structure, data operation, and storage device unit. The assigned value-set of this component depends on the data information details and resource allocation features. This component points out the set of actions that are needed to configure the resources. Based on actions, this information list can have two categories.

1. Storage Infrastructure: This sub-component mentions information, which helps to sketch the storage architecture. For example, storage location, device unit name, attribute list, database name, etc., help define the storage schema. The storage schema structure varies with data type and resource type.

2. Data Operation Type Any storage system has to support at least three types of data operation: read, write, and delete. For each operation, necessary storage structure definition is needed.

3.5 Modeling Elements

In this section, we describe the structural representation of the storage model. Extensible Markup Language is chosen to describe the storage model because it is machine-understandable and user-friendly. Each XML-based storage model has a specific identifier mentioned as mentioned in 3.1 so that a user can uniquely identify it.

Listing 3.1: Example of Storage Model identifier

```
<storagemodel id=<unique id value of storage model>>
:
</storagemodel>
```

The modeling components mentioned earlier in this section are presented in the XML-based storage model using a few elements. These elements have specific meanings to the storage system. They act as control units to identify storage resources needed for the incoming data, map with the user-defined information, and allocate corresponding resources to store this incoming data. These elements are described based on their functionality:

1. Database Element: It is used to define the data storage structure of the data storage units for the incoming data. This element is responsible for configuring allocated storage resources. The XML tags of this element present the storage resource architecture.

   Listing 3.2: Example of database element

   ```
   <database id="flag1">
     <account>U009</account>
     <container>U009_1234</container>
     <primarykey name="time" type="long"/>
     <attribute name="Data" type="int"/>
     <attribute name="DeviceType" type="String"/>
      <databaseURI>http://<machine IP>:<port number>/
           cassandraDB/rest/store</databaseURI>
   </database>
   ```

   Listing 3.2 presents an example of data storage resource configuration. In this example, <account> tag is used to mention the name of the database space (i.e., U009), <container> mentions the name of data tuple space (i.e., U009_1234), <attribute name="Data" type="int"/> and <primarykey name="time" type="long"/> define the name of the attribute (i.e., Data) and name of the primary key (i.e., time) along with the corresponding data type (i.e., int, long). The storage system can communicate with resources through the rest API. For performing individual data operation, there is a specific URI which is mentioned as follows. <databaseURI> (i.e., http://<machine IP>:<port number>cassandraDB/rest/store). Each resource is identified by a specific identity value presented under <database id> tag.

2. Property Element: Cassandra is a storage resource in Listing 3.2. To configure Cassandra as a storage space we need some specific information like Keyspace (presented as account), column-family (presented as a container), attribute list (presented as an attribute), primary key name (presented as primary key), and column data type (presented as a type) are used to configure Cassandra as a storage space. Here, Cassandra is identified as flag 1

   In the XML based storage model property element is written in <property> tag, shown in listing 3.3. In this listing, the "number of visualized components" is an application feature used as a resource allocation feature. In this case, storage resource is selected based on their output representation. For example, MongoDB stores data as a document; so, the number of visualized components is one. Cassandra stores data in a column family way; this implies the number of visualized components is equal to the number of columns (i.e., more than one number of component).

   Listing 3.3: Example of property element

   ```
   <property>
   ```

```
number of visualized component
</property>
```

3. Operation Element: Data storage system supports more than one type of data operation (like, READ, WRITE, DELETE), and their working are different from each other. Based on the operation type, resource configuration has to vary. This element represents the content of the data operation type component.

   Each storage model mentions only a single data operation component, enclosed within <operation> tag.

   Listing 3.4: Example of operation element

   ```
   <operation>
   write
   </operation>
   ```

4. Strategy Element: After configuring storage resources, a system must allocate them based on certain conditions. Every resource is not capable to support any job; it has limitations. For performing specific jobs, we need to allocate an appropriate resource. The storage model defines this requirement using an XML element called strategy element.

   Listing 3.5 represents the example of strategy element. The instructions are put under the "strategy" tag. There may be more than one strategy content in a single storage model. Each strategy is identified through its id. The condition is written inside the condition tag. What type of action is taken is based on the mentioned condition is written in the "output" tag. In the example listing, we mentioned that if the number of visualized components is greater than 2, the flag is set to flag1 means Cassandra is selected as a storage resource.

   Listing 3.5: Example of strategy element

   ```
   <strategy id="st1">
     <condition>number of visualized component> 2</condition>
     <output>setFlag=flag1 </output>
   </strategy>
   ```

3.6 Testing Platform

In this chapter, we focused on storing big data that handles the variety property. When a single data store system is used to provide storage services to big data, and more than one type of data is collected, the system fails to store data and provide services in an effective manner. For handling this challenge a storage model has been proposed in which multiple types of data can be stored in their appropriate storage resources.

Listing 3.6: Configuration of Storage Model

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
```

```
 <storagemodel id="90">
  <database id="flag1">
  <account>U009</account>
  <primarykey name="time" type="long"/>
  <attribute name="Data" type="int"/>
  <attribute name="DeviceType" type="String"/>
  <container>U009_1234</container>
  <databaseURI>
        http://<machine IP>:<port number>/cassandraDB/rest/store
   </databaseURI>
</database>
 <database id="flag2">
   <account>U009</account>
   <container>U009_1234</container>
  <databaseURI>
        http://<machine IP>:<port number>/mongoDB/rest/store
   </databaseURI>
</database>
<property>Data Type</property>
<operation>write</operation>
<strategy id="st1">
  <condition>T</condition>
  <output>setFlag=flag1 </output>
</strategy>
<strategy id="st2">
  <condition>D</condition>
  <output>setFlag=flag2 </output>
</strategy>
</storagemodel>
```

3.7 Resultset

We consider a use case that helps to highlight the usefulness of the proposed storage model. In this use case, we consider a user who wants to store two types of data, document, and text. He makes a data storage request where the incoming data type is either "D" or "T", where D represents document type of data and T represents text type of data. The incoming data can appear in any sequence, like "DDTT", or "TDDTD".

To support such a situation, we take help of a storage model, which is configured in such a way as presented in the listing 3.6. Cassandra is a storage resource for text-oriented data, and MongoDB for document-based data. Each storage resource follows its own architecture to store data. The data storage configuration of each storage resource is mentioned in the storage model. This storage model defines the a condition that decides under which a specific resource becomes active.

3.8 Discussion

The proposed storage model works in a distributed system and each storage unit is accessed through rest APIs, Thus, they become similar to storage resources. For experimentiation, we choose two types of data storage units as storage resources: column family-oriented database Cassandra, and document-based database MongoDB. We use 2 $\times$ m1.large in-

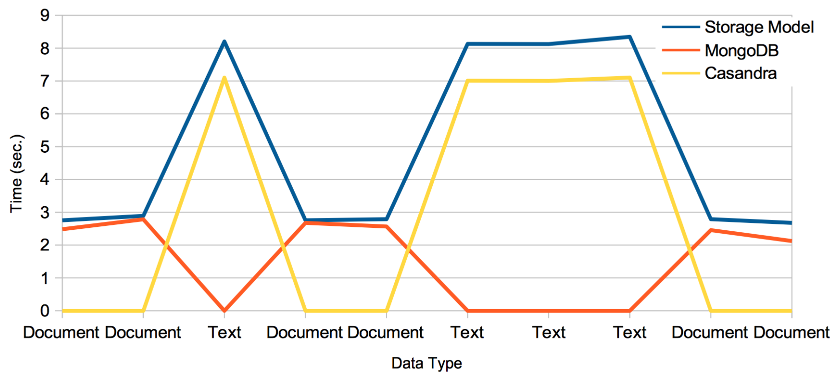


Figure 3.2: Applicability of a storage model

stances with 8 GB memory to support the storage resource configuration. One instance is assigned to Cassandra and another to MongoDB.

In Figure 3.2, we present the pictorial representation of the facility of applying the storage model in a storage system. We consider that the occurring data format sequence is "DDTDDTTTDDD". The storage model structure is already defined in the previous subsection. When the data type of incoming data"D" is "D", element content of <condition>, present in the storage model, then the storage system allocates the storage resource whose <database id> is flag 2. In the same way, when "T" occurs <database id=flag 1> will be true and appropriate resource is allocated.

The figure 3.2 represents the significant difference between two situations. In one, storage model is properly designed and in another storage model is not properly designed in the storage system. In the absence of a proper storage model, when "T" meets MongoDB, it does not perform any task; when "D" meets Cassandra DB, it is in an idle position. The presence of a storage model makes the storage system active in both situations. Also, using this storage model, users can provide a degree of flexibility to the storage structure in the storage system.

3.9 Conclusion

The proposed XML-based storage model helps the user to inform the storage system to identify which storage resource should be allocated under different circumstance at run time. Also, it mentions about storage schema structure of prevalent in different run-time conditions. Thus, the storage system becomes more flexible and less complex. Using this storage model, a user can create a storage system for different conditions (by means of conditions). An user does not need to set up the storage system for handling different types of input data. The storage model takes that decision automatically. Also, a user does not need to change the storage system structure every time to support a new condition.

Chapter 4

Load Balancing in cloud platform

4.1 Introduction

According to the definition of cloud computing by the National Institute of Standards and Technology (NIST) [139], it is a model where on-demand computation services can be performed via internet. A shared pool of computing resources (e.g., networks, servers, storage, applications, and services) exist to provide infrastructure-based services. This online way of computing is accomplished by acquiring and release of resources according to the needs of a service.

Due to the increasing demand for computing services, the load on available resources becomes enormous. The cloud service provider can control the combination of load and resources by distributing the load to a limited number of resources in an efficient manner. This task is known as *load balancing technology* [140, 141] in the literature.

Load balancing algorithms distribute incoming job requests or network traffic among the resources available in the resource pool. The main objective of a load balancing algorithm is to enhance the system's performance by optimizing resource utilization, response time, throughput, and to avoid overload or under-load situation of resource [142, 140].

4.2 Background

There are two types of load balancing algorithms: static load balancing algorithm [143] [144] and dynamic load balancing algorithm [145]. The difference between static and dynamic load balancing algorithms arises at the time of resource scheduling. The static load balancing algorithm does not consider the present status of resources for resource selection, but the dynamic load balancing algorithm considers the present status of resources for resource selection[146]. Such dynamic resource selection approach increases decision accuracy and improves the performance of a dynamic load balancing algorithm [147]. Static load balancing algorithms are more stable than dynamic load balancing algorithms. Dynamic load balancing algorithms are based on ant colony [148], honey bee [149], etc. Round robin [150], Random [151] [152], and Threshold [153] are some examples of static load balancing algorithm.

Due to dynamic workload in a cloud environment, some resources are underutilized and some resources are over-utilized. To attain load balancing, we have to avoid needless resource consumption, and try to achieve efficient resource utilization through the concept of the live VM migration [154]. The technique of migrating VM from one host to another physical host allows a user to remain connected so as to cause the minimum down-time. Using this live VM migration technique, the VMs from the overloaded hosts may be migrated to under-loaded hosts. If we migrate under-loaded hosts to moderately loaded hosts, we may fee some the hosts which can be switched off. However, such over consolidation of VMs in a host results in degraded system performance.

4.3 Motivation and Research Problem

Cloud computing provides computing resources as a utility based on Service Level Agreement (SLA) between users and their cloud service provider. The Amazon EC2 [27], Google App Engine [155] and Microsoft Azure [1] are a few major cloud service providers that provide Platform, Infrastructure, and Software oriented services.

To increase the overall performance of cloud computing, one needs to distribute incoming load efficiently among various nodes in the cloud computing environment. For this purpose, researchers have proposed various load balancing algorithms such as Throttled Load Balancing Algorithm [156], Active Monitoring Load Balancing Algorithm [157], and Round Robin [158, 159] algorithm. These algorithms are more or less effective in accomplishing the task task of load balancing.

Still there are some research questions which need to be .

- Can we consider role of resources available to the provider to take decisions while balancing load?
- Can task scheduling increase the overall performance of a cloud computing platform?
- Nowadays, cloud computing platforms consume a lot of energy to run their system. Can we reduce this energy consumption by using a suitable load balancer?

In the following, some solutions are proposed to address the research questions mentioned earlier.

- A novel dynamic load balancing algorithm named dynamic resource service quality-based load-balancing algorithm is introduced. It helps a load balancer to support diverse rate of incoming requests from the applications to the database. The load balancer selects the proper resource from the resource pool based on the provided data service quality at the time of task scheduling to achieve this goal. To judge the service quality, we consider the present status of specific resource metrics. These metrics include CPU clock frequency, load in the last 15 minutes, number of processes, number of running processes, Number of CPU cores, free RAM size, free SWAP memory size, free disk size, bytes in and bytes out. Using these metrics, the load balancer avoids back-end resources which are already dead or in a bottleneck situation.
- An algorithm Double Threshold based Power Aware Honey Bee Foraging Load Balancing (DTPAHBFLB) has also been introduced, which is based on swarm intelligence-based technique [160, 161]. Specifically, it is based on Honey Bee Foraging Load Balancing algorithm [162, 163] for uniform distribution of loads within the VM's of cloud along with double threshold based power aware load distribution mechanism to reduce energy consumption.

[1]https://docs.microsoft.com/en-us/azure/storage/

4.4 Resources relevant for Load Balancing in the Cloud

Load balancing is a mechanism to utilize the resources efficiently to ensure the maximum throughput as well as the minimum response time. There are three essential resources in a cloud system: (i) network, (ii) Memory, and (iii) CPU. In the context of request-response processing, these three resources are equally loaded and affect the system performance equally. To improve performance metrics, one needs to balance loads of three resources. In the following, we present a view of the load balancing mechanism based on these system resources.

Network Load Balancing It focuses on network traffic through which users try to access a single resource. Network load can be balanced by controlling transmission protocols (TCP/IP) [164, 125]. There are different techniques to balance network load such as: (a) creation of multiple sessions where each session is dedicated to serving a particular type of job; (b) distributing network traffic in different clusters to different hosts to reduce network load of any particular host. Microsoft Windows Server 2008 [165] follows this technique. (c) controlling routing, network overflow and congestion may be prevented. Ad-hoc networks use this technique to increase throughput, decrease packet loss ratio, and end-to-end delay [166].

Memory Load Balancing When an application uses a very significant amount of memory (RAM) compared to the capacity of the server, the server becomes overloaded. In this situation, the server wastes considerable time by page swapping, which degrades the system performance. Memory load balancing mechanism [167] focuses on resolving the memory overload situation by customizing hardware and software. One of the most popular techniques, Storage virtualization [168, 169]. is used to design a memory load balancer. In such systems, there are single storage pools (consists of hard disk, optical disk, tape, etc.) where all are treated as single physical memory and controlled by a central console. IBM SAN Volume Controller (SVC) [170] uses such storage virtualization technique to balance memory load in the IBM data center.

CPU load Balancing When many clients request for one resource, and all clients go to waiting state, this situation is known as CPU over-load. This situation occurs in the cloud very frequently. It results in failure of the whole system due to low performance. There are hardware (i.e., multi-core CPU implementation) and software solutions (i.e., load balancing algorithms) for to mitigate CPU over load situations [171].

4.5 DRSQ-Dynamic Resource Service Quality Based Load Balancing Algorithm

In this section, we discuss a novel load balancing algorithm where our target is to manage task load and, side by side, provide adequate client services.

4.5.1 Algorithm

In this section, we describe a new load balancing algorithm named "Dynamic Resource Service Quality Based Load Balancing Algorithm" (DRSQ). Back-end resource selection is based on the service quality of the resources. Service quality is measured by considering the request type (e.g., READ, WRITE) and the value set of resource metrics.

DRSQ works in three phases: (a) Identifying the type of application requests, (b) Configuration of the resource list, and (c) Identifying the most appropriate resource. The detailed algorithm is described in Algorithm 1.

Algorithm 1 Dynamic resource service quality based load balancing algorithm

```
Input:  Application Request (READ/WRITE) operation
Output:  Backend resource which is the best to handle the request
  Phase 1: Identifying the type of request
    Phase 1.1: Wait for the incoming request
    Phase 1.2: Register New Request
    Phase 1.3: Fing the type of request which either can be READ or WRITE
  Phase 2: Configuration of Resource LIST
    Phase 2.1: If the operation to be performed is READ
        Phase 2.1.1: Select and store a back-end resource with maximum CLOCK_SPEED,minimum LOAD in the last 15 minutes, minimum number of processes, minimum number of RUNNING processes, maximum Number of CPU CORES respectively in a LIST
        Phase 2.1.2: Append the Server name in the LIST after all the parameters are checked
    Phase 2.2: If the operation to be performed is WRITE
        Phase 2.2.1: Select and store a back-end resource with maximum FREE RAM, maximum FREE SWAP, maximum FREE DISK, minimum LOAD in the last 15 minutes respectively in a LIST
        Phase 2.2.2: Append the resource in the LIST after all parameters are checked.
  Phase 3: Identifying the most appropriate resource based on LIST components
    Phase 3.1: Select the resource which has maximum occurrence in LIST
    Phase 3.2: Forward the request to the selected resource
```

4.5.2 Testing Platform

One aim of the load balancing algorithm is to control network traffic. This is also an objective of the proposed Dynamic Resource Service Quality based Load Balancing Algorithm. Figure 4.1 describes the experimental system setup. This setup is divided into two parts: *Data Zone* responsible for client requests and *Cloud Zone* presents the information about resources and the load balancer. In this experiment, four application domains are considered each consisting of temperature sensors. *Load Balancer* runs the DRSQ to select the "best" resource for handling network traffic generated by the clients' requests. In this experiment, we use real temperature sensor data. In this setup, the clients' requests are either reading or writing.

The efficiency of DRSQ is measured by increasing and decreasing the network data traffic. This effect is caused by varying the frequency of incoming data.

Six temperature sensors are used and each sensor's data generation frequency rate can be varied. In this setup (See Figure 4.1), one extreme is when each application domain consists of a single temperature sensor, and at the other extreme, each application domain holds two temperature sensors.

Five virtual machines and one load balancer machine are considered for the experiment. MongoDB [93], Apache Tomcat [172] and Ganglia [173] are used to configure the machines according to their needs as presented in figure 4.1. Ubuntu 14.04 is used as an operating system for these machines. Ganglia plug-ins are used like Gmetad 3.6.0 in the load balancer and Gmond 3.6.0 in the resource part for collecting the resource metrics value set corresponding to each machine. MongoDB 3.6.19 is used as a back end database to store the incoming data. Java based web applications are run on tomcat server 8.5.28.

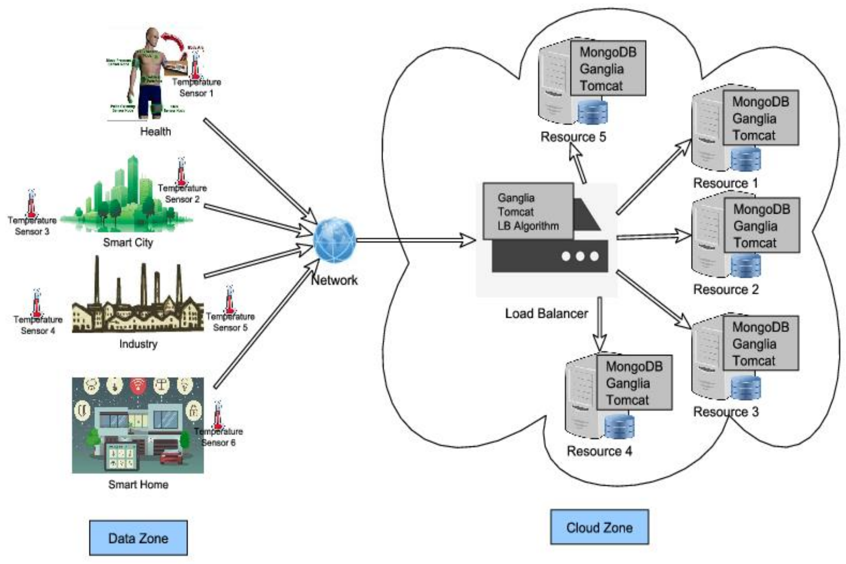


Figure 4.1: Diagramatic sketch of the Experiment Setup

Algorithm	Execution Time	CPU Utilization	Memory Utilization
Dynamic Resource Service Quality Based Load Balancing Algorithm	1.98 second	0.451%	6 MB
Round-Robin Load Balancing Algorithm	0.401 second	0.028%	5 MB

Table 4.1: Performance Results of Dynamic Resource Service Quality and Round-Robin Load Balancing Algorithm

4.5.3 Comparative Results

We choose Round-Robin load balancing algorithm as the base algorithm to compare with because it is considered to be the standard load balancing algorithm in many application domains. We first compare the overall performance (i.e., CPU Utilization, Memory Utilization, and Execution Time) between Round-Robin and DRSQ algorithms. We find that Round-Robin algorithm performs better in all aspects, as shown in Table 4.1.

To analyze the performance behavior in streaming data, we consider five data requests from each sensor of each application domain (Figure 4.1) which sends data requests in the following data rates: 5KB/sec, 10 KB/sec, 15 KB/sec, 20 KB/sec, 25 KB/sec and 30 KB/sec. In Figure 4.2, when the data frequency rate is changed, after some time, the response time changes in DRSQ algorithm. Otherwise, it maintains a constant value.. Figure 4.3 represents the characteristics of response times of each request under different frequency rates in a more elaborate way. Here, we can see that, in every frequency rate only for 'Request 1' (Figure 4.3(a)) DRSQ algorithm takes more time than Round-Robin, and in all other requests (Figure 4.3[(b), (c), (d), (e)]) DRSQ takes less time.

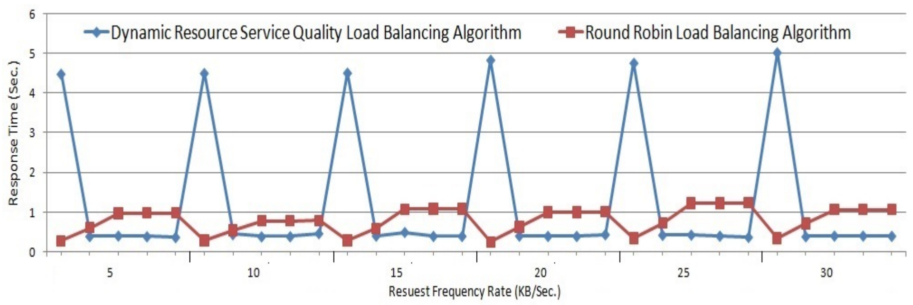


Figure 4.2: Pattern of response times for streaming data requests

4.5.4 Discussion

Dynamic Resource Service Quality algorithm selects a resource based on the monitored status parameters collected from ganglia-generated log files. Initially, the algorithm invests time to trigger the ganglia monitoring daemons to monitor the resources, connects with the back-end servers, fetches the log files generated by ganglia daemons and finally parses the desired values from the log files. When the load balancer receives the subsequent requests after some time, it modifies the generated log file contents, analyzes the new values, and sends the incoming requests to the selected resource. Hence, if ganglia initial setup time can be reduced, it further reduces response time.

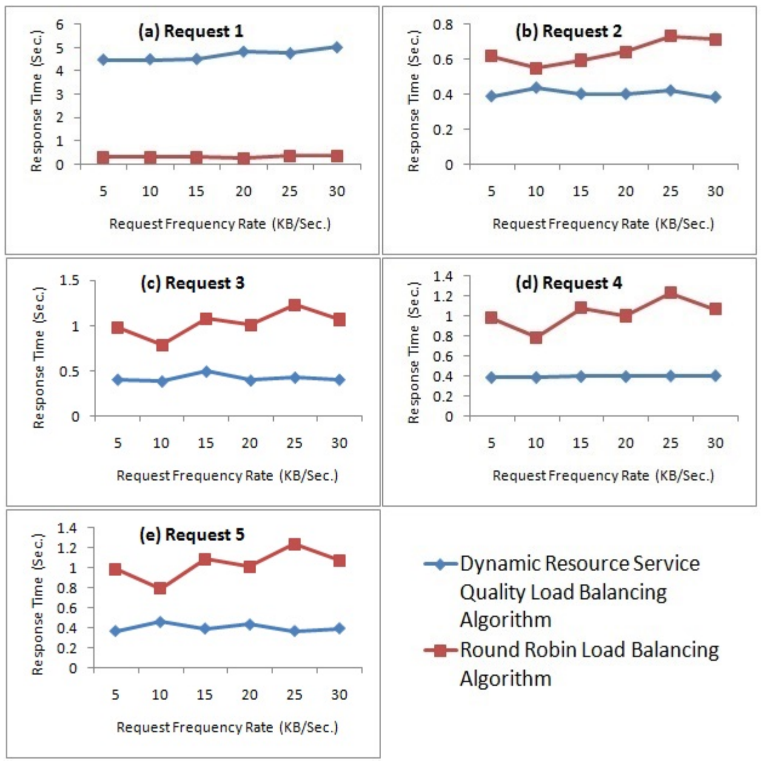


Figure 4.3: Characteristics of requests under different frequency rates

The Round-Robin algorithm circularly selects the resources, one after another. In such a situation, there are significant chances that a crashed server or a heavily loaded server is selected as many requests will be pending with it. Furthermore, all the requests may cause a bottleneck at the back-end server, which increases the response time. Therefore, sometimes response time is very low, and sometimes very high. For the DRSQ, there is no chance to select a resource in overloaded condition. For this reason, except for the initial set-up stage, response times show a more or less a very similar value.

Also, we use 'Task Accuracy Rate' to justify the usefulness of DRSQ Load Balancing algorithm. For this experiment, we changed our setup, as shown in Figure 4.1 a little bit where resource five is in dead condition. We continuously make 10,000 'Write' requests with a data bundle consisting of 1000 data records in each request. According to the working principle of the Round-Robin algorithm, requests will hit resource five 2,000 times, and it fails to serve those requests. Whereas, DRSQ does not select Resource 5 at all because before selecting a server, the load balancer checks whether the resource is in a workable state or not. If we assume 1% packet loss occurs due to network issues, the computed accuracy percentage for both algorithms is shown in Figure 4.4. For DRSQ, data loss occurs only due to network issues. However, for the Round-Robin algorithm, the inaccuracy rate arises due to both network faults and selection of crashed the resource.

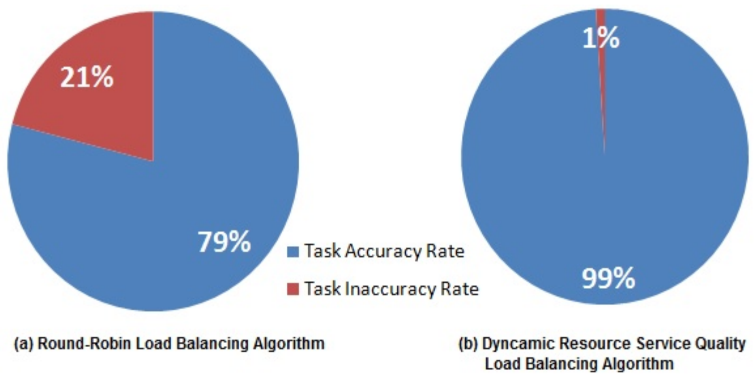


Figure 4.4: Task accuracy rate in percentile for (a) Round-Robin and (b) Dynamic Resource Service Quality Load Balancing Algorithm in mentioned scenario

We conclude that if we see the overall performance, the Round-Robin algorithm provides better performance for many applications but it does not provide satisfactory performance in task accuracy or performance rate for streaming data. However, in such situations, DRSQ provides good much better performance.

4.6 Double Threshold Based Power Aware Honey Bee Cloud Load Balancing Algorithm (DTPAHBF)

4.6.1 Algorithm

In a cloud computing environment, there are several data stores and virtual machines. These virtual machines have ID, CPUs, bandwidth capacity, memory, and processing power. Cloud is decentralized to make the system scalable and to avoid a single point of failure. The new algorithm uses the following parameters:

- The maximum number of jobs that can be allocated in VMs
- Number of currently allocated jobs in VMs
- Number of currently active VMs
- Virtual machine states list
- Lower and Upper cutoff value of modified honey bee algorithm

The new algorithm is based on an innate intelligent behavior of a honey bee swarm called foraging behavior. A new Double Threshold-based Power-Aware Honey Bee (DT-PAHB) Load Balancing algorithm that simulates real honey bees' behavior is discussed as a

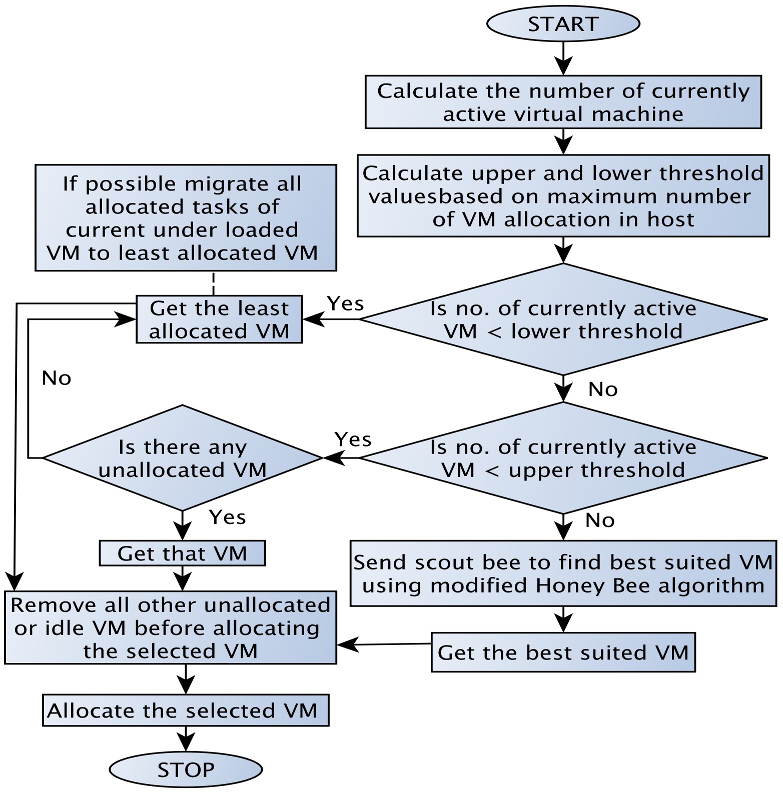

START
Calculate the number of currently active virtual machine
Calculate upper and lower threshold valuesbased on maximum number of VM allocation in host
If possible migrate all allocated tasks of current under loaded VM to least allocated VM
Get the least allocated VM
Yes
Is no. of currently active VM < lower threshold
No
No
Is there any unallocated VM
Yes
Is no. of currently active VM < upper threshold
Yes
No
Get that VM
Send scout bee to find best suited VM using modified Honey Bee algorithm
Remove all other unallocated or idle VM before allocating the selected VM
Get the best suited VM
Allocate the selected VM
STOP

Figure 4.5: Flowchart of proposed double threshold based power aware honey bee (DT-PAHBF) load balancing algorithm

solution to the problem of load balancing. The flowchart of the proposed double threshold-based power-aware honey bee (DTPAHBF) load-balancing algorithm is shown in Figure 4.5. This honey bee-inspired load balancing is based on a dynamic approach on double threshold values depending on the maximum number of virtual machine counts. The upper threshold value is 90% of the maximum count, and the lower threshold value is 20% of the maximum count. Once the tasks are allotted to the VMs, the current load is calculated. If the VM becomes overloaded, the task is transferred to the VM based on the currently active VM count for the lower and upper thresholds. Suppose the currently active VM count is less than the lower threshold value. In that case, the least loaded VM is chosen for task allocation instead of using a modified honey bee algorithm to reduce migration time and cost, storage cost, CPU cost, and memory cost and thus save energy. If the currently active VM count is greater than the upper threshold value and an unallocated VM is available, we choose that VM; otherwise, we choose the least allocated VM instead of using the modified honey bee algorithm to save time, cost, and energy.

When currently active VM count is within the lower and upper threshold value, i.e., within the normal range, then a double threshold-based modified honey bee algorithm is followed to get the optimized VM allocation of the task. The flowchart of the modified double threshold honey bee foraging algorithm is shown in figure 4.6. In the modified honey bee algorithm, if the number of VM allocations is within the normal range, i.e., within 20% to 90% of the maximum allocation count, scout bees will not be sent further searching for

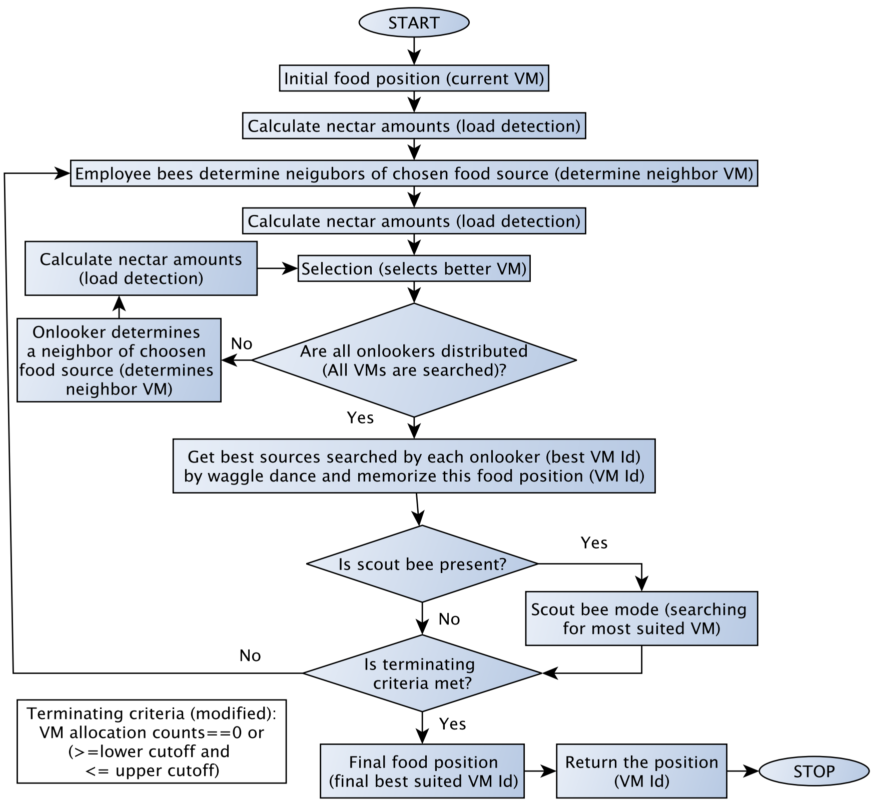

START
Initial food position (current VM)
Calculate nectar amounts (load detection)
Employee bees determine neigubors of chosen food source (determine neighbor VM)
Calculate nectar amounts (load detection)
Calculate nectar amounts (load detection)
Selection (selects better VM)
Onlooker determines a neighbor of choosen food source (determines neighbor VM)
No
Are all onlookers distributed (All VMs are searched)?
Yes
Get best sources searched by each onlooker (best VM Id) by waggle dance and memorize this food position (VM Id)
Is scout bee present?
Yes
Scout bee mode (searching for most suited VM)
No
No
Is terminating criteria met?
Terminating criteria (modified): VM allocation counts==0 or (>=lower cutoff and <= upper cutoff)
Yes
Final food position (final best suited VM Id)
Return the position (VM Id)
STOP

Figure 4.6: Flowchart of the modified double threshold honey bee foraging algorithm

food sources. Otherwise, it will select that VM. After the VM for task allocation is chosen, all unallocated or idle VMs are detected and removed from the host to reduce unnecessary energy consumption. If any host contains VMs that are all unallocated or idle, then the host must be shut down or sent to sleep mode to reduce colossal energy consumption. An idle VM consumes about 70% of energy over a fully utilized VM. Honey Bee forage technique hires a sub-urbanized load balancing methodology, and task transfer is assumed to be flying of the bee.

4.6.2 Testing Platform

Simulation Configuration

We performed a simulation experiment followed by performance analysis in the CloudAnalyst toolkit [174]. CloudAnalyst is designed to model resource scheduling algorithms, cloud service brokers, and cloud data centers. VM load balancer component of the CloudAnalyst is used to implement the load balancing mechanism. This simulator provides a user-friendly GUI to reduce complexities of programming. It allows parameter sweep to do the experiments by users. This CloudAnalyst framework allows users to set the regions for cloud-based user bases and data centers. Several other parameters can also be configured like: the number of requests generated per user per hour, number of user bases, number of VMs and number of CPUs, amount of storage and bandwidth of the network, and some other significant parameters as shown in Table 4.2.

Table 4.2: Parameter Settings for CloudAnalyst Simulation

Sl No.	Parameter	Value
1.	VM Memory	512 MB
2.	Data Center OS	Linux
3.	Data Center VM	Xen
4.	Data Center Architecture	x86

Based on the parameters mentioned in Table 4.2, the cloud analyst assesses a simulation run, and the results are presented in a graphical format. In the following, we discuss some statistical metrics based on which we have derived the simulation output and compared the performance.

1. Average Response Time of the system
2. Average Processing Time of the Data Center
3. CPU Cost of the Virtual Machine
4. Storage Cost of the system
5. Memory Cost of the system
6. Total Data Transfer Cost
7. Energy consumption of the overall process

The CloudAnalyst enables repeated execution simulation runs after varying parameters.

Experimental Setup

The parameters for the configuration of User Bases, Application Deployment, Data center, and Physical Hardware details of any Data Center are defined as given in Table 4.3, Table 4.4, Table 4.5 and Table 4.6, respectively.

Table 4.3 shows that six distinct cloud regions are selected to set up users' locations. Four data centers handle the service request of these user bases. The data centers are located in such a way that the first is in region 0, the second is in region 2, the third is in region 1, and the fourth is in region 5. The number of allocated VMs is presented in data centers (DC) like 2 VMs in DC1, 5 VMs in DC2, 10 VMs in DC3, and 4 VMs in DC4. Users select optimized response time-based data center that runs an application implementing the proposed load balancing algorithm. The application uses the Optimized Response Time broker policy.

Table 4.3: Configuration of User Bases used in the experiment

Name	**Region**	**Requests /User /Hr.**	**Data Size /Req. (Bytes)**	**Peak Hours Start (GMT)**	**Peak Hours End (GMT)**	**Avg. Peak Users**	**Avg. Off Peak Users**
UB1	2	60	100	3	9	1000	100
UB2	0	60	100	3	9	1000	100
UB3	1	60	100	3	9	1000	100
UB4	3	60	100	3	9	1000	100
UB5	4	60	100	3	9	1000	100
UB6	1	60	100	3	9	1000	100
UB7	3	60	100	3	9	1000	100
UB8	5	60	100	3	9	1000	100
UB9	4	60	100	3	9	1000	100
UB10	0	60	100	3	9	1000	100
UB11	1	60	100	3	9	1000	100
UB12	4	60	100	3	9	1000	100
UB13	5	60	100	3	9	1000	100
UB14	2	60	100	3	9	1000	100
UB15	0	60	100	3	9	1000	100
UB16	3	60	100	3	9	1000	100

Table 4.4: Configuration of Application Deployment used in experiment

Data Center	# VMs	Image Size	Memory	BW
DC1	2	10000	512	1000
DC2	5	10000	512	1000
DC3	10	10000	512	1000
DC4	4	10000	512	1000

Each data center contains different number of Virtual machines. Each virtual machine has 512 Mb RAM and 10Mb bandwidth. Simulated hosts are equipped with Xen as a virtual machine monitor, sits on Linux operating system, and is based on x86 architecture. The hosts have 100GB storage, and 2 GB RAM. For others, each machine has the same number of CPUs and speed. The grouping is done in such a way that users by a factor of 10, and requests by a factor of 10. One hundred instructions corresponding to each user request is executed. CPU takes nearly 45 watts, and other units take about 28 watts to process each request. The simulation duration is set to 10 minutes. We used metrics such as response time, processing time, CPU cost, storage cost, memory cost, data transfer cost, and energy consumption to compare the proposed algorithm with other existing algorithms.

Table 4.5: Configuration of Data Center used in the experiment

Name	Region	Arch.	OS	VMM	Cost /VM ($/Hr)	Memory Cost ($/s)	Storage Cost ($/s)	Data Transfer Cost ($/GB)	Physical HW Units
DC1	0	x86	Linux	Xen	0.1	0.005	0.01	0.1	2
DC2	2	x86	Linux	Xen	0.1	0.005	0.01	0.1	5
DC3	1	x86	Linux	Xen	0.1	0.005	0.01	0.1	10
DC4	5	x86	Linux	Xen	0.1	0.005	0.01	0.1	4

Table 4.6: Configuration details at one Data Center (e.g., DC2) used in the experiment

Id	Memory (MB)	Storage (MB)	Available BW	# Processors	Processor Speed	VM Policy
0	204800	100000000	1000000	4	10000	TIME_SHARED
1	204800	100000000	1000000	4	10000	TIME_SHARED
2	204800	100000000	1000000	4	10000	TIME_SHARED
3	204800	100000000	1000000	4	10000	TIME_SHARED
4	204800	100000000	1000000	4	10000	TIME_SHARED

4.6.3 Comparative Results

Results of our experiments are displayed in Table 4.7 and Table 4.8. Corresponding graphs are shown in Figure 4.7, 4.8, 4.9, 4.10, 4.11 and 4.12.

Table 4.7: Response Time (RT) and Processing Time (PT) considering Optimized Response Time service broker policy

Algorithms		Avg. (ms)	Min (ms)	Max (ms)
RR [175]	RT	148.29	37.60	381.13
	PT	0.42	0.01	0.88
ESCE [176]	RT	148.46	37.60	520.10
	PT	0.42	0.01	0.88
TLB [177]	RT	148.47	37.60	520.10
	PT	0.43	0.01	0.86
HBF [178]	RT	148.41	37.62	367.63
	PT	0.47	0.01	3.51
ACO [179]	RT	148.30	37.60	367.61
	PT	0.43	0.01	0.95
DTPAHBF	RT	148.22	37.48	520.10
	PT	0.46	0.01	3.51

The results in Table 4.7 shows that the proposed algorithm performed better (is more efficient) in terms of overall response time compared to other existing algorithms using optimized Response Time Service Broker policy. Also, it can be seen that Round Robin is better than ESCE, TLB, HBF, and ACO algorithms. TLB, and HBF work better than ESCE and ACO. ACO is better than ESCE, and TLB is better than the HBF algorithm.

Table 4.8: CPU cost, storage cost, memory cost, data transfer cost, and energy consumption results considering Dynamic Service Broker policy

Algorithms	CPU Cost (/$)	Storage Cost (/$)	Memory Cost (/$)	Data Transfer Cost (/$)	Energy Consumption (/Watts)
RR [175]	0.50	181.80	90.90	0.18	4.15926
ESCE [176]	0.53	191.40	95.70	0.18	4.16402
TLB [177]	0.48	174.60	87.30	0.18	4.16447
HBF [178]	0.48	174.60	87.30	0.18	4.16347
ACO [179]	0.50	179.40	89.70	0.18	4.15971
DTPAHBF (Proposed)	0.48	173.40	86.70	0.18	4.15798

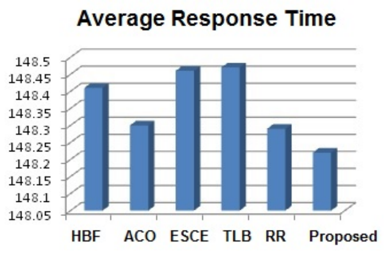


Figure 4.7: Comparison of all algorithms in terms of average response time

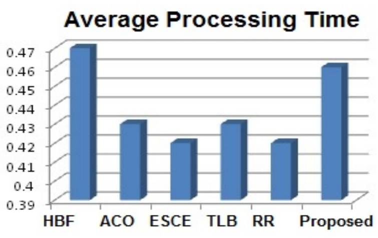


Figure 4.8: Comparison of all algorithms in terms of average processing time

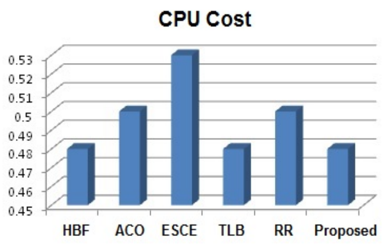


Figure 4.9: Comparison of all algorithms in terms of total CPU cost

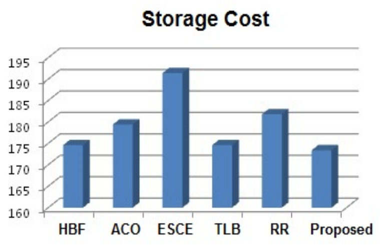


Figure 4.10: Comparison of all algorithms in terms of total storage cost

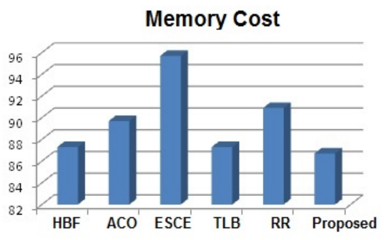


Figure 4.11: Comparison of all algorithms in terms of total memory cost

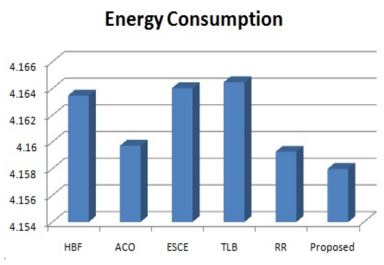


Figure 4.12: Comparison of all algorithms in terms of energy consumption per request

4.6.4 Discussion

Analyzing the generated results of the proposed algorithm in Table 4.8 it is seen that the proposed algorithm performs better in terms of CPU, Storage, and Memory cost. Thus, it is more efficient than other existing algorithms as far as Dynamic Service Broker policy is concerned. This so happens because of the distribution of equal loads among all VMs. Also, we find that the RR, TLB, HBF, and ACO algorithms are better than the ESCE because they do not have a large computation overhead. ESCE, TLB, and HBF algorithms are better than the RR and ACO algorithms. Similarly, ACO is better than Round Robin. The proposed algorithm does not fair better for processing time to other existing algorithms except the HBF algorithm as shown in Figure 4.8. In terms of energy consumption, the proposed algorithm DTPAHBF performs better than others as mentioned in Table 4.8. Figure 4.12 shows a comparative study among RR, ESCE, TLB, HBF, ACO, DTPAHBF. TLB consumes the maximum energy to process individual requests compared to others.

4.7 Conclusion

In this paper, we have presented a novel dynamic load balancing algorithm, DRSQ, which balances the workload and provides efficient application services particularly for streaming. This algorithm selects resources based on the system monitored status. So, it eliminates the possibility of selecting a resource in the dead state or bottleneck situation.

Cloud Computing has become extremely popular in spite of having several challenges, such as multi-tenancy, virtual machine migration, etc.

All existing algorithms in the literature focus mainly on overhead reduction, migration time reduction, performance enhancement, etc. We have proposed an algorithm in this section that balances the workload in the cloud environment. Our proposed algorithm DTPAHBF is inspired by the foraging principle of honey bees and a double threshold-based power-aware mechanism. CloudSim toolkit is used for experimental purposes. The results obtained through the CloudAnalyst simulator show that the proposed algorithm DTPAHBF performs better than the other widely known existing algorithms, namely, RR, ESCE, TLB, HBF, and ACO, in different aspects.

This proposed double threshold-based power-aware mechanism has notable enhancements compared to other traditional load balancing algorithms in terms of average data center processing time, overall response time, energy consumption, and total cost. Regarding the data-center processing time, DTPAHBF is less efficient than RR, ESCE, TLB, and ACO but presents better results than HBF.

Increasing cloud computing popularity, in turn increases the workload. In this section, a taxonomy is discussed to help cloud developers by emphasizing the areas where load balance is really needed and how. Moreover, a performance parameter list has been prepared which helps to assess any load balancing algorithm in the cloud framework.

To the cloud application developer, energy consumption by the resources is one of the most challenging issues along with load balancing and task scheduling in the cloud environment. Our load balacing algorithm aims to help developers in achieving that objective.

CHAPTER 5

OBJECT BASED SCHEMA ORIENTED CLOUD STORAGE SYSTEM

5.1 Introduction

In recent years, the extensive involvement of digitization in the healthcare domain has resulted in the generation of a vast amount of health data. This data is not only significant in volume but also diverse, making health data a prime example of big data. One important issue is the speed at which the data is generated, which is much faster than the data storage time on a disk [180, 181]. The organization of data has a significant impact on the architecture of data storage systems, making the need for an intelligent storage system imminent.

Due to structural differences, each dataset requires a different storage medium. In addition, all types of heterogeneous data need to be able to communicate with each other. As a result, there is a growing need for a storage system that not only supports the storage of multiple data types in a combined manner but also supports a uniform query structure for these different types of datasets.

5.2 Background

There are various storage systems with distinct features available in the data storage field. For example, some focus on data storage structure, data security, data volume, or durability issues [182]. Traditional databases, for instance, prioritize maintaining the relationship among datasets through their data storage structure [183]. On the other hand, NoSQL databases place a greater emphasis on reliability, durability, and scalability over maintaining relationships among datasets [184, 60]. An object-oriented cloud storage system treats data as objects and manages them through web services such as Amazon S3 [185, 186] and Swift [109]. The selection of a storage system depends on the data unit structure used by data service applications [127, 187].

5.3 Research Challenges and Solutions

Developing and managing the storage aspect of an object-based data storage system is highly challenging due to the diversity of data. These days, storing data involves more than just placing it in any data storage device. It also requires storing data in a manner that meets the demands of applications, such as fast response times, resource capabilities, and application context [188, 185, 125, 189]. The main question here is how to design the data storage space to handle data diversity from the perspective of applications. Existing storage systems either focus on handling structured data (stored in a table format) [190, 191], semi-structured data (such as documents and graph-based structures) [98, 192], or unstructured data (such as files, images, audio, and video) [193, 80]. Object-based storage techniques [194] allow for the storage of multiple heterogeneous data types within a container by applying an abstraction layer and assigning a unique key value. Several research questions remain, such as:

- How will the storage system behave when structured and semi-structured data are stored with unstructured data?
- How should the data storage space be designed when multiple types of data are stored together?
- How will the data storage system behave when the storage system satisfies application-defined components?

The following techniques are examined to address the mentioned challenges:

- An architectural view of the storage object explains what components are involved in storing data in a storage system and how they are involved.
- An algorithm that outlines the step-by-step process for storing, retrieving, and deleting a dataset.
- A data model that is used to map data objects to storage devices.
- A query structure for read, write, and delete queries to handle heterogeneous datasets through a single interface.
- A framework that outlines the architecture for a storage system that supports the aforementioned mapping and query-related tasks.
- Justification for this storage system."

5.4 Object Storage Space

5.4.1 Global Database Schema

Algorithm 2 describes the steps for creating a global schema for the data storage system. It is executed only once, meaning that the system initially runs this algorithm and does not need to change it in response to dynamic user requests. This algorithm will be called after any physical changes are made to the data storage system, such as adding or deleting a data storage unit or updating the number or content of the local schema of existing data storage units.

The global database schema describes the logical view of all the available data storage units in the data storage system. It is represented as a tuple $S_{globalDB}$={data attribute (DA), local schema (LS), data storage unit (DS)}. In the proposed data storage system, the global view of the storage system is represented as:
f1: DA $\rightarrow$ HV
f2: LS $\rightarrow$ HE
f3: DS $\rightarrow$ HHE
For the case of example 1, Global database Schema is: $S_{globalDB}$={{DeviceID, Data, time, PatientID}, {L1,L2}, {Cassandra, MongoDB}}. It helps perform the searching job, mentioned in part 2 of algorithm 1.

Algorithm 2 A Generalized Algorithm For Designing The Global Database Schema

```
Input: Database set, local schema set, data attribute set,
Output: Global view of data storage devices
  Step 1: Generate the structure of each hyperVartex corresponding to each data attribute.

  Step 2: Generate a single hyperedge corresponding to each data storage schema
  Step 3: Generate a single hyperhyperedge corresponding to each data storage unit
  Step 4: Combine all the generated hyperhyperedges for making the global view of the
  data storage devices.
```

5.4.2 Temporary Database

The temporary schema represents the parameters needed to fulfill the users' demands. Conceptually a temporary schema structure is represented as a tuple Stemporary: {userID, Attribute Name, condition}. For the case of example 1, if a user (user id=u001) requests only the real-time data set for a particular device, the tuple of the temporary schema will be: Stemporary={u001, {data, time},{begin time=11.9.15 00:09:45, end time=11.9.15 00:10:00}}. Here, the condition part is optional and has no role in designing the object storage space. This user can provide other important information.

5.4.3 Hierarchical Structure of The Storage Object

The hierarchical structure of the storage object is shown in Figure 5.1. It is not sufficient to store data directly under the container as an object for storing structured datasets. It is worth noting that under a single request, a user in the healthcare domain may demand more than one type of data [195, 196]. Therefore, our focus is on combining multiple types of data without violating the storage structure of each type of data. For these reasons, we store data under the object through multiple layers. The object points to more than one type of database because each database can store a single type of data; for example, Cassandra [128] cannot store documents. The key value of the object will be the combined value of the user ID of the requesting user and the request time. Data is stored under a selected local schema. This concept allows the user to make their request using the names of the desired data attributes.

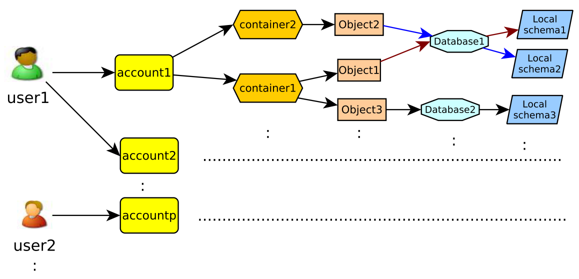


Figure 5.1: Multilevel view of the data container for storing different types of datasets

The object storage space's schema determines how it stores the collected dataset in the storage system. Conceptually schema of the object based storage space looks like:
Sobject={$\underline{\text{userID\_requestTime}}$,{{Local Schema....}, Database} ..... }

5.5 Hypergraph Data Model

A hypervertex (HV) is used to represent an existing data attribute node that is already present in the data storage platform. It acts as the vertex node of the hypergraph and has six properties: userID, dataType (e.g., structured, unstructured), context (e.g., text, document, id), attributeName, schemaName, and databaseName to describe the structure of data attributes. In example 1, the DeviceID, Data, time, PatientID, ID, and Document attributes of the set of local schemas act as the set of hypervertices.

A HyperEdge (HE) determines the location of an attribute within the local schema. Data is stored through a static or dynamic schema for structured or unstructured databases. Each hyperedge connects all hypervertices belonging to the same local schema. Conceptually it is represented as: HG[E] = $\bigcup_{i=1}^{n} HE_i$, Here n$\leq$ the number of local schema of all databases and each hyperedge (HE) represents a local schema. HE_i= $\bigcup_{i=1}^{m} HV_i$, Here m$\leq$number of attributes, $HG[E]$=edge set of hyper graphs, HE=hyperedge and HV=hypervertex. At the same time, we update the property list of the hypervertices by adding schemaName property having the value of local schema name. Considering example 1, we may have a hyperedge HE1 as mentioned in the following. HE1 ={ HV1, HV2} and the value of HV1="Data", HV2="time". According to example 1, HE1 represents the local schema L1. Therefore, we add property store=L1 in the property list of HV1 and HV2.

HyperhyperEdge (HHE) provides the information of all local schemas in the data storage units. Each data storage unit is represented as a single hyperhyperedge. It is represented as mentioned in the following. HHG[E]= $\bigcup_{i=1}^{m} HHE_i$ where, m $\leq$ number of HE in HG[E], HHE_i= $\bigcup_{j=1,k=1}^{n,p} \{HE_j/HV_k \in HE_j in HHG\}$ where HHG[E]=hyperhyperedge list of hypervartex. As before, we update the structure of hypervertices by adding databaseName property having the value of database name. By considering example 1, HHG[E]={HHE1, HHE2}, where HHE1 represents Cassandra and HHE2 represents MongoDB. HHE1={HE1, HE2}, where HE1 corresponds to L1 and HE2 corresponds to L2. Also, HE1 ={ HV1, HV2} and HE2 ={ HV5, HV6}. Then, we add property databaseName=Cassandra to HV1, HV2, HV5 and HV6.

Working procedure of hypergraph data model: The configuration procedure follows three steps to design the storage object: (1) Configuration of the hypervertex set: This step aims to establish the relationship between the users' requested data parameters and the attribute set of the global database schema. It does this by updating the hypervertex property list; the working procedure is shown in Figure 5.2. (2) Segregation of the hypervertex set: The attribute set of the global database schema is divided into a certain number of sets of hypervertices. Each set represents the requested parameters of a user. The hypergraph coloring algorithm 3 is used to accomplish this task. (3) Determining the location of the requested parameters: This step is dedicated to finding the location of the requested data parameters, i.e., the local schema of the database. The tasks mentioned in this step are used to generate the final object storage space corresponding to each user's request, as shown in Figure 5.2.

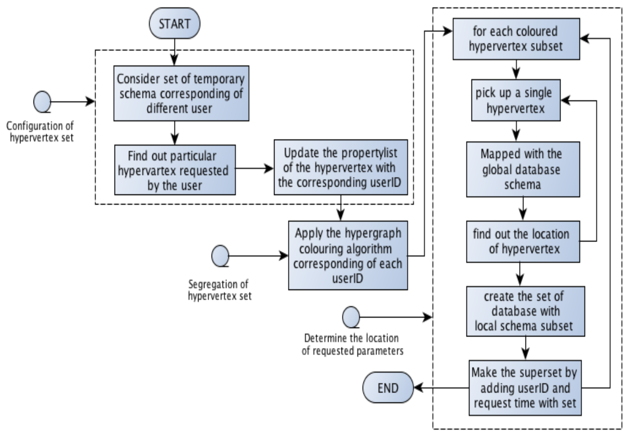

START
Configuration of hypervertex set
Consider set of temporary schema corresponding of different user
Find out particular hypervartex requested by the user
Update the propertylist of the hypervertex with the corresponding userID
Segregation of hypervertex set
Apply the hypergraph colouring algorithm corresponding of each userID
for each coloured hypervertex subset
pick up a single hypervertex
Mapped with the global database schema
find out the location of hypervertex
Determine the location of requested parameters
create the set of database with local schema subset
Make the superset by adding userID and request time with set
END

Figure 5.2: Configuration of requested object storage space

Algorithm 3 Segregation Of Hypervertex Set According To The Each User's Request

Input: userID set(UID[]), structure of hypervartex(HS.UID, HS.DID, HS.TYPE, HS.CONTEXT, HS.SCHEMANAME, HS.DATABASENAME), hypervartex set(HV[]), number of hypervertex (HN), Number of user (UN), request List of user

Output: coloured hypervertex set

```
1: begin
2: set K to 1
3: for all Row I in HV[] and J in UID[] do
4:     Compare between HS.UID of HV[I] and UID[J]
5:     Collect the equal valued vertex respect of single UID[J]
6:     Set the colour of HV[I] is C[K]
7: end for
8: Set value of C[K] with C[K+1]
9: end
```

5.6 Architecture

The configuration procedure follows three steps to design the storage object:

1. Configuration of the hypervertex set: This step aims to establish the relationship between the users' requested data parameters and the attribute set of the global database

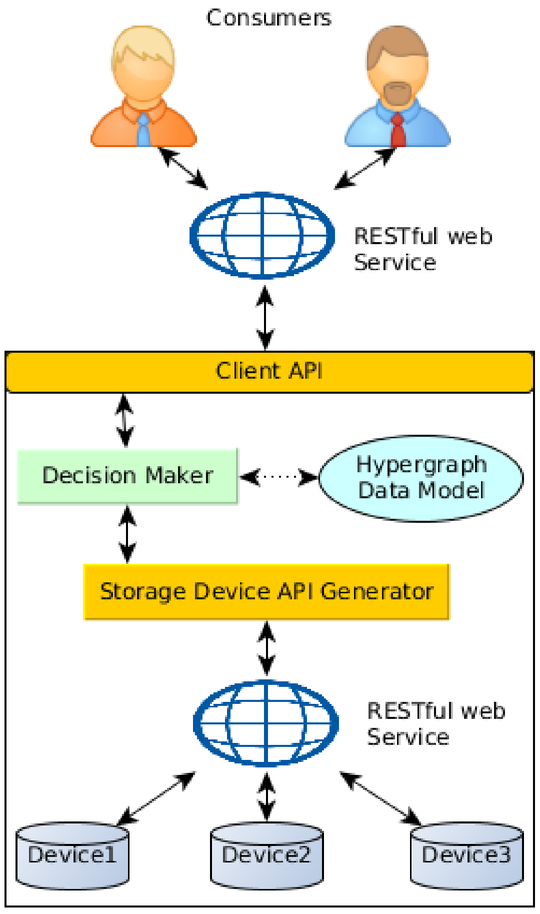


Figure 5.3: Basic architecture of object based schema oriented data storage system

schema. It does this by updating the hypervertex property list; the working procedure is shown in Figure 5.2.

2. Segregation of the hypervertex set: The attribute set of the global database schema is divided into a certain number of sets of hypervertices. Each set represents the requested parameters of a user. The hypergraph coloring algorithm 3 is used to accomplish this task.

3. Determining the location of the requested parameters: This step is dedicated to finding the location of the requested data parameters, i.e., the local schema of the database. The tasks mentioned in this step are used to generate the final object storage space corresponding to each user's request, as shown in Figure 5.2.

The RSoS system communicates with different storage devices for handling different types of datasets. Each storage device is dedicated to storing a particular type of dataset. For example, Device 1 is responsible for managing table-type datasets, Device 2 for document-type datasets, and Device 3 for file-type datasets. Each storage device is run by the POST method of the RESTful web service under a unique URI. To communicate with the storage device unit, JSON-formatted data is sent to instruct about the contents of the data object, the account ID, and the container name. The Storage Device API Generator is responsible for generating this input JSON format data for a particular storage device unit. It is also responsible for calling the storage device unit instructed by the decision-maker, using the assigned URI. At this time, URI selection is also dependent on the database operation because each storage device unit has three different URIs for three basic database operations.

5.7 Query Elements

The HTTP method of the REST API is used to support database operations. The storage system creates a container during a database write operation and deletes a container during a delete operation. Supported database operations are READ, WRITE, and DELETE. The POST verb is used to draw the data service infrastructure. Table 5.1 shows the details of the operations used to perform database operations. Each database operation has a unique PATH value under the POST verb, and the JSON API is used to query the database.

In this protocol, two components are used, viz. URL and input object.

(a) **URL:** It is the path that is unique for a specific database operation. The RESTful API service is used to communicate with the RSoS system. A single machine is set up as a database server for the RSoS system. To communicate with this database server, the machine's IP and port number and server identification address are needed. The URI of this database server is 'http://<machine IP>:<machine port>/SchemaBasedObjectDB/rest'. The unique path value, added to the URI, is used to specify the URL of a specific database operation shown in this table.

(b) **Input Object:** Here, the JSON API is used to describe the resources with the stored datasets, known as the Input Object. This information is used to query the storage system. For each database operation, the schema structure (i.e., a combination of tag-value pairs) of the JSON API varies. Some fixed tags of the JSON API are used for all database operations, for example, 'account' for specifying the user's identification value, 'container' for the user-provided unique identification value, and 'operation' for specifying the type of operation. The 'WRITE' database operation is represented by 'Store', 'READ' is represented by 'Retrieve', and 'DELETE' by 'Delete'. Some variable tags are used for specific database operations. 'Datagroups' for the 'WRITE' operation is mentioned in the input data object. 'Condition' for the 'READ' operation consists of a subpart of the data object to filter the data object that the user wants to retrieve. For the 'DELETE' operation, the 'eliminate' tag is used to specify the data object that the user wants to delete by referring to the attribute-values of the data object.

5.8 Machine Configuration

The machine configuration includes the hardware and software information about the machines used to perform the experiment. For the RSoS system, four machines are used, where one machine acts as a server node and three other machines act as storage device units. At

Table 5.1: Query Elements for performing database operations

Operations		
Operation Name	Operation Type	Operation Value
WRITE	URL	http://<machine IP>:<machine port>/SchemaBasedObjectDB/rest/store
	input object	{ "account":"U101", "operation":"Store", "container":"U04_ecg_12345", "datagroups":{ "deviceid":"k009", "sensortype":"ecg", "patientname":"Sushanto", "patientid":"P001" } }
READ	URL	http://<machine IP>:<machine port>/SchemaBasedObjectDB/rest/retrieve
	input object	{ "account":"U101", "operation":"Retrieve", "container":"U04_ecg_12345", "condition":{ "deviceid":"k009" } }
DELETE	URL	http://<machine IP>:<machine port>/SchemaBasedObjectDB/rest/delete
	input object	{ "account":"U101", "container":"U04_ecg_12345", "operation":"Delete", "eliminate":{ "deviceid":"k009" } }

the backend, the server node runs on the tomcat server. Four RESTful web services are running on top of the tomcat server. Three of them are dedicated to controlling each storage device unit individually. One of them is used to control the storage device unit via the three dedicated web services. All of these setups are located in Mumbai, India using the Amazon EC2 setup. The client node of the RSoS system is located in our laboratory at the Salt Lake campus of Jadavpur University in Kolkata, India.

The main purpose of the RSoS system is to handle different types of datasets using a single setup. To achieve this, it utilizes the functionality of existing database units such as MongoDB [129], Cassandra [128], and HDFS [193] as storage device units. HDFS is capable of handling file data, MongoDB for documents, and Cassandra for table format. Each device unit is accessed by a unique URI provided by the dedicated web service. The RSoS system communicates with them using these URIs via the REST API. The detailed description of the machine setup is presented in Table 5.2.

In our lab, client APIs have been used to access the storage features of Amazon S3 and Microsoft Azure. To reduce the network latency, the nearest data center location has been chosen as the region, such as Mumbai for Amazon S3 and South India for Microsoft Azure. Table 5.2 provides detailed information about these client API setups and storage space descriptions.

5.9 Testing Platform

For the experiment, a set of real healthcare data is considered from the public repository available on the Internet. Three types of datasets of cancer patients are used as test datasets:

1. Patients' treatment-related information [197] with descriptions about the disease details; the associated data attributes are mentioned in Table 5.4.

2. Treatment expenditure-related dataset [198] which is organized as a .JSON format for carrying out the experiment.

3. Medical images of breast cancer patients collected from [199]. In the test dataset, it has two attributes: PatientID and Medical Image. The 'Medical Image' attribute refers to the image file as a .png format, and 'PatientID' acts as an image identifier by renaming the image file name using the corresponding PatientID value.

A detailed description of the attributes of these three test datasets is presented in Table 5.4. 'PatientID' acts as the ID for all these types of test datasets. The three types of datasets are designed to highlight the big data variety property in the health domain and how the RSoS system, Amazon S3, and Microsoft Azure perform in this scenario.

For the RSoS system, our main focus is to design the storage space using a schema-oriented object-based technique. Here, any type of data is considered a data object and is stored in this data storage system by creating schema-database pairs, known as object storage spaces. The detailed description of the object storage space with the storage architecture is already described in Section 5.6. Listing 5.1 represents the structure of the global database schema of the test dataset (Table 5.4).

For Amazon S3 [8] and Microsoft Azure [17], all types of data are stored in the form of data files. For experimental purposes, to store the non-file structured dataset, an empty file is first created by assigning the name, same as the name of the object key for Amazon S3 and blob key for Microsoft Azure (Table 5.3), and then passing data as a data input stream. For file-structured data, the data file contents are copied to the newly created object file. An individual bucket in Amazon S3 or container in Microsoft Azure is created for each type of dataset (Table 5.3).

```
Global Database Schema,
 $S_{globalDB}$={{PatientId, Outcome, Time, Tumor size
, Image, expenditure},{Local Schema of
Cassandra(L1), Local Schema of HDFS(L2), Local
Schema of MongoDB(L3)}, {Cassandra, HDFS,
MongoDB}}
```

Listing 5.1: Global Database Schema and Object Storage Schema of RSoS System for experimental dataset

Table 5.2: Information about Resource setup

Resource Role	**Purpose**	**Configuration**
Server Node	To run RSoS setup.	4 t2.micro instance of Amazon EC2 with 1 VCPU and 1 GB memory. Ubuntu Server 14.04 LTS-64 bit run as backend operating system. one instance is used to run the web services and other three instances are acts as storage device units.
Client node	Used for sending http request to execute the query.	32 GB RAM, 1.2 TB hard disk, 4 core processor, Ubuntu 14.04 LTS operation system
Tomcat Server	Run as backend server for RSoS package.	Number of active threads 850, length of completed queue 300.
MongoDB	Used for handling document type dataset.	RSoS system use REST API to communicate with/accessing all the features of mongoDB which acts as storage device unit of RSoS system.
HDFS	Used for handling file type dataset.	HDFS handled by REST API and it acts as file storage device unit of RSoS System.
Cassandra	Used for handling table format dataset.	For supporting time series sensor generated data. Cassandra acts as storage device unit of proposed RSoS unit. RSoS communicates with cassandra by REST API.
Amazon S3	For performance comparison	Amazon aws java package is used for designing client API, Bucket region is placed in Asia Pacific (i.e., Mumbai).
Microsoft Azure	For performance comparison	Blob region is placed in south India. LRS (Local Redundancy Storage: multiple synchronous data copy in a single data center) redundancy technique is used. Azure-storage java package is used.

Table 5.3: Detailed description of the database architecture of various case study

Test Dataset	**Data Format**	**Data Type**	S_{object} **for RsoS System**	**Amazon S3**	**Microsoft Azure**
Patient's disease information	Table	Text	{PatientId,{{Outcome, Time, Tumor size}, Cassandra}}	Bucket Name: askmtable Object Key: PatientId.txt	Container Name:askmtable Blob Key: PatientId.txt
Medical Image	File	Image	{PatientId,{{Image}, HDFS}}	Bucket Name: askmimage Object Key: PatientId.png	Container Name:askmimage Blob Key: PatientId.png
Treatment expenditure	Document	JSON	{PatientId,{{Document}, MongoDB}}	Bucket Name: askmdocument Object Key: PatientId.json	Container Name:askmdocument Blob Key: PatientId.json

5.9.1 Query Descriptions

For the performance evaluation of the RSoS system, four database queries (Table 5.4) are considered. These queries represent four different complex jobs in four directions:

1. Query I represents the data append operation of three different types of datasets. The performance of the RSoS system for this query is measured on the types of datasets.
2. Query II extracts the datasets that the user needs. This job involves a single type of data or any combination of data.
3. Query III represents an aggregate function, which is a mathematical operation that is part of a database operation.
4. Query IV deletes a dataset from the database. In order to delete all related information, it is necessary to survey all types of data.

Table 5.4 presents the detailed description of all queries with all the data attributes. These jobs are used to demonstrate how the database architecture affects database performance.

Table 5.4: The description about queries

Number	Query Description	Data Attribute
Query I (Append operation)	(i) Insert medical image of effected region of breast cancer patient.	(i) PatientId and Medical Image in .png format.
	(ii) Insert disease details of a breast cancer patient.	(ii) PatientId presented in 9 numeric text value, Outcome of treatment procedure [Recover(R) or non recover (N)], Treatment duration in years, Tumor size in cm.
	(iii) Insert expenditure details of cancer patients.	(iii) PatientID, Cancer Site (types of cancer), Year of treatment duration, Sex (Male or female), Age in years, Incidence and Survival Assumptions, Annual Cost Increase (applied to initial and last phases), Total Costs, Initial Year After Diagnosis Cost, Continuing Phase Cost, Last Year of Life Cost.
Query II (read operation)	Retrieve all the information (patient details, treatment expenditure, medical image) or only patient details or treatment expenditure or medical image of the breast cancer patient who recover properly.	PatientId, Outcome, Treatment duration in years, Tumor size, sex, age, Annual Cost Increase (applied to initial and last phases), Total Costs, Initial Year After Diagnosis Cost, Continuing Phase Cost, Last Year of Life Cost, Medical Image.
Query III (aggregate function)	Make a summarization of total cost of all melanoma cancer patients	Cancer Site, Total Costs
Query IV (delete operation)	Deletion of all the information of breast cancer patient where total cost is less than 10,000.	PatientId, Treatment duration in years, Tumor size, sex, age, Annual Cost Increase (applied to initial and last phases), Total Costs, Initial Year After Diagnosis Cost, Continuing Phase Cost, Last Year of Life Cost, Medical Image, Outcome.

5.9.2 Query Time

The main goal of the RSoS system is to efficiently perform database operations such as append, read, delete, and aggregate. The experiment is conducted on various data sizes: 10, 100, 1000, and 10,000 on three types of datasets: table, document, and file. Each data size represents a single record or object. For example, if the data size for table-type data is 10, it represents 10 data records or objects that contain the same attribute sets. Similarly, for the document-type dataset, if the data size is 10, it contains 10 different documents with different or the same key-value pairs, and for file-type data, it consists of 10 different files.

Here, a table represents a structured dataset, the document represents a semi-structured dataset, and the file represents an unstructured dataset. The generated resultset after running these queries is shown in Table 5.5.

The RSoS system executes query II by displaying the data contents of the table and document type datasets in the console and downloading the image data (file format). Amazon S3 and Microsoft Azure download all types of data as a file.

For query I, the execution time is calculated based on the data type. Query II extracts information from a particular type of dataset or a combination of all types of datasets. Table 5.5 represents the results for table, document, file, and all (a combination of table, document, and file type datasets) respectively. The run time of query III and query IV is not segregated based on the data type. They process the corresponding dataset of any type and the generated results are shown in Table 5.5.

Table 5.5: The experimental resultset of four different queries

(a)Performance of Query I in HH:MM:SS

Number of records	Patient's disease information (table format)			Medical Image (file format)			Treatment Expenditure (document format)		
	RSoS	S3	Azure	RSoS	S3	Azure	RSoS	S3	Azure
10	5.01	24.73	5.109	6.89	7.25	8.53	1.22	4.68	2.71
100	6.81	19.97	6.81	45.71	33.13	26.01	1.64	18.57	10.62
1000	5.35	2:41.68	1:02.39	6:29.64	3:52.71	10:02.02	3.03	2:52.58	1:10.79
10,000	19.03	32:26.99	10:41.25	9:18.85	46:20.32	1:25:41.31	30.86	27:38.28	15:33.75

(b)Performance of Query II in HH:MM:SS

Number of records	RSoS				S3				Azure			
	All	Table	File	Document	All	Table	File	Document	All	Table	File	Document
10	10.25	3.68	17.64	4.70	16.76	12.61	18.5	12.69	12.73	6.70	6.81	8.71
100	21.55	3.53	19.33	3.81	01:27.21	43.24	01:20.32	45.84	1:57.06	33.63	1:16.52	39.89
1000	2:20.39	11.25	1:42.66	5.02	28:04.52	11:38.96	12:37.30	8:50.93	1:22:34.49	17:59.31	24:11.96	20:10.54
3000	4:58.05	7.21	4:14.63	11.02	4:27:11.84	59:27.36	1:16:00.31	1:15:03.62	7:19:03.07	2:36:53.50	2:40:53.21	2:26:02.07
10,000	21:07.66	37.88	16:50.21	5:39.81	>20:00:00	9:24:08.13	10:33:38.9	10:49:38.5	>20:00:00	>20:00:00	>20:00:00	>20:00:00

(c)Performance of Query III in HH:MM:SS

Number of records	RSoS	S3	Azure
10	0.90	6.59	6.96
100	0.91	27.25	16.58
1000	1.0	02:15.26	2:35.66
10,000	2.30	18:42.37	19:01.77

(d)Performance of Query IV in HH:MM:SS

Number of records	RSoS	S3	Azure
10	7.64	22.24	11.40
100	7.47	24.92	10.40
1000	28.44	02:54.87	1:42.14
10,000	02:13.68	50:10.51	15:42.95

5.10 Comparative Resultset

Table 5.6: Detailed description of the data types of various case study

Test Dataset	**Data Format**	**Data Type**
Patient's disease information	Table	Text
Medical Image	File	Image
Treatment expenditure	Document	JSON

To evaluate the performance of the RSoS system, four queries were considered which reflect database operations in four directions as presented in Table 5.4. A test dataset was prepared which includes three types of data: structured (i.e., table), semi-structured (i.e., document), and unstructured (i.e., file) as described in Table 5.6. By comparing all the figures (Figure 5.4, Figure 5.5, Figure 5.6, Figure 5.7, Figure 5.8, and Figure 5.9) in Figure 5.10, it was found that the RSoS system performs better than Amazon S3 and Microsoft Azure except at Figure 5.6 up to 1000 records.

5.11 Conclusion

Storage architecture plays an important role in a data storage system to provide good data services. A novel storage architecture has been proposed here, named "Object-based schema-oriented data storage System" (RSoS System). The RSoS System follows the account-container-database-schema storage architecture. The main idea behind this architecture is that data is distributed in two layers of database and schema within an object. The motivation for this storage architecture is to handle the big data variation property. Health data has been used for experimental purposes.

The RSoS System is capable of storing three types of data: documents, tuples, and images using a single interface. To demonstrate the usefulness of the RSoS System in the database world, a comparison has been made with the two most familiar storage architectures: bucket-oriented (e.g., Amazon S3) and account-container-oriented (e.g., Microsoft Azure). A table shows the comparative output of the execution time of running four different queries on three data types (table, file, and document) with varying numbers of records (e.g., 10, 100, 1000, 10,000). In this comparison, the RSoS System provides much better performance compared to Amazon S3 and Microsoft Azure. Additionally, a comparative result-set has been presented to show the flexible nature of the RSoS System compared to Amazon S3 based on data type variation.

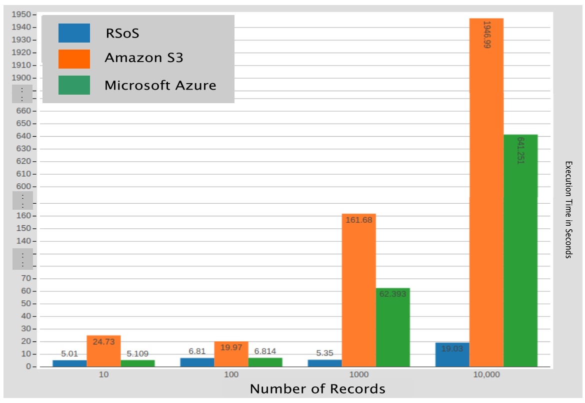


Figure 5.4: Query1 execution time on table type dataset

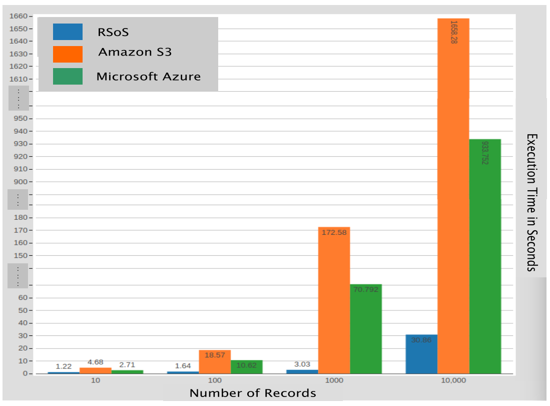


Figure 5.5: Query1 execution time on document type dataset

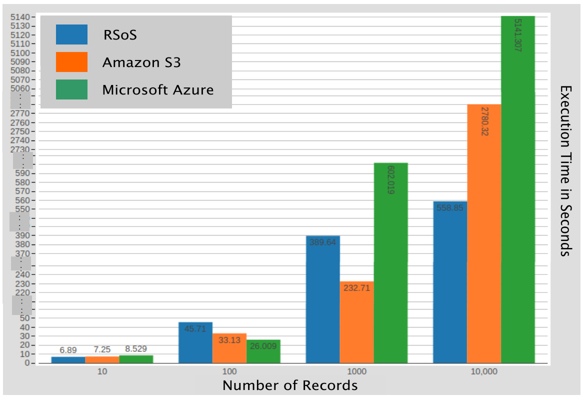


Figure 5.6: Query1 execution time on file type dataset

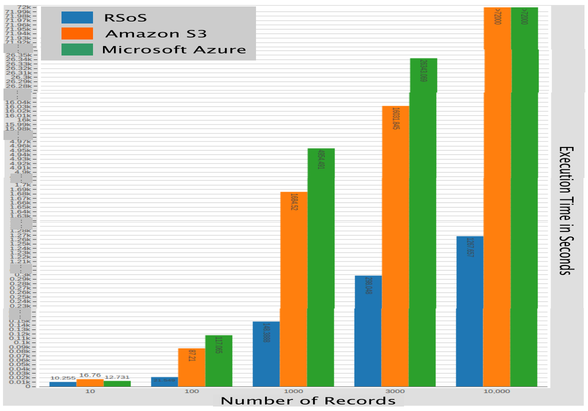


Figure 5.7: Query2 execution time on all type dataset

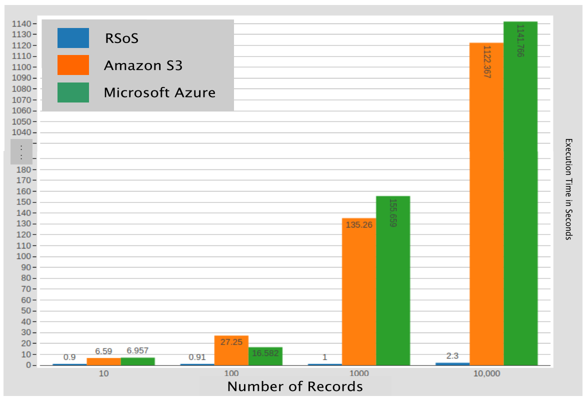


Figure 5.8: Query3 execution time

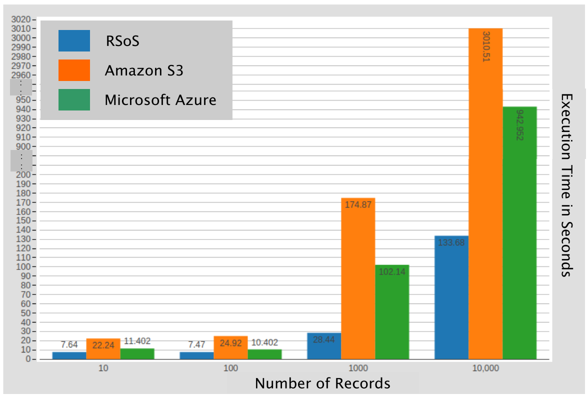


Figure 5.9: Query4 execution time

Figure 5.10: Comparison of query execution times in between Amazon S3, Azure and RSoS

Chapter 6

Automatization of Object based schema oriented cloud storage system

6.1 Introduction

Nowadays, the application areas of a cloud storage system are not limited to just digital data deposits. Many machine learning approaches [114, 200, 201] are being implemented in renowned cloud storage systems to make them more computationally advanced. One important application of cloud storage systems is storage location prediction.

Health data comes from various sources, such as sensor providers, hospital managers, account managers, and patient's relatives, and the structure of the generated data varies depending on the source. Over time, more information may be added to the existing data, and the structure may become unstable as new data is added [202]. This makes the task of predicting storage space more complex.

There are various cloud storage systems available in the literature to support health data, including Amazon S3 [82] which provides a bucket-oriented object storage architecture, Openstack Swift [109] which supports an account-container based object storage architecture, and the Object-based schema-oriented data storage System [48] which follows a schema-based account-container oriented object storage architecture. Some time-series databases are also used to handle health sensor datasets, such as PhilDB [203] which uses metadata tracking architecture to identify every time-series data type, and SciDB [204] which follows a native array data model for handling time-series datasets. The challenge is that health data is not just composed of time-series data or large chunks of data files; it is a combination of different types of datasets (time-series data, graph-based data, files, etc.). Only a schema-based architecture can provide the necessary flexibility for storing these different types of health data. In this way, the characteristics of this schema-based storage architecture not only support the variety of big data, but also reduce the execution time of database operations, as shown in [48]. For this reason, we have considered the RSoS system as a prototype design in this chapter.

The RSoS system may use different database models for storing various types of data, so it is necessary to allocate separate storage space for different data types. However, manually identifying the data types takes time and requires constant human intervention. Therefore, it is important to automate the process of identifying the data type before sending it to its designated storage space. Machine learning approaches can assist with this by automatically predicting the storage space corresponding to each type of incoming data.

The rest of this chapter is structured as follows. The purpose of this chapter is described in the Motivation and approach chapter (Section 6.2). Section 6.3 describes the existing classification engines. Section 6.4 outlines the proposed classification engine framework with the workflow model. The main focus of Section 6.4.4 is to identify the components of the proposed classification engine, namely a feature selection algorithm and a classifier, through an experimental comparative study. Finally, Section 6.5 concludes the chapter with a discussion on future work.

6.2 Motivation and Approach

To communicate with the RSoS system (Object-Based Schema-Oriented Data Storage System), users use three queries in the JSON format: READ, WRITE, and DELETE. Before storing any data in the RSoS, a user must provide related information such as the account, attribute name, attribute status (key or nonkey), and storage location (database name). This information is maintained in a graphml.xml file using a hypergraph data model, as shown in Listing 6.1. According to this listing, the account values are U101, U102, U103, and U104, and the attribute names are Time, Temperature, PatientID, PatientDetails, fileextension, remotefiledata, and id. The storage devices are named based on the database model used in that storage device. In this case, three database models are used: Cassandra, MongoDB, and HadoopDFS. These storage locations can be accessed using unique and individual URIs, as shown in Listing 6.1.

The hypergraph data model, known as the graphml.xml update, must be updated manually with every little change to the data schema or with every new dataset entry. This is a laborious and time-consuming task for humans. Therefore, in this chapter, we propose the following to address these challenges.

- A classification engine framework to predict the probable object storage space architecture from the characteristics of the input data.
- The detailed architecture of the RSoS system associated with the proposed classification engine framework.
- A workflow model of the classification engine framework, along with a detailed description of the involved sub-units and their internal communication.
- A comparative analysis to determine the best combination of a feature selection algorithm and classifier for the designed classification engine framework.

```
<?xml version="1.0" encoding="UTF-8"?>
<graphml xmlns="http://graphml.graphdrawing.org/xmlns/graphml"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://graphml.graphdrawing.org/xmlns/graphml">
<graph edgedefault="directed"  >
<node id="1" userID="U101" name="Time" status="key"/>
<node id="2" userID="U101" name="Temperature" status="nonkey"/>
<node id="3" userID=":U101:U104" name="PatientID" status="nonkey"/>
<node id="4" userID="U103" name="PatientID" status="key"/>
<node id="5" userID="U103" name="PatientDetails" status="nonkey"/>
<node id="6" userID="U102" name="fileextension" status="nonkey"/>
<node id="7" userID="U102" name="remotefiledata" status="nonkey"/>
<node id="8" userID="U102" name="id" status="key"/>
<edge source="1" target="2" HyperEdgeName="L1" HyperHyperEdgeName=
"Cassandra" URI="http://localhost:8080/cassandra/rest" />
<edge source="2" target="3" HyperEdgeName="L2" HyperHyperEdgeName=
"Cassandra" URI="http://localhost:8080/cassandra/rest" />
```

```
<edge source="3" target="1" HyperEdgeName="L3" HyperHyperEdgeName=
"Cassandra" URI="http://localhost:8080/cassandra/rest" />
<edge source="5" target="4" HyperEdgeName="L4" HyperHyperEdgeName=
"MongoDB" URI="http://localhost:8080/MongoRestServer/webresources
/rest"/>
<edge source="7" target="8" HyperEdgeName="L5" HyperHyperEdgeName=
"HadoopDFS" URI="http://localhost:8080/HadoopRestServer/webresources
/rest"/>
<edge source="6" target="7" HyperEdgeName="L6" HyperHyperEdgeName=
"HadoopDFS" URI="http://localhost:8080/HadoopRestServer/webresources
/rest"/>
<edge source="6" target="8" HyperEdgeName="L7" HyperHyperEdgeName=
"HadoopDFS" URI="http://localhost:8080/HadoopRestServer/webresources
/rest"/>
</graph>
</graphml>
```

Listing 6.1: Example of data contents presented in hypergraph data model

The functionality of the classification engine depends on the associated applicant [205, 206, 207, 208]. However, the purpose of the classification engine in this case is different from that of existing ones, motivating the creation of a new classification engine framework for the RSoS system.

The RSoS system is a cloud storage system where data is received in its own query format. Due to the unique structure of each query for different dataset types, the datasets generated from each query act as the input to the designed classification engine. This generated dataset holds the status values of the system sub-parts of the server corresponding to the query. The input data looks like a two-dimensional matrix, $M = [m_{ij}] \in \mathbb{R}^{m \times n}$, where m and n represent the number of queries and the number of system sub-parts treated as features, respectively. The training dataset includes an additional column called 'class' which holds the data type value of the corresponding query with the input data. The test dataset includes only the feature values of the input data.

To determine the most suitable and efficient machine learning technologies for this framework, a comparative study of feature selection approaches combined with classifiers is conducted. The feature selection algorithms applied to the training dataset are responsible for selecting the relevant features. The trained classifier then determines the class value of the test dataset based on the selected features. For comparison purposes, three feature selection algorithms (Lll21, Fisher score, F-Score) and three classifiers (K-NN, Neural Network, SVM with three kernel functions: linear, polynomial, Radial Basis Function (RBF)) are considered.

The ultimate goal of this proposed classification engine framework is to devise a feature selection algorithm along with a classifier to determine the proper object storage space for the incoming data presented in the WRITE query. The experiment in this chapter is limited to the prediction of the storage device URI as part of the object storage space, with the other parts (e.g., attribute list) being constrained to the supporting storage device.

6.3 Background

In this section, we discuss some well-known classification engines and describe how they perform tasks based on the demands of the associated application. The development of a classification engine as a machine learning technology for cloud storage systems can be broadly categorized into two research areas: the services provided by existing classification engines, and the machine learning approaches associated with different cloud storage systems.

One example of a classification engine is the Varonis engine [208], which aims to classify, manage, and protect sensitive datasets from cyberattacks. To accomplish this, it monitors user behavior and determines who should have access to which types of data. Additionally, it helps prevent unauthorized access to data.

The Autonomous Classification Engine (ACE) provides a framework that allows users to optimize the classifier, classifier parameters, and reconfigure the problem by providing feature vectors [205]. Using this framework, the authors in [209] classify beatboxing sounds and find that the adaBoost classifier with C4.5 decision trees provides more accurate results than other methods. A genetic algorithm is used for feature vector generation in this study.

The Advanced Classification Engine (ACE) is maintained and designed by the Websense security lab (Security Overview Websense ACE (Advanced Classification Engine)). It is designed to provide real-time Websense gateway and cloud security services by classifying web page contents, URL, and protocols inline based on the presence or absence of hidden malicious messages in page data contents [206, 210]. PSIGEN also releases an ACE (Accelerated Classification Engine) [211], which allows for custom classification of documents based on user requirements.

The Autonomous Classification Engine (ACE) is a framework consisting of three classification engines: the Data Classification Engine, the Storage Classification Engine, and the Data Placement Engine for Information Lifecycle Management (ILM) [212]. In this framework, the storage location of a data item is based on its business value. The authors in [213] proposed a classification engine that is used to identify the name of the social network platform of an image. This engine is built using K-nearest neighbor (K-NN) classification and a decision tree mechanism.

Veritas has proposed an integrated classification engine [207] for multi-cloud data management, which is used to classify risky and sensitive data worldwide in order to meet data protection requirements.

The Google Cloud Machine Learning Engine is a computing platform that provides training and prediction services [201]. It allows developers to build advanced and complex machine learning models and can be used in the Google Cloud Storage system for data processing. It also automatically scales the number of server clusters to generate results within the access time limit.

Amazon Machine Learning provides developers with services for predictive analysis by building machine learning models [200, 214]. Using an AWS account, a user can select the

Amazon Machine Learning standard setup and select necessary data from Amazon S3 and Redshift. After performing predictive analysis on the selected dataset, Amazon Machine Learning generates predictive or machine learning models.

Microsoft's Azure Machine Learning datastores provide easy data access to clients without requiring hardcoded connection information, such as subscription IDs and token authorization [114, 215]. When a client registers Azure storage solutions as a datastore, Azure Machine Learning is responsible for creating and registering the necessary datastores, which store all related connection information for the corresponding storage account.

Table 6.1 compares some popular smart storage solutions that use machine learning approaches to make them intelligent. Archivist [216] is a mechanism for selecting the storage device unit from solid-state drives (SSDs), conventional hard disk drives (HDDs), and Non-volatile RAM (NVM) for file placement. The Adaptive Resource Management (ARM) system developed by [217] is another machine learning-based technology that enables object-based storage clusters to be self-managing and self-adaptive systems.

On the other hand, AIOps (Artificial Intelligence for IT Operations) platforms designed by Gartner [218, 219] are a combination of big data and machine learning. They can be used to detect failure parameters of cloud object storage services at IBM, such as IBM COS [220]. A Smart Object Storage System (SOSS) [221] is a machine-level storage system that uses machine-level technologies to make it intelligent.

Our proposed approach in this chapter differs in two ways from the existing approaches, as shown in Table 6.1. Here we develop a machine learning unit (called as classification engine). It works as a part of the cloud storage system, unlike the discussed ones that provide a platform where clients can build their own machine learning models using the facilities of that storage system to make the storage system presuming services more advanced. Secondly, the objective of the proposed classification engine unit differs from the discussed ones, and the working process and architecture vary with the demands of associated applications.

In this chapter, our main aim is to predict the storage space for the dataset presented in the client's write query request where multiple database models are involved. In the research world, some popular recent research work uses traditional strategy rather than machine learning for handling multiple databases. Such as, PolyglotHIS [222] uses the query mapper to divide a query into subqueries for accessing data from multiple data storage systems, which come from different database models. Data mapper [223] is used to map the dataset to the corresponding database system by considering the data object's relationship with the database system. Polyglot Persistence Mediator (PPM) [224] selects the database at run time based on tenant's defined information, where it consists with the database schema with the nodes values, SLA information with the functional (ACID transactions, Joins, updates etc.), continuous (e.g., availability, latency, throughput etc.), and non-functional (e.g., scalability, elasticity, durability, replication etc.) parameters as the inputs. Wiki-Health platform [195] considers the ontology engine is used to describe the storage systems by referring to the semantic information of storage systems. These mentioned four research works focus on a different direction for working with multiple database models. However, none of the above works is focused on automatic storage space prediction using machine learning. The approach proposed in this work is thus novel to the best of our knowledge.

Table 6.1: Comparative analysis on systems overview of ML-based storage solutions

Name	Objective	Challenges	Working Principle	Disadvantages
Archivist [216]	Selection of data placement location in storage system for reducing file access latency	Run-time status and diverse properties of both data and storage systems	ML-based mechanism to optimize access pattern prediction for data file placement at runtime in storage system	Data movement cost and data storage capacity as negligible by using HDD.
ARM system [217]	Develop a self-managing and self-adaptive storage cluster	In the presence of diverse workload patterns and resource bottlenecks	ML-based system adaptation technique to track hotspot performance to take correct decisions for optimizing future performance	Considers each and every node in the cluster has the same configuration.
AIOps [219]	Design an automated system to discover failure parameters in IBM COS	Detect, predict and prevent failures and performance slowdowns that could impact users	A data-driven ML-based platform to find out the root cause of performance slowdown by data analysis	Background jobs, load imbalance, and resource bottlenecks have not been considered which may affect the system's performance.
SOSS System [221]	Design an intelligent storage system to satisfy the changing attribute pattern	Object, composed of application data and data attribute, is considered as a storage unit to accomplish the predicting job	Adopting the storage system to achieve intelligence such as pattern recognition, predicting object properties, and an adaptive cache replacement policy	The complex nature of the storage system makes it difficult to handle
RSoS System [46]	Storage space prediction for health data based on its data structure	Handling Big Data Variety Issues of Health Data	The ML-based framework analyzes the system behavior corresponding to input data and matches the related storage device	Background job affects the value of system monitoring output when an input hits the system

6.4 Classification Engine Framework

6.4.1 Architecture

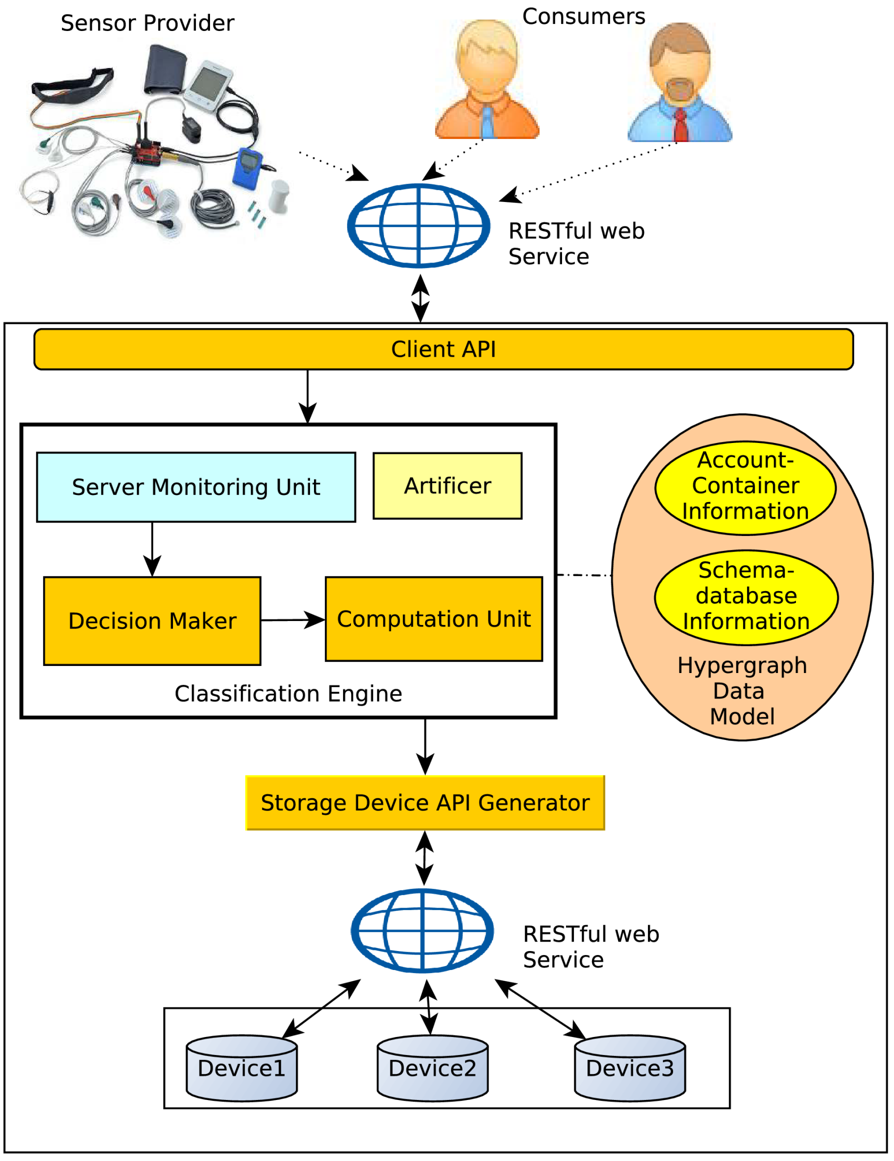


Figure 6.1: Architecture of RSoS System for Automatic Storage Space Prediction

The RSoS cloud storage platform is used to describe the framework of the desired classification engine due to its application dependency. The architecture of the designed classification engine unit associated with the RSoS system is illustrated in Figure 6.1. The classification engine framework is composed of four main components: the Server Monitoring Unit, the Artificer, the Decision Maker, and the Computation Unit.

Server Monitoring Unit collects the information of cloud storage server (e.g., RSoS system) subparts (viz., load, memory, input/output, processor) when a write query hits this system. *Artificer* is responsible for running the data type prediction jobs. If it finds the storage space information of the incoming data is present in the hypergraph data model, it stops the job. Otherwise, it continues further operations. The sub-components (viz., feature selection approach and classifier) of *Decision Maker* are dedicated to predicting the data type of this arrived dataset. *Computation unit* collects the corresponding information of storage space (viz., database name, storage device URI, attribute names) for this dataset which is required to update the metadata (e.g., hypergraph data model).

The *Server Monitoring Unit* of the classification engine framework collects information about the different subparts of the cloud storage server, such as the load, memory, input/output, and processor, when a write query is received. The *Artificer* is responsible for running data type prediction jobs. If it finds that the storage space information of the incoming data is already present in the hypergraph data model, it stops the job. Otherwise, it continues with further operations. The *Decision Maker*, which includes sub-components such as a feature selection approach and a classifier, is used to predict the data type of the incoming dataset. The *Computation Unit* collects the necessary information about the storage space, such as the database name, storage device URI, and attribute names, that is needed to update the metadata, such as the hypergraph data model, for this dataset.

The classification engine is connected to three components of the RSoS system: the Client API, the Storage Device API Generator, and the Hypergraph Data Model. When a query is received from consumers, it is passed through the Client API to the classification engine unit. The Client API also serves as a communication medium between the RSoS system and consumers. The classification engine then informs the Storage Device API Generator about the storage space for the dataset included in the received query. The Storage Device API Generator converts this query into a form that can be understood by the storage device and generates the corresponding storage device URI, which varies based on the storage space value and the type of query (e.g., read, write, delete). Finally, the classification engine updates the Hypergraph Data Model with the generated storage space information, which is used by the classification engine to determine the storage space for similar types of datasets in the future.

6.4.2 Workflow

The classification engine follows a workflow model presented in Figure 6.2 to predict the object storage space of an unknown write query, taking into consideration the RSoS cloud storage platform. At the initial stage, when the write query hits the classification engine, two parallel tasks are performed: the *Server Monitoring Unit* monitors the server status, and the Artificer checks the contents of the graphml.xml file. The graphml.xml file is the written format of the hypergraph data model in a file structure.

The *Server Monitoring Unit* observes the server status value related to resource performance metrics (e.g., memory, processor, Input/output, and load)[225] when a JSON query hits the RSoS server. The generated server-status value set corresponding to the same query is not the same every time, but the fluctuation value is usually negligible across different

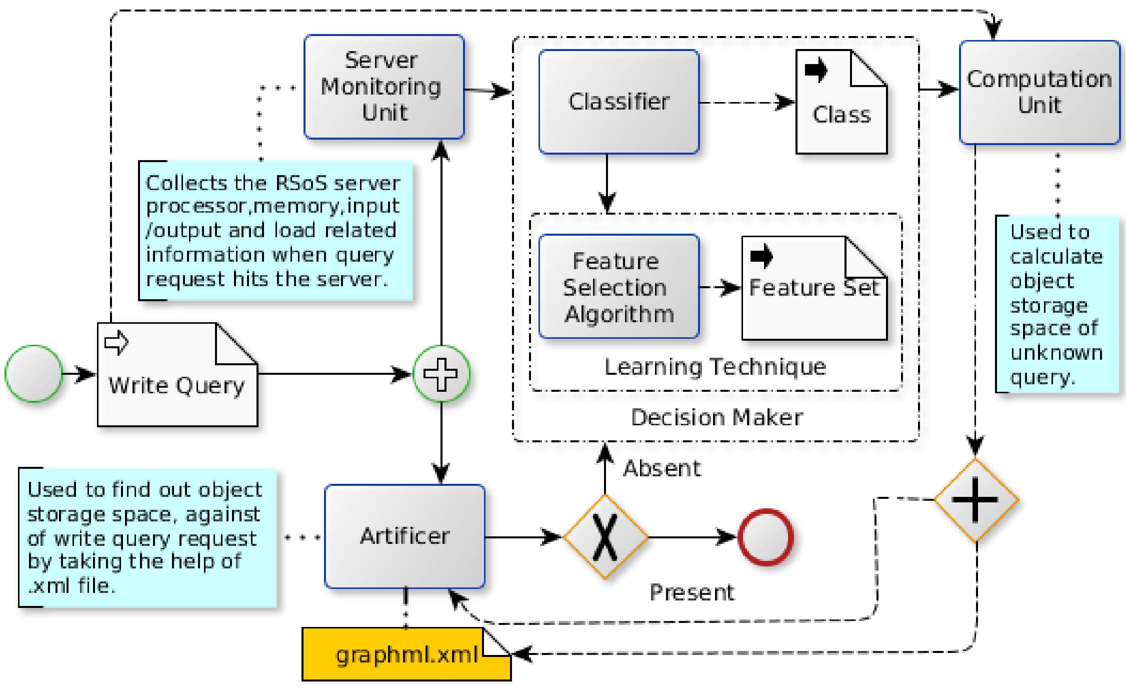

Server Monitoring Unit
Classifier
Class
Computation Unit
Collects the RSoS server processor,memory,input /output and load related information when query request hits the server.
Feature Selection Algorithm
Feature Set
Learning Technique
Decision Maker
Used to calculate object storage space of unknown query.
Write Query
Absent
Used to find out object storage space, against of write query request by taking the help of .xml file.
Artificer
Present
graphml.xml

Figure 6.2: Workflow model of proposed classification engine

queries. These characteristics help the classification engine to decide the class of the unknown query.

The *Artificer* acts as an investigator to find out the object storage space of the incoming write query in the graphml.xml file. If it achieves the desired aim, the classification engine stops its tasks. Otherwise, it starts the *Decision Maker* activity to move forward. The *Decision Maker* is the combination of two activities, the *Learning Technique* and the *Classifier*.

The classifier communicates with a learning technique to determine the class value (i.e., data type) of the dataset in the unknown write query. This process involves several steps. First, the learning technique performs its tasks offline. It trains a set of server monitoring datasets whose class values are known and extracts a set of features for classification. Here, we consider three class values: (1) File, (2) Sensor, and (3) Document. Second, using the selected features, the classifier determines the class value of the received monitoring dataset in real-time."

The Computation Unit generates the object storage space for the unknown arrived write query. For generating object storage space, the object key-value, attribute names, and database names are needed to be known. The received 'time' value is varied with every incoming query. Therefore, by default, time is considered as an object key. After receiving the write query, the computation unit can determine the attribute names. The database name is obtained from the class value obtained from the Decision Maker. Each class value corresponds to a database name such as 'HadoopDFS' for 'File', 'Cassandra' for 'Sensor' and 'MongoDB' for 'Document'. The computation unit updates the graphml.xml file with the predicted object storage space information, and at the same time, it sends this information to the Artificer.

6.4.3 Testing Platform

When a dataset hits the RSoS server, it comes as a query request. These queries are of three types: write, read, and delete. In this experiment, we focus on the write query.

We use three types of data for the experiment: sensor, document, and file, as shown in Table 6.3. Real health sensors send tuple-structured datasets within write queries, which are considered as sensor type data. These data come infrequently. We consider five types of sensors: Temperature, ECG, Pulse, Airflow, and Oxygen. Consumers, mainly health workers, send documents or file-type datasets to the RSoS server in the write query format. For practical simulation, we consider three consumers who send medical images (as file type), patient photos (as file type), and patient information (as document type) individually.

Table 6.2 compares different technical parameters between the RSoS system and four other machine learning-based storage solutions. It can be seen that none of the solutions handle more than one type of data format or three different types of database models under a single platform like RSoS. Additionally, two recent technological considerations, Ganglia and RestAPI, have been used to set up the experiments by RSoS, while the others use classical technologies such as RAM, LRU, DLR, etc. Some storage solutions use benchmarks to set up their experimental parameters to measure their task performance. In the case of RSoS, the parameters have been chosen according to the components selected for the classification engine unit. These factors make us reconsider the design of the experimental components.

Table 6.2: Comparative study on technical parameters of ML-based storage solutions

Systems	Data Store Used	Data Considered	Technology Considered	Benchmark used for experiment
Archivist [216]	HDDs, SSDs, and NVM	Log files, Configuration files	LRU, Baseline and optimized Algorithm.	Filebench Benchmark including WebProxy, WebServer, Fileserver, Varmail.
ARM System [217]	Ceph [118]	Data File	Linux SAR utility, Ceph default, Baseline, DLR, ARM	NSF Cloud's Chameleon testbed, COSBench Benchmark version 0.4.2 , sysbench CPU Benchmark
AIOps [219]	IBM Cloud Object Storage [220], HDFS [193]	Log file (Access logs, Connectivity logs)	Apache Spark, Elastic search, Apache Kafka, Grafana, Slack	N. A.
SOSS System [221]	Object Storage System (OSS) [226]	Application data, attribute data	Switch, RAM, CPU, EPROM, Disk	N. A.
RSoS System	Cassandra, MongoDB, HDFS	File Data, Document Data, Sensor Data	Ganglia, RestAPI	N. A.

Table 6.3: Example of query set for different users

Data source	Example	Query Structure
Sensor	Pulse	{"container":"U009_ecg_12348", "account":"U101", "operation":"Store", "datagroups":{ "patientid" : "anita123", "pulse" : "74", "Time" : "Tue May 08 16:06:53 GMT+05:30 2018"}}
	Temper-ature	{"container":"U009_ecg_12348", "account":"U101", "operation":"Store", "datagroups":{ "patientid" : "anita123", "temperature" : "33.2", "Time" : "Tue May 08 17:06:53 GMT+05:30 2018" }}
	Oxygen	{"container":"U009_ecg_12348", "account":"U101", "operation":"Store", "datagroups":{ "patientid" : "anita123", "spo2" : "99", "Time" : "Tue May 08 18:06:53 GMT+05:30 2018" }}
	Airflow	{"container":"U009_ecg_12348", "account":"U101", "operation":"Store", "datagroups": { "patientid" : "anita123", "airflow" : "12", "Time" : "Tue May 08 19:06:53 GMT+05:30 2018" }}
	ECG	{"container":"U009_ecg_12348", "account":"U101", "operation":"Store", "datagroups": { "patientid" : "anita123", "ecg" : "10", "Time" : "Tue May 08 20:06:53 GMT+05:30 2018" }}

Continued on next page…

Table 6.3 – *Continued*

Consumer1	Patient Details	{"container":"U009_ecg_12348",
		"account":"U101",
		"operation":"Store",
		"datagroups":{
		{"familyHistory": {
		"father":{"history": ["High blood pressure",
		"Diabetes"]},
		"mother":{"history": ["Diabetes", "Asthma"]}},
		"generalObservation": ["In pain", "Looks unwell",
		"Appears very thin"],
		"habit": ["Smoking", "Tobacco chewing"],
		"medicalHistory": ["High blood pressure",
		"Stroke", "Diabetes"],
		"patientDemographicInfo": {
		    "address": "xyz",
		    "age": "31 years, 25 days",
		    "email": "xyz@xyz.in",
		    "gender": "male",
		    "occupation": "business",
		    "patient_id": "6",
		    "patient_name": "xyz",
		    "ph": "xyz"},}}}

Consumer2	Patient Medical Image	{"account":"U104",
		"operation":"Store",
		"container":"ecg",
		"datagroups":{
		    "id":"BenchmarkGain1",
		    "remotefiledata":"xyz....",
		    "fileextension":"png",}}

RSoS Server Monitoring

Ganglia 3.6.0 [173] plug-ins are used in this scenario, in which we observe the experimental scenario for one hour. Within this time, four sensors and three consumers randomly make write query requests. The RSoS system runs on our local workstation during this experiment. Ganglia begins monitoring the server status of the RSoS system in four parts (processor, memory, input/output, and load [225]) from the first query request and stops at the last query request. Whenever Ganglia encounters a query request, it generates a corresponding monitored value matrix. The dimension of this matrix is (query number $\times 18$), where 18 is the number of properties that show the corresponding status values of the RSoS server node's system subparts (memory, input-output, load, CPU) for that query. Each query has a different set of property values, but the property names are the same in every query. These property values act as a set of features that help the classification engine distinguish the dataset according to its types. If we analyze the property value sets from query to query, we may notice a minor difference.

Monitored Dataset Processing

After collecting the RSoS server monitoring dataset, we need to use it to train a classification engine to predict the object storage space of incoming data queries. For experimental purposes, we divide the collected dataset into two parts: a training dataset and a test dataset. Both the training and test datasets are in .csv format and have an additional column representing the class. The class attribute indicates the type of query dataset (document, file, or sensor). The class value is known for the training dataset, but it must be predicted for the test dataset.

Feature Selection

Feature selection is the first step in the classification engine process. In this step, it is necessary to select features that will effectively distinguish between different classes. These techniques are applied to the training dataset. There are 18 property values, but not all of them are equally necessary for differentiating the classes. Therefore, we use some feature selection algorithms to select the relevant properties.

In this experiment, we apply feature selection algorithms that select features (or properties) based on label information. Three feature selection algorithms are used from three different subcategories: (1) Fisher score, (2) F-Score, and (3) Lll21. Fisher score [227] is a supervised, similarity-based feature selection algorithm that selects properties whose values are similar for the same class and dissimilar for different classes. F-Score [228] is a supervised statistical feature selection criterion that selects properties by measuring the statistical value of each feature individually. Lll21 [229] is a supervised, sparse learning-based feature selection algorithm that selects properties using sparsity-induced regularization terms by making feature coefficients small or zero. This feature selection algorithm considers the concept of multiple classes or multiple targets during feature selection.

Dataset Classification

In the next step, our goal is to divide the collected dataset based on their data types or class values. For the training dataset, the class values are known. There are three data types, representing three class labels: sensor, file, and document. The classifier learns from the training dataset, which is divided into these three classes. This knowledge is then used to predict the class value of the test dataset.

In this experiment, we consider three supervised classification algorithms: Support Vector Machine (SVM), K-Nearest Neighbors (K-NN), and Neural Network (NN). K-NN [230] follows the "closest point search" mechanism, which means it assigns unknown data points to the class in which the majority of their K nearest neighbors fall. In this experiment, the value of K is 5. SVM [231] is also known as a "discriminative classifier" because it calculates the optimal hyperplane separating the classes from the training dataset and uses this hyperplane to determine the class of unknown data points. In this experiment, three types of SVM are used: Linear SVM, Radial basis function (RBF) SVM, and Polynomial (Poly) SVM. The value of the regularization parameter C is 1.0 for all three SVM types. RBF SVM and Poly SVM use a gamma value of 2.0, and Poly SVM uses a degree value of 2.0. Neural network [232] follows the mechanism of the human mind. It generates rules or functionality from the training dataset using a multi-layer approach and applies these rules to determine the class of unknown data points. In this experiment, the neural network uses three hidden layers with thirty neurons in each layer and the ReLU activation function. The Adam weight optimization algorithm is used for the hidden layers, and 200 epochs are used.

Table 6.4: Comparative study on experimental parameters of ML-based storage solutions

Systems	ML Framework Used	ML Algorithm Used	ML Algorithm Compared	Input Parameter	Comparison Parameters
Archivist [216]	Offline Classifier	Neural Network (16-Neuron, 32-Neuron)	Naive Bayes Classifier and Support Vector Machine	Offsetw, offsetr, lengthw, lengthr, spanw, spanr	Classifier Accuracy, File Access Latency
ARM System [217]	Reinforcement Learning	TensorFlow Neural Network	N. A.	System-level performance metrics of Ceph Storage such as state and reward information	I/O and Network Interference (Average read and write response time)
AIOps [219]	Big data analysis platform	Smart group By method, Multivariate Anomaly Detection Algorithm, Feature Isolation Technique, Map-Reduce Programming Model	N. A.	Operation type, bucket and object names of Access logs, HTTP return code, start and end times of the operation, and various latency statistics	Anomaly Score (Z-Score) of the aggregated latency.
SOSS System [221]	Object Classifier	Pattern Classification	N. A.	Object attribute, Access-based object attribute (including application assistance and existing user input), Inter-object relationships	N. A.
RSoS System	Classification Engine	K Nearest Neighbor (K-NN) Classifier	Linear SVM, Poly SVM, RBF SVM, Neural Network	proc_run, load_five, disk_free, proc_total, load_one, mem_cached, bytes_in, bytes_out, mem_free, mem_buffers pkts_in, load_fifteen, cpu_idle, cpu_user, cpu_nice, cpu_system, cpu_wio, pkts_out	Classifier Accuracy, Precision, Recall

Dataset for Prediction

After learning about classification, the next goal of the classification engine is to predict the data type of incoming datasets. To do this, SVM, neural network, and K-NN classifiers are used. The dataset prediction process goes through a few steps. First, the classifiers identify the class value (sensor, file, or document) of the dataset in the incoming query request. Then, the classification engine generates the corresponding object storage space for this query and writes the related information into the hypergraph data model if it is not already present there.

Table 6.4 shows that different storage technologies use different machine learning approaches and parameters to improve the performance of their storage solutions. It can be seen that classification-based technologies are mainly used for this type of prediction task. The effectiveness of the RSoS system also depends on the performance of the designed classification engine unit. Therefore, three performance parameters of the classification engine are considered that provide a better analysis compared to other storage technologies, since they do not take into account the performance of the individual components. Input parameters or feature set selection are important parameters for any classification task. As can be seen from the table, the RSoS system uses many important features as input parameters compared to other storage solutions, which helps to design the classification engine for the RSoS system. We need to choose an appropriate feature selection algorithm and corresponding classifier for this important task. As shown in the results in the next section, it can be seen that the Lll21 feature selection algorithm and K-NN classifier are the most suitable components for this classification engine.

6.4.4 Results and Discussion

There are two main components needed to design the classification engine: a feature selection algorithm and a classifier. Therefore, we compare different feature selection algorithms and classifiers to find a suitable combination for this classification engine. The results are shown in Table 6.6.

According to the results, the input to the feature selection algorithm is the output generated by ganglia. When the RSoS server receives a query request, it passes through ganglia, and the feature selection algorithm selects some features from these outputs. In this case, the top 5 features are selected from 18 features. The selected feature sets by the three mentioned feature selection algorithms are presented separately in Listing 6.9. The classifiers use these features to identify the class value of the test dataset.

```
Fisher score: {proc_run, load_five, disk_free, proc_total, load_one}
F-Score: {proc_run, load_five, disk_free, proc_total, load_one}
Ll121: {mem_cached, bytes_in, bytes_out, mem_free, mem_buffers}
```

Listing 6.9: Selected Feature-set by the specified feature selection algorithms

The results are shown in two separate tables. Table 6.5 is for the test dataset and only holds feature values. Table 6.6 consists of the original class values and the predicted class values. The pointer column in Table 6.5 and Table 6.6 is used to connect these two tables. If we analyze Table 6.5 and Table 6.6, we will find that some rows are marked and some are unmarked. The input feature values of the marked data rows are present in the training dataset.

The class column of Table 6.6 represents the original class values of the input dataset. In Table 6.6, we present the outputs of three feature selection algorithms: Fisher Score, FScore, and Lll21, after applying 3 different SVMs (Linear SVM, Poly SVM, RBF SVM), Neural Network, and K-NN classifier as Pfisher_Linear, Pfisher_Poly, Pfisher_RBF, Pfisher_Neural, Pfisher_K-NN, PFScore_Linear, PFScore_Poly, PFScore_RBF, PFScore_Neural, PFScore_K-NN, respectively. For the Lll21 feature selection algorithm, we only consider Linear SVM, Neural Network, and K-NN classifier, which are represented as PLll21_Linear, PLll21_Neural PLll21_K-NN, respectively.

Accuracy, precision, and recall are used to evaluate the performance of the designed classification engine [233, 234], as shown in Table 6.7. These metrics are measured based on the predictions made on the test data. The values of these metrics vary depending on the algorithms applied (the combination of the feature selection algorithm and classifier). Accuracy measures the correctness of the prediction. Precision is used to measure the positive predictive value, and recall is used to calculate the true positive rate. These two metrics are calculated based on the class items.

The Decision Maker is the main backbone of the classification engine unit. The entire system can be compromised by selecting the wrong components for the Decision Maker. Figure 6.3 presents the visualization of the performance metric values, which helps to make a fair decision. The three sub-plots compare the precision (Fig. 6.3(a)), recall (Fig. 6.3(b)), and accuracy (Fig. 6.3(c)) values between 13 feature selection algorithms and classifier algorithms. Precision and recall are calculated based on three different data types (Document, File, and Sensor).

The combination of the Lll21 feature selection algorithm and K-NN classifier gives a 100% accuracy rate. The Fisher-Score feature selection algorithm gives a very low accuracy rate (13.33%) with a neural network classifier. The combination of the Lll21 feature selection algorithm and K-NN classifier has the highest precision and recall values, at 100% for all three class items.

Table 6.5: Example of feature values used for testing purpose

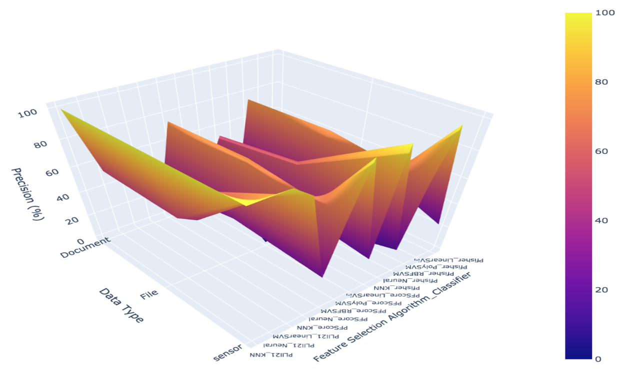
Pointer	proc_run	load_five	disk_free	mem_cached	pkts_in	bytes_in	bytes_out	mem_free	load_fifteen	cpu_idle	cpu_user	cpu_nice	mem_buffers	cpu_system	cpu_wio	proc_total	pkts_out	load_one
Test1	**1**	**0.53**	**860.35**	**22568560**	**1.22**	**198.14**	**301.85**	**611476**	**0.67**	**82.2**	**14**	**0**	**617340**	**2.8**	**1**	**917**	**2.17**	**0.49**
Test3	**0**	**0.59**	**860.37**	**22487480**	**7.4**	**1221.02**	**939.57**	**587852**	**0.67**	**79.5**	**13.5**	**2.6**	**617972**	**3.3**	**1.1**	**842**	**7.97**	**0.41**
Test5	**1**	**0.7**	**860.37**	**22545304**	**0.85**	**122.24**	**221.25**	**350468**	**0.69**	**85.6**	**11.8**	**0**	**618544**	**2.2**	**0.3**	**835**	**1.65**	**0.63**
Test7	**3**	**0.91**	**860.37**	**22537512**	**0.72**	**79.77**	**163.7**	**301044**	**0.77**	**84.5**	**11.6**	**0**	**618992**	**3.3**	**0.7**	**831**	**1.37**	**1.26**
Test9	**1**	**0.62**	**860.37**	**22451576**	**0.75**	**94.33**	**176.4**	**277464**	**0.71**	**86.3**	**11.4**	**0**	**619500**	**2.2**	**0.2**	**830**	**1.45**	**0.28**
Test10	**0**	**0.56**	**860.37**	**22439988**	**1.62**	**334.98**	**299.58**	**308536**	**0.67**	**86.4**	**10.9**	**0**	**619700**	**2.6**	**0.1**	**832**	**2.27**	**0.41**
Test11	**0**	**0.52**	**860.37**	**22378904**	**1.15**	**146.03**	**276.75**	**251528**	**0.62**	**86.2**	**10.8**	**0**	**620436**	**2.4**	**0.6**	**829**	**1.93**	**0.36**
Test13	**0**	**0.45**	**860.37**	**22285800**	**1.55**	**277.24**	**335.15**	**289580**	**0.56**	**86.5**	**10.9**	**0**	**620920**	**2.4**	**0.2**	**838**	**2.15**	**0.48**
Test14	1	0.56	860.37	22252728	0.65	85.9	120.95	307456	0.58	84.8	12	0	621160	3.1	0.1	831	1.5	0.71
Test15	2	0.48	860.37	22249184	1.03	182.88	261.73	263672	0.55	86.5	10.9	0.1	621416	2.5	0.1	830	1.73	0.39
Test17	**2**	**0.45**	**860.37**	**22152916**	**0.8**	**144.44**	**205.16**	**289816**	**0.51**	**84.3**	**11.6**	**0**	**621892**	**3.3**	**0.7**	**832**	**1.55**	**0.5**
Test19	1	0.55	860.37	22132340	0.92	198.04	215.01	306696	0.52	86.5	10.8	0	622380	2.6	0.1	835	1.67	0.71
Test21	**2**	**0.35**	**920.50**	**3281772**	**29.1**	**41004.89**	**1061.7**	**21286644**	**0.41**	**90.7**	**5.9**	**0**	**589016**	**3.3**	**0.2**	**859**	**12.67**	**0.34**
Test26	1	1.34	920.49	3234444	28.67	38324.01	715.25	21375240	1.04	68.8	16.5	0	604732	12.6	2.1	864	6.9	1.31
Test27	1	1.34	920.49	3234444	28.67	38324.01	715.25	21375240	1.04	72.2	14.5	0	604732	11.4	1.8	864	6.9	1.31
Test28	1	1.34	920.49	3234444	28.67	38324.01	715.25	21375240	1.04	72.2	14.5	0	604732	11.4	1.8	864	6.9	1.31

Table 6.6: Comparative resultset shows the test output of feature selection algorithms with classification algorithm where Sensor = S, Document = D, File = F

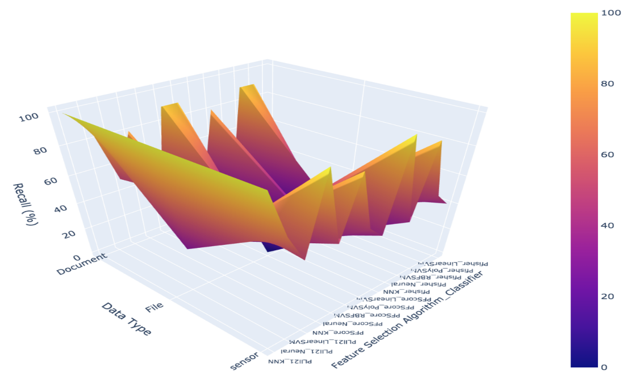
Pointer	Class	Pfisher_Linear	Pfisher_Poly	Pfisher_RBF	Pfisher_Neural	Pfisher_K-NN	PFScore_Linear	PFScore_Poly	PFScore_RBF	PFScore_Neural	PFScore_K-NN	PLll21_Linear	PLll21_Neural	PLll21_K-NN
Test1	**D**	**S**	**F**	**D**	**S**	**F**	**S**	**F**	**D**	**S**	**F**	**D**	**D**	**D**
Test3	**D**	**S**	**D**	**D**	**S**	**D**	**D**	**D**	**D**	**F**	**D**	**F**	**D**	**D**
Test5	**D**	**S**	**D**	**D**	**S**	**D**	**S**	**D**	**D**	**S**	**D**	**D**	**D**	**D**
Test7	**D**	**D**	**D**	**D**	**S**	**D**	**S**	**D**	**D**	**S**	**D**	**D**	**D**	**D**
Test9	**D**	**D**	**D**	**D**	**S**	**D**	**S**	**D**	**D**	**S**	**D**	**D**	**D**	**D**
Test10	**D**	**S**	**D**	**D**	**S**	**D**	**D**	**D**	**D**	**D**	**D**	**D**	**D**	**D**
Test11	**F**	**D**	**D**	**S**	**S**	**D**	**S**	**D**	**S**	**S**	**F**	**F**	**F**	**F**
Test13	**F**	**F**	**D**	**F**	**S**	**F**	**S**	**D**	**F**	**F**	**F**	**F**	**D**	**F**
Test14	F	S	D	S	S	D	D	D	S	S	D	F	D	F
Test15	F	S	D	S	S	F	D	D	S	S	F	D	D	F
Test17	**F**	**S**	**D**	**F**	**S**	**D**	**S**	**D**	**F**	**S**	**D**	**D**	**F**	**F**
Test19	F	S	D	D	S	D	S	D	D	S	D	D	D	F
Test21	**S**	**D**	**F**	**S**	**F**	**S**	**S**	**F**	**S**	**S**	**S**	**S**	**S**	**S**
Test26	S	D	S	S	F	S	F	S	S	F	S	F	D	S
Test27	S	S	S	S	S	S	S	S	S	F	S	F	S	S
Test28	S	D	S	S	S	S	D	S	S	S	S	D	S	S

Table 6.7: Performance Matrix of the Combined Feature Selection Algorithm and Classifier.

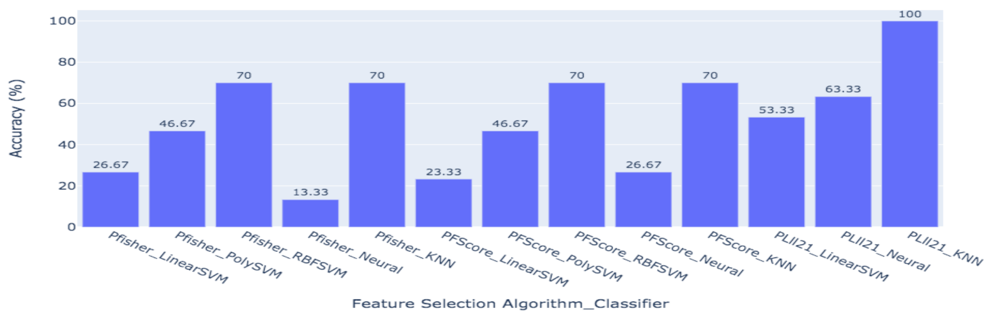
Feature Selection Algorithm_Classifier	Accuracy (%)	Precision _Document (%)	Precision _File (%)	Precision _Sensor (%)	Recall _Document (%)	Recall _File (%)	Recall _Sensor (%)
Pfisher_LinearSVM	26.67	25	50	22.22	20	20	40
Pfisher_PolySVM	46.67	47.37	0	100	90	0	50
Pfisher_RBFSVM	70	75	75	64.28	90	30	90
Pfisher_Neural	13.33	0	0	16.67	0	0	40
Pfisher_K-NN	70	53.3	60	100	80	30	100
PFScore_LinearSVM	23.33	25	33.33	21.05	20	10	40
PFScore_PolySVM	46.67	47.36	0	100	90	0	50
PFScore_RBFSVM	70	75	75	64.28	90	30	90
PFScore_Neural	26.67	25	33.33	25	10	20	50
PFScore_K-NN	70	57.14	60	90.9	80	30	100
PLl21_LinearSVM	53.33	45.45	42.85	100	50	60	50
PLl21_Neural	63.33	52.94	50	100	90	30	70
PLl21_K-NN	100	100	100	100	100	100	100



(a) Precision values of the decision maker



(b) Recall values of the decision maker



(c) Accuracy values of the decision maker

Figure 6.3: Comparative study of the performance matrix values of the components of the decision maker

6.5 Conclusion

In this chapter, we have proposed a way to convert an object-based storage system like RSoS into an automated system by automatically identifying the storage spaces for incoming queries. To do this, we have built a classification engine to categorize data based on their structure format. The RSoS system stores data based on its structure by implementing an object storage space, which is a combination of the database name, data attribute name, and object key. The storage system can accommodate three types of data (sensor, document, and file) on a single storage platform. The integration of this classification engine with the RSoS system enables it to predict the corresponding object storage space for incoming queries without manual intervention.

The effectiveness of the classification engine depends on the performance of its two main components: a feature selection approach and a classifier. Three supervised feature selection algorithms from three categories, Fisher score from similarity-based, F-score from statistical-based, and Lll21 from sparse learning-based approaches, have been compared. Three classifiers, SVM, K-NN, and Neural Network, have been used with the mentioned feature selection algorithms. After analyzing the results, the Lll21 feature selector combined with the K-NN classifier provides the best performance.

Chapter 7

Conclusion and Future Work

7.1 Outcome of the thesis

The outcomes of the thesis are as follows:

- In this thesis, we have discussed big data and their characteristics. It is shown that the health data follows all the big data properties namely, volume, variety, velocity, veracity, and value. Handling volume and variety properties of health data is the main theme of our research. In the thesis, we have proposed new algorithms for big data in general. But they are validated in experiments where dataset (sometimes real) is drawn from the health domain
- Next, we discussed management system for handling big data. In this context, we have also discussed several cloud storage systems and database systems that can support big data from a storage perspective. A state of the art survey has been reported regarding recent storage technologies and solutions for handling big data properties.
- We have designed a storage model for supporting big data variety property in a unified platform.
- One major outcome of this thesis is to design a novel cloud storage system named object based schema oriented cloud storage system (RSoS System) to address big data volume and variety property. It is already known that cloud storage systems are efficient to support the big data volume property. In this thesis, our main concern is supporting the big data variety property. Here, it is shown that this new system adequately addresses the big data variety issue and reduces task execution time.
- Load balancing techniques are discussed in a cloud platform and a thorough survey has been carried out about the challenges of task distribution among the available resources and their existing solutions. Two novel load balancing algorithms are proposed here to efficiently distribute tasks to resources. The first algorithm is better for streaming applications. The second one is inspired the foraging mechanism of honeybees. This algorithm is shown to be energy-efficient.
- Finally, we enhanced the capability of the proposed RSoS cloud storage system by incorporating some machine learning techniques. The intelligence so introduced helps in deciding the appropriate object storage space for an incoming data set.

7.2 Future Work

In the future, our target is to move forward with the proposed RSoS cloud storage system and make it a more advanced and user-friendly storage system. For this purpose, some research areas have been selected for further exploration. These are enumerated below.

- We want to incorporate a load balancing technique within the RSoS system so that RSoS can support multi-tenant service requests efficiently.
- Apply machine learning techniques so that RSoS system becomes energy efficient.

Bibliography

[1] Chevvanthi E. *Cloud storage system design.* https://medium.com/@chevv/cloud-storage-system-design-21322e2f9551. [Online; accessed 19-November-2019].

[2] YADULLAH ABIDI. *A brief history of storage devices.* https://www.himss.org/resources/blending-structured-and-unstructured-data-develop-healthcare-insights. [Online; accessed 19-April-2021].

[3] Chelsey Farris. *History of Cloud Storage.* https://capacity.com/cloud-storage/history-of-cloud-storage/. [Online; accessed 19-April-2021].

[4] IBM. *The official website of IBM cloud.* https://www.ibm.com/cloud. [Online; accessed 19-April-2021].

[5] Keith D. Foote. *A Brief History of Cloud Computing.* https://www.dataversity.net/brief-history-cloud-computing/. [Online; accessed 19-April-2021].

[6] Paul McFedries. *Cloud computing: beyond the hype.* Citeseer, 2012.

[7] salesforce. *The official website of salesforce.* https://www.salesforce.com/in/?ir=1. [Online; accessed 19-April-2021].

[8] *Amazon S3.* https://aws.amazon.com/s3/.

[9] netflix. *The official website of netflix.* https://www.netflix.com/in/. [Online; accessed 19-April-2021].

[10] icloud. *The official website of Apple iCloud.* https://www.apple.com/in/icloud/. [Online; accessed 19-April-2021].

[11] oracle. *Welcome to Oracle Cloud Infrastructure.* https://docs.oracle.com/en-us/iaas/Content/GSG/Concepts/baremetalintro.htm. [Online; accessed 19-April-2021].

[12] Blesson Varghese. *History of the cloud.* https://www.bcs.org/articles-opinion-and-research/history-of-the-cloud/. [Online; accessed 19-April-2021].

[13] Harald Schützeichel. *A tribute to the inventor of Pay-as-you-go: Jürgen Gehr.* https://www.sun-connect-news.org/de/articles/technology/details/a-tribute-to-the-inventor-of-pay-as-you-go-juergen-gehr/. [Online; accessed 19-April-2021].

[14] Nicolas Serrano, Gorka Gallardo, and Josune Hernantes. "Infrastructure as a service and cloud technologies". In: *IEEE Software* 32.2 (2015), pp. 30–36.

[15] *Rackspace.* https://www.rackspace.com/cloud_hosting_products/files. [Online; accessed 8-May-2021].

[16] Keke Gai and Annette Steenkamp. “A feasibility study of Platform-as-a-Service using cloud computing for a global service organization”. In: *Journal of Information Systems Applied Research* 7.3 (2014), p. 28.

[17] *Microsoft Azure.* https://docs.microsoft.com/en-us/azure/storage/storage-introduction. [Online; accessed 8-November-2020].

[18] KKM Kumar. “Software as a service for efficient cloud computing”. In: *environment* 7 (2014), p. 10.

[19] Blesson Varghese. *History of the cloud.* https://www.bcs.org/articles-opinion-and-research/history-of-the-cloud/. [Online; accessed 19-April-2021].

[20] HOCHUL YU. *Challenge 2: The Story of Digital Storage Device.* https://design4dotme.wordpress.com/2012/04/03/challenge-2-the-story-of-digital-storage-device/. [Online; accessed 19-April-2021].

[21] U.S. Department of the Interior. *Cloud Service Models.* https://www.doi.gov/cloud/service. [Online; accessed 19-April-2021].

[22] Caesar Wu and Rajkumar Buyya. “Chapter 12 - Cloud Storage Basics”. In: *Cloud Data Centers and Cost Modeling.* Ed. by Caesar Wu and Rajkumar Buyya. Morgan Kaufmann, 2015, pp. 425–495. ISBN: 978-0-12-801413-4. DOI: https://doi.org/10.1016/B978-0-12-801413-4.00012-X.

[23] Pierre Bijaoui and Juergen Hasslauer. “Chapter 3 - Storage Technologies”. In: *Designing Storage for Exchange 2007 SP1.* Ed. by Pierre Bijaoui and Juergen Hasslauer. Digital Press Storage Technologies. Digital Press, 2008, pp. 75–116. ISBN: 978-1-55558-308-8. DOI: https://doi.org/10.1016/B978-1-55558-308-8.00003-X. URL: https://www.sciencedirect.com/science/article/pii/B978155558308800003X.

[24] JEFF FOWLER. *WHAT IS A NETWORK ATTACHED STORAGE?* https://sandstormit.com/what-is-a-network-attached-storage/. [Online; accessed 19-April-2021].

[25] EUGENE. *Virtualization Techniques in Cloud Computing.* https://www.sam-solutions.com/blog/virtualization-techniques-in-cloud-computing/. [Online; accessed 19-April-2021].

[26] Brandon Salmon. *Understanding cloud storage models.* https://www.infoworld.com/article/2871290/understanding-cloud-storage-models.html. [Online; accessed 19-April-2021].

[27] Gurudatt Kulkarni, Ramesh Sutar, and Jayant Gambhir. “Cloud computing-Infrastructure as service-Amazon EC2”. In: *International Journal of Engineering Research and Applications* 2 (2012).

[28] openstack. *The official website of openstack.* https://www.openstack.org/. [Online; accessed 19-April-2021].

[29] Brian Curtis. *DIFFERENT TYPES OF CLOUD STORAGE MODELS EXPLAINED.* https://www.yourtechdiet.com/blogs/cloud-storage-models/. [Online; accessed 19-April-2021].

[30] openstack cinder. *The official website of openstack cinder.* https://www.openstack.org/software/releases/mitaka/components/cinder. [Online; accessed 19-April-2021].

[31] Himakshi Goswami. *NAS vs Object Storage: what's best for unstructured data?* https://www.loadbalancer.org/blog/nas-vs-object-storage-whats-the-best-solution/. [Online; accessed 19-April-2021].

[32] Dinesh. *What is DAS, NAS & SAN?* https://techiemaster.wordpress.com/2016/06/12/what-is-das-nas-san/. [Online; accessed 19-April-2021].

[33] Francis X. Diebold. “"Big Data" Dynamic Factor Models for Macroeconomic Measurement and Forecasting”. In: *Advances in Economics and Econometrics: Theory and Applications, Eighth World Congress.* Vol. 3. Cambridge University Press. 2003, pp. 115–122.

[34] Doug Laney et al. “3D data management: Controlling data volume, velocity and variety”. In: *META group research note* 6.70 (2001), p. 1.

[35] Tim O'reilly. *What is web 2.0.* 2005.

[36] Owen O'Malley. “Introduction to Hadoop”. In: *Yahoo Inc* (2008).

[37] Yun Yang, Wenhao Li, and Dong Yuan. *Reliability assurance of big data in the cloud: Cost-effective replication-based storage.* Morgan Kaufmann, 2014.

[38] Liang Zhao et al. *Cloud data management.* Springer, 2014.

[39] ApacheBooster. *DIFFERENT TYPES OF STORAGE | OBJECT VS FILE VS BLOCK STORAGE.* https://apachebooster.com/blog/different-types-of-storage-object-vs-file-vs-block-storage/. [Online; accessed 19-April-2021].

[40] V Spoorthy, M Mamatha, and B Santhosh Kumar. “A survey on data storage and security in cloud computing”. In: *International Journal of Computer Science and Mobile Computing* 3.6 (2014), pp. 306–313.

[41] Josef Spillner, Johannes Müller, and Alexander Schill. “Creating optimal cloud storage systems”. In: *Future Generation Computer Systems* (2013).

[42] Arjun Kumar, HoonJae Lee, and Rajeev Pratap Singh. “Efficient and secure Cloud storage for handling big data”. In: *2012 6th International Conference on New Trends in Information Science, Service Science and Data Mining (ISSDM2012).* IEEE. 2012, pp. 162–166.

[43] Martin Henze et al. "Complying with data handling requirements in cloud storage systems". In: *IEEE Transactions on Cloud Computing* (2020).

[44] Hussain AlJahdali et al. "Multi-tenancy in cloud computing". In: *2014 IEEE 8th international symposium on service oriented system engineering*. IEEE. 2014, pp. 344–351.

[45] Anindita Sarkar et al. "A survey of issues and solutions of health data management systems". In: *Innovations in Systems and Software Engineering* 15 (2019). DOI: 10.1007/s11334-019-00336-4.

[46] Anindita Sarkar Mondal et al. "Object based schema oriented data storage system for supporting heterogeneous data". In: *2016 International Conference on Advances in Computing, Communications and Informatics (ICACCI)*. 2016, pp. 1025–1032.

[47] Anindita Sarkar Mondal and Samiran Chattopadhyay. "A Storage Model for Handling Big Data Variety". In: *Computational Intelligence, Communications, and Business Analytics*. Springer, 2017. DOI: 10.1007/978-981-10-6427-2_5. URL: https://doi.org/10.1007/978-981-10-6427-2_5.

[48] Anindita Sarkar Mondal et al. "Performance analysis of an efficient object-based schema oriented data storage system handling health data". In: *Innovations in Systems and Software Engineering* (2019), pp. 1–15.

[49] Anindita Sarkar and Samiran Chattopadhyay. "Comparative Analysis of Load Balancing Algorithms in Cloud Computing". In: (2021).

[50] Anindita Sarkar Mondal et al. "A Double Threshold-Based Power-Aware Honey Bee Cloud Load Balancing Algorithm". In: *SN Computer Science* 2.5 (2021), pp. 1–16.

[51] Anindita Sarkar Mondal, Kshitij Pant, and Samiran Chattopadhyay. "DRSQ-A Dynamic Resource Service Quality Based Load Balancing Algorithm". In: *International Conference on Computational Intelligence, Communications, and Business Analytics*. Springer. 2018, pp. 97–108.

[52] Anindita Sarkar Mondal, Anirban Mukhopadhyay, and Samiran Chattopadhyay. "Machine learning-driven automatic storage space recommendation for object-based cloud storage system". In: *Complex & Intelligent Systems* (2021), pp. 1–17.

[53] John H Holland et al. "What is a learning classifier system?" In: *International Workshop on Learning Classifier Systems*. Springer. 1999, pp. 3–32.

[54] Douglas Laney. "3D data management: Controlling data volume, velocity and variety". In: *META group research note* 6.70 (2001).

[55] James Manyika et al. *Big data: The next frontier for innovation, competition, and productivity*. McKinsey Global Institute, 2011.

[56] Vinod Saratchandran. *5 Ways Big Data is Changing the Healthcare Industry*. https://www.fingent.com/blog/5-ways-big-data-is-changing-the-healthcare-industry/. [Online; accessed 19-April-2021].

[57] HIMSS. *Healthcare Information and Management Systems Society (HIMSS)*. https://candid.technology/history-of-storage-devices/. [Online; accessed 19-April-2021].

[58] Francesca Bugiotti and Luca Cabibbo. *An Object-Datastore Mapper Supporting NoSQL Database Design*. 2013.

[59] Luca Cabibbo. "Ondm: an object-nosql datastore mapper". In: *Faculty of Engineering, Roma Tre University. Retrieved June 15th* (2013).

[60] Francesca Bugiotti et al. *A Logical Approach to NoSQL Databases*. 2013.

[61] Paolo Atzeni, Francesca Bugiotti, and Luca Rossi. "Uniform access to non-relational database systems: The SOS platform". In: *Advanced Information Systems Engineering*. Springer. 2012, pp. 160–174.

[62] Olivier Curé et al. "On the Potential Integration of an Ontology-Based Data Access Approach in NoSQL Stores". In: *Emerging Intelligent Data and Web Technologies (EIDWT), 2012 Third International Conference on*. 2012, pp. 166–173.

[63] Olivier Curé, Myriam Lamolle, and Chan Le Duc. "Ontology based data integration over document and column family oriented nosql". In: *arXiv preprint arXiv:1307.2603* (2013).

[64] Kiran V K and R Vijayakumar. "Ontology based data integration of NoSQL datastores". In: *Industrial and Information Systems (ICIIS), 2014 9th International Conference on*. IEEE. 2014, pp. 1–6.

[65] *Microsoft Azure*. https://docs.microsoft.com/en-us/azure/storage/common/storage-introduction. [Online; accessed 8-May-2021].

[66] Jeffrey Dean and Sanjay Ghemawat. "MapReduce: simplified data processing on large clusters". In: *Communications of the ACM* 51.1 (2008), pp. 107–113.

[67] Rares Vernica, Michael J Carey, and Chen Li. "Efficient parallel set-similarity joins using MapReduce". In: *Proceedings of the 2010 ACM SIGMOD International Conference on Management of data*. ACM. 2010, pp. 495–506.

[68] Yuting Lin et al. "Llama: leveraging columnar storage for scalable join processing in the mapreduce framework". In: *Proceedings of the 2011 ACM SIGMOD International Conference on Management of data*. ACM. 2011, pp. 961–972.

[69] Shojiro Muro, Tiko Kameda, and Toshimi Minoura. "Multi-version concurrency control scheme for a database system". In: *Journal of Computer and System Sciences* 29.2 (1984), pp. 207–224.

[70] Theo Härder. "Observations on optimistic concurrency control schemes". In: *Information Systems* 9.2 (1984), pp. 111–120.

[71] Bogdan Nicolae. "BlobSeer: Towards efficient data storage management for large-scale, distributed systems". PhD thesis. Université Rennes 1, 2010.

[72] Hsiang-Tsung Kung and John T Robinson. “On optimistic methods for concurrency control”. In: *ACM Transactions on Database Systems (TODS)* 6.2 (1981), pp. 213–226.

[73] Viet-Trung Tran et al. “DStore: An in-memory document-oriented store”. In: (2012).

[74] *CouchDB*. http://couchdb.apache.org/.

[75] Konstantin Shvachko et al. “The hadoop distributed file system”. In: *2010 IEEE 26th symposium on mass storage systems and technologies (MSST)*. IEEE. 2010.

[76] *VoltDB*. https://voltdb.com/.

[77] Carlo Batini et al. “Methodologies for data quality assessment and improvement”. In: *ACM Computing Surveys (CSUR)* 41.3 (2009), p. 16.

[78] Giuseppe DeCandia et al. “Dynamo: amazon’s highly available key-value store”. In: *ACM SIGOPS Operating Systems Review*. Vol. 41. 6. ACM. 2007, pp. 205–220.

[79] Ronald C Taylor. “An overview of the Hadoop/MapReduce/HBase framework and its current applications in bioinformatics”. In: *BMC bioinformatics* 11.12 (2010).

[80] Sanjay Ghemawat, Howard Gobioff, and Shun-Tak Leung. “The Google file system”. In: *ACM SIGOPS operating systems review*. Vol. 37. 5. ACM. 2003, pp. 29–43.

[81] Brian F Cooper et al. “PNUTS: Yahoo!’s hosted data serving platform”. In: *Proceedings of the VLDB Endowment* 1.2 (2008), pp. 1277–1288.

[82] Mayur R. Palankar et al. “Amazon S3 for Science Grids: A Viable Solution?” In: *Proceedings of the 2008 International Workshop on Data-aware Distributed Computing*. New York, NY, USA, 2008.

[83] Judith S Bowman, Sandra L Emerson, and Marcy Darnovsky. *The practical SQL handbook: using structured query language*. Addison-Wesley Reading, Mass., 1996.

[84] Serge Abiteboul. “Querying semi-structured data”. In: *International Conference on Database Theory*. Springer. 1997, pp. 1–18.

[85] Robert Blumberg and Shaku Atre. “The problem with unstructured data”. In: *Dm Review* 13.42-49 (2003), p. 62.

[86] DC Tsichritzis and Frederick H. Lochovsky. “Hierarchical data-base management: A survey”. In: *ACM Computing Surveys (CSUR)* 8.1 (1976), pp. 105–123.

[87] James Rumbaugh et al. *Object-oriented modeling and design*. Vol. 1991. 1. Prentice-hall Englewood Cliffs, NJ, 1991.

[88] Nishtha Jatana et al. “A survey and comparison of relational and non-relational database”. In: *International Journal of Engineering Research & Technology* 1.6 (2012), pp. 1–5.

[89] Salahaldin Juba, Achim Vannahme, and Andrey Volkov. *Learning PostgreSQL*. Packt Publishing Ltd, 2015.

[90] Changlin He. “Survey on NoSQL database technology”. In: *Journal of Applied Science and Engineering Innovation Vol* 2.2 (2015), pp. 50–54.

[91] ABM Moniruzzaman and Syed Akhter Hossain. “Nosql database: New era of databases for big data analytics-classification, characteristics and comparison”. In: *arXiv preprint arXiv:1307.0191* (2013).

[92] Josiah L Carlson. *Redis in action*. Manning, 2013.

[93] David Hows, Peter Membrey, and Eelco Plugge. *MongoDB Basics*. Apress, 2014.

[94] Peter Membrey et al. *The definitive guide to MongoDB: the noSQL database for cloud and desktop computing*. Springer, 2010.

[95] Kyle Banker. *MongoDB in action*. Manning, 2012.

[96] Claudio Tesoriero. *Getting started with OrientDB*. Packt Publishing Ltd, 2013.

[97] Mahesh Lal. *Neo4j graph data modeling*. Packt Publishing Ltd, 2015.

[98] Neo4j. *Website of Neo4j*. https://neo4j.com/. [Online; accessed 19-May-2021]. 2021.

[99] Chengzhang Peng and Zejun Jiang. “Building a cloud storage service system”. In: *Procedia Environmental Sciences* 10 (2011), pp. 691–696.

[100] Dejun Wang. “An efficient cloud storage model for heterogeneous cloud infrastructures”. In: *Procedia engineering* 23 (2011), pp. 510–515.

[101] Arkaitz Ruiz-Alvarez and Marty Humphrey. “An automated approach to cloud storage service selection”. In: *Proceedings of the 2nd international workshop on Scientific cloud computing*. ACM. 2011, pp. 39–48.

[102] Gaurav Kulkarni et al. “Cloud storage architecture”. In: *7th International Conference on Telecommunication Systems, Services, and Applications (TSSA’12)*. IEEE. 2012, pp. 76–81.

[103] Yang Li, Li Guo, and Yike Guo. “CACSS: Towards a Generic Cloud Storage Service.” In: *CLOSER*. 2012.

[104] Xiaojing Jia. “Google cloud computing platform technology architecture and the impact of its cost”. In: *2010 Second World Congress on Software Engineering*. Vol. 2. IEEE. 2010, pp. 17–20.

[105] Ekaba Bisong. “An Overview of Google Cloud Platform Services”. In: *Building Machine Learning and Deep Learning Models on Google Cloud Platform* (2019), pp. 7–10.

[106] *Google Cloud Storage*. https://cloud.google.com/storage/. [Online; accessed 8-May-2021].

[107] Joe Arnold. *Openstack swift: Using, administering, and developing for swift object storage*. O’Reilly Media, Inc., 2014.

[108] Prosunjit Biswas, Farhan Patwa, and Ravi Sandhu. “Content level access control for openstack swift storage”. In: *Proceedings of the 5th ACM Conference on Data and Application Security and Privacy*. ACM. 2015, pp. 123–126.

[109] *Openstack Swift.* https://www.swiftstack.com/docs/introduction/openstack_swift.html. [Online; accessed 19-May-2021].

[110] *Rackspace Object Storage.* https://developer.rackspace.com/docs/user-guides/infrastructure/cloud-config/storage/cloud-files-product-concepts/object-storage/. [Online; accessed 8-May-2021].

[111] *Rackspace Products.* https://www.rackspace.com/cloud_hosting_products/files.. [Online; accessed 8-May-2021].

[112] *Rackspace Architecture.* https://support.rackspace.com/how-to/rackspace-open-cloud-reference-architecture/. [Online; accessed 8-May-2021].

[113] *Rackspace CDN.* https://www.rackspace.com/en-in/cloud/cdn-content-delivery-network. [Online; accessed 8-May-2021].

[114] Scott Klein. "Azure Data Factory". In: *IoT Solutions in Microsoft's Azure IoT Suite: Data Acquisition and Analysis in the Real World.* Apress, 2017, pp. 105–122.

[115] Brad Calder et al. "Windows Azure Storage: a highly available cloud storage service with strong consistency". In: *Proceedings of the Twenty-Third ACM Symposium on Operating Systems Principles.* ACM. 2011, pp. 143–157.

[116] Sogand Shirinbab, Lars Lundberg, and David Erman. "Performance Evaluation of Distributed Storage Systems for Cloud Computing." In: *IJ Comput. Appl.* 20.4 (2013), pp. 195–207.

[117] Bastiaan Stougie et al. *Distributed object storage system.* US Patent 8,386,840. 2013.

[118] Sage A Weil. "Ceph: reliable, scalable, and high-performance distributed storage". PhD thesis. UNIVERSITY OF CALIFORNIA SANTA CRUZ, 2007.

[119] Sage A Weil et al. "Rados: a scalable, reliable storage service for petabyte-scale storage clusters". In: *Proceedings of the 2nd international workshop on Petascale data storage: held in conjunction with Supercomputing'07.* ACM. 2007, pp. 35–44.

[120] Kazutaka Morita. "Sheepdog: Distributed storage system for qemu/kvm". In: *LCA 2010 DS&R miniconf* (2010).

[121] Paulo Maciel et al. "Performance evaluation of sheepdog distributed storage system". In: *2014 IEEE International Conference on Systems, Man, and Cybernetics (SMC).* IEEE. 2014, pp. 3370–3375.

[122] Saurav Haloi. *Apache zookeeper essentials.* Packt Publishing Ltd, 2015.

[123] Stephen Kaisler et al. "Big Data: Issues and Challenges Moving Forward". In: *46th Hawaii International Conference on System Sciences (HICSS'13).* 2013.

[124] Gaurav Kulkarni et al. "Cloud storage architecture". In: *Telecommunication Systems, Services, and Applications (TSSA), 2012 7th International Conference on.* IEEE. 2012, pp. 76–81.

[125] Michael Armbrust et al. “A view of cloud computing”. In: *Communications of the ACM* 53.4 (2010), pp. 50–58.

[126] Arkaitz Ruiz-Alvarez and Marty Humphrey. “An automated approach to cloud storage service selection”. In: *Proceedings of the 2nd international workshop on Scientific cloud computing*. ACM. 2011, pp. 39–48.

[127] Seiichi Yamamoto et al. “Materialized View as a Service for Large-Scale House Log in Smart City”. In: *Cloud Computing Technology and Science (CloudCom), 2013 IEEE 5th International Conference on*. Vol. 2. IEEE. 2013, pp. 311–316.

[128] *Cassandra*. http://cassandra.apache.org/.

[129] *MongoDB*. https://www.mongodb.org/.

[130] Miranda Zhang et al. “An ontology-based system for Cloud infrastructure services’ discovery”. In: *8th international conference on collaborative computing: networking, applications and worksharing (CollaborateCom)*. IEEE. 2012, pp. 524–530.

[131] Kewen Wu, Julita Vassileva, and Yuxiang Zhao. “Understanding users’ intention to switch personal cloud storage services: Evidence from the Chinese market”. In: *Computers in Human Behavior* 68 (2017), pp. 300–314.

[132] Srimanyu Timmaraju, Vadlamani Ravi, and GR Gangadharan. “Ranking of Cloud Services Using Opinion Mining and Multi-Attribute Decision Making: Ranking of Cloud Services Using Opinion Mining and MADM”. In: *Handbook of Research on Advanced Data Mining Techniques and Applications for Business Intelligence*. IGI Global, 2017, pp. 379–396.

[133] Leland P Sidwell. *System for modifying JCL parameters to optimize data storage allocations*. 2000.

[134] Arkaitz Ruiz-Alvarez and Marty Humphrey. “A model and decision procedure for data storage in cloud computing”. In: *12th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGrid’12)*. IEEE. 2012, pp. 572–579.

[135] P. R. Panda et al. “Data and Memory Optimization Techniques for Embedded Systems”. In: *ACM Transactions Des. Autom. Electron. Syst.* (2001).

[136] Kohei Takahashi et al. “Design and implementation of service api for large-scale house log in smart city cloud”. In: *4th IEEE International Conference on Cloud Computing Technology and Science Proceedings*. IEEE. 2012, pp. 815–820.

[137] Gaurav Mathur et al. “Capsule: An Energy-optimized Object Storage System for Memory-constrained Sensor Devices”. In: ACM, 2006.

[138] Venkat N Gudivada, Ricardo Baeza-Yates, and Vijay V Raghavan. “Big data: promises and problems”. In: *IEEE Computer Journal* 3 (2015), pp. 20–23.

[139] Peter Groves et al. “The ’big data’ revolution in healthcare”. In: *McKinsey Quarterly* (2013).

[140] Rajwinder Kaur and Pawan Luthra. "Load balancing in cloud computing". In: *Proceedings of international conference on recent trends in information, telecommunication and computing, ITC*. Citeseer. 2012.

[141] Monika Lagwal and Neha Bhardwaj. "Load balancing in cloud computing using genetic algorithm". In: *2017 International Conference on Intelligent Computing and Control Systems (ICICCS)*. IEEE. 2017, pp. 560–565.

[142] Mayanka Katyal and Atul Mishra. "A Comparative Study of Load Balancing Algorithms in Cloud Computing Environment". In: *CoRR* abs/1403.6918 (2014).

[143] Supriya P Belkar and Vidya Handur. "Comparative Study of Static Load Balancing Algorithms in Distributed System Using CloudSim". In: *International Journal of Advanced Research in Basic Engineering Sciences and Technology (IJARBEST)* (2017), pp. 26–30.

[144] Asser N. Tantawi and Don Towsley. "Optimal Static Load Balancing in Distributed Computer Systems". In: *J. ACM* (), pp. 445–465.

[145] Ming Wang and Jianfeng Guan. "An adaptive dynamic feedback load balancing algorithm based on QoS in distributed file system". In: *Journal of Communications and Information Networks* (2017), pp. 30–40.

[146] Sandeep Sharma, Sarabjit Singh, and Meenakshi Sharma. "Performance analysis of load balancing algorithms". In: *World Academy of Science, Engineering and Technology* (2008), pp. 269–272.

[147] Li Zhou et al. "Optimize block-level cloud storage system with load-balance strategy". In: *Parallel and Distributed Processing Symposium Workshops & PhD Forum (IPDPSW), 2012 IEEE 26th International*. IEEE. 2012, pp. 2162–2167.

[148] Kumar Nishant et al. "Load Balancing of Nodes in Cloud Using Ant Colony Optimization". In: *2012 UKSim 14th International Conference on Computer Modelling and Simulation*. 2012, pp. 3–8.

[149] Anureet Kaur and Bikrampal Kaur. "Load balancing in tasks using honey bee behavior algorithm in cloud computing". In: *Wireless Networks and Embedded Systems (WECON), 2016 5th International Conference on*. IEEE. 2016, pp. 1–5.

[150] Sukrati Jain and Ashendra K. Saxena. "A survey of load balancing challenges in cloud environment". In: *2016 International Conference System Modeling Advancement in Research Trends (SMART)*. 2016, pp. 291–293.

[151] Ali M. Alakeel. "A Guide to dynamic Load balancing in Distributed Computer Systems". In: *International Journal of Computer Science and Network Security (IJCSNS* (2010), pp. 153–160.

[152] Neeraj Rathore. "Dynamic Threshold Based Load Balancing Algorithms". In: *Wireless Personal Communications* (2016), pp. 151–185.

[153] Hendra Rahmawan and Yudi Satria Gondokaryono. “The simulation of static load balancing algorithms”. In: *2009 International Conference on Electrical Engineering and Informatics*. 2009, pp. 640–645.

[154] P. Getzi Jeba Leelipushpam and J. Sharmila. “Live VM migration techniques in cloud environment — a survey”. In: *IEEE Conference on Information & Communication Technologies*. IEEE. 2013, pp. 408–413.

[155] Alexander Zahariev. “Google app engine”. In: *Helsinki University of Technology* (2009), pp. 1–5.

[156] Shridhar G Domanal and G Ram Mohana Reddy. “Load balancing in cloud computingusing modified throttled algorithm”. In: *2013 IEEE International Conference on Cloud Computing in Emerging Markets (CCEM)*. IEEE. 2013, pp. 1–5.

[157] Ankit Kumar and Mala Kalra. “Load balancing in cloud data center using modified active monitoring load balancer”. In: *2016 International Conference on Advances in Computing, Communication, & Automation (ICACCA)(Spring)*. IEEE. 2016, pp. 1–5.

[158] Ajay Gulati and Ranjeev K Chopra. “Dynamic round robin for load balancing in a cloud computing”. In: *IJCSMC* 2 (2013).

[159] Dr Nusrat Pasha, Amit Agarwal, and Ravi Rastogi. “Round robin approach for VM load balancing algorithm in cloud computing environment”. In: *International Journal* 4 (2014).

[160] Gamal F. Elhady and Medhat A. Tawfeek. “A comparative study into swarm intelligence algorithms for dynamic tasks scheduling in cloud computing”. In: *IEEE Seventh International Conference on Intelligent Computing and Information Systems (ICICIS)*. 2015, pp. 362–369.

[161] Simon Garnier, Jacques Gautrais, and Guy Theraulaz. “The biological principles of swarm intelligence”. In: *Swarm intelligence* 1.1 (2007), pp. 3–31.

[162] Dušan Teodorović. “Bee Colony Optimization (BCO)”. In: *Innovations in Swarm Intelligence*. Springer, 2009, pp. 39–60.

[163] Dervis Karaboga and Bahriye Akay. “A comparative study of artificial bee colony algorithm”. In: *Applied mathematics and computation* 214.1 (2009), pp. 108–132.

[164] Partha P Dutta, Nino Vidovic, and Dalibor F Vrsalovic. *System and method for network load balancing*. 2003.

[165] Mark Long. *Microsoft windows server 2008*. Virtual Training Company, Inc., 2008.

[166] Jaspreet Singh and CS Rai. “An efficient load balancing method for ad hoc networks”. In: *International Journal of Communication Systems* (2018).

[167] Krishnamurthy Bhaskar et al. *Memory load balancing*. US Patent 7,865,037. 2011.

[168] Aameek Singh, Madhukar Korupolu, and Dushmanta Mohapatra. “Server-storage virtualization: integration and load balancing in data centers”. In: *Proceedings of the 2008 ACM/IEEE conference on Supercomputing*. IEEE Press. 2008, p. 53.

[169] Wayne Karpoff and Brian Lake. *Storage virtualization system and methods.* US Patent 7,577,817. 2009.

[170] Shannon Meier. *IBM Systems Virtualization: Servers, Storage, and Software.* 2008.

[171] M. Shoaib Jameel, Muruganant Marimuthu, and Tejbanta Chingtham. "Deploying CPU Load Balancing in the Linux Cluster Using Non-Repetitive CPU Selection". In: *International Journal of Computer and Electrical Engineering* 1 (2009).

[172] Vivek Chopra, Sing Li, and Jeff Genender. *Professional apache tomcat 6.* John Wiley & Sons, 2007.

[173] Matthew L Massie, Brent N Chun, and David E Culler. "The ganglia distributed monitoring system: design, implementation, and experience". In: *Parallel Computing* 30 (2004).

[174] Bhathiya Wickremasinghe and Rajkumar Buyya. "CloudAnalyst: A CloudSim-based tool for modelling and analysis of large scale cloud computing environments". In: *DISTRIBUTED COMPUTING PROJECT, CSSE DEPT., UNIVERSITY OF MELBOURNE* (2009), pp. 433–659.

[175] E. L. Hahne. "Round-robin scheduling for max-min fairness in data networks." In: *IEEE J Sel Areas Commun.* 9.7 (1991), pp. 1024–1039.

[176] Tangang et al. "Comparative analysis and simulation of load balancing scheduling algorithm based on cloud resource." In: *Proceedings of international conference on computer science and information technology.* Springer, 2014.

[177] V. Tyagi and T. Kumar. "ORT broker policy: reduce cost and response time using throttled load balancing algorithm." In: *Proc Comput Sci.* 48 (2015), 217–221.

[178] Sudha Senthilkuma et al. "Honey-Bee Foraging Algorithm for Load Balancing in Cloud Computing Optimization". In: *IJESC* 7.12 (2017).

[179] K. Nishant. "Load Balancing of Nodes in Cloud Using Ant Colony Optimization." In: *14th International Conference on Computer Modeling and Simulation.* IEEE, 2012.

[180] R Agarwal et al. "The role of information systems in healthcare: Current research and road ahead". In: *Information Systems Research* 22 (2011), pp. 419–428.

[181] Peter Groves et al. "The 'big data' revolution in healthcare: Accelerating value and innovation". In: (2016).

[182] Divyakant Agrawal, Sudipto Das, and Amr El Abbadi. "Big data and cloud computing: current state and future opportunities". In: *Proceedings of the 14th international conference on extending database technology.* 2011, pp. 530–533.

[183] Krishna Kulkarni, Nelson Mattos, and Roberta Cochrane. "Active Database Features in SQL3". In: *Active Rules in Database Systems.* Springer New York, 1999, pp. 197–219.

[184] Jing Han et al. "Survey on NoSQL database". In: *Pervasive computing and applications (ICPCA), 2011 6th international conference on.* IEEE. 2011.

[185] Mike Mesnier, Gregory R Ganger, and Erik Riedel. “Object-based storage”. In: *IEEE Communications Magazine* 41.8 (2003), pp. 84–90.

[186] Stanley Benjamin Zdonik and David Maier. *Readings in object-oriented database systems.* Morgan Kaufmann, 1990.

[187] Zohreh Goli-Malekabadi, Morteza Sargolzaei-Javan, and Mohammad Kazem Akbari. “An effective model for store and retrieve big health data in cloud computing”. In: *Computer Methods and Programs in Biomedicine* 132 (2016).

[188] Anthony JG Hey and Anne E Trefethen. “The data deluge: An e-science perspective”. In: (2003).

[189] Yang Li, Li Guo, and Yike Guo. “An efficient and performance-aware big data storage system”. In: *International Conference on Cloud Computing and Services Science.* Springer. 2012, pp. 102–116.

[190] Bruce Momjian. *PostgreSQL: introduction and concepts.* Addison-Wesley New York, 2001.

[191] Paul DuBois. *MySQL.* Pearson Education, 2008.

[192] J Chris Anderson, Jan Lehnardt, and Noah Slater. *CouchDB: the definitive guide.* "O’Reilly Media, Inc.", 2010.

[193] Dhruba Borthakur. “HDFS architecture guide”. In: *Hadoop Apache Project* 53 (2008).

[194] M. Mesnier, G. R. Ganger, and E. Riedel. “Object-based storage”. In: *IEEE Communications Magazine* 41 (2003).

[195] Yang Li et al. “Building a cloud-based platform for personal health sensor data management”. In: *Biomedical and Health Informatics (BHI), 2014 IEEE-EMBS International Conference on.* IEEE. 2014, pp. 223–226.

[196] *Hortonworks.* http://hortonworks.com/blog/heterogeneous-storage-policies-hdp-2-2/.

[197] *UCI Cancer Dataset.* https://archive.ics.uci.edu/ml/datasets/Breast+Cancer+Wisconsin+Prognostic.

[198] *Cancer Cost Dataset.* https://costprojections.cancer.gov/.

[199] *Cancer Cost Dataset.* http://cancer.digitalslidearchive.net/.

[200] Ralf Herbrich. “Machine Learning at Amazon.” In: *WSDM.* 2017, p. 535.

[201] Ekaba Bisong. “Google Cloud Machine Learning Engine (Cloud MLE)”. In: *Building Machine Learning and Deep Learning Models on Google Cloud Platform.* Springer, 2019, pp. 545–579.

[202] Paul KJ Han, William MP Klein, and Neeraj K Arora. “Varieties of uncertainty in health care: a conceptual taxonomy”. In: *Medical Decision Making* 31.6 (2011), pp. 828–838.

[203] Andrew MacDonald. “PhilDB: The time series database with built-in change logging”. In: *PeerJ Computer Science* 2 (2016), e52.

[204] Michael Stonebraker et al. “SciDB: A Database Management System for Applications with Complex Analytics”. In: *IEEE Annals of the History of Computing* 15.03 (2013), pp. 54–62.

[205] Cory McKay et al. “ACE: A Framework for Optimizing Music Classification.” In: *ISMIR*. 2005, pp. 42–49.

[206] websense. *Advanced analysis using real-time classification*. https://www.websense.com/content/support/library/web/hosted/bsky_help/content_analysis.aspx. [Online; accessed 19-November-2019].

[207] Veritas. *Veritas Introduces New Classification Engine for Intelligent Data Management Across its Portfolio*. https://www.veritas.com/news-releases/2017-07-25-veritas-introduces-new-classification-engine-for-intelligent-data\protect\@normalcr\relax-management-across-its-portfolio. [Online; accessed 19-November-2019]. 2019.

[208] Varonis. *Varonis, DATA CLASSIFICATION ENGINE*. https://www.varonis.com/products/data-classification-engine/. [Online; accessed 19-November-2019].

[209] Elliot Sinyor et al. “Beatbox classification using ACE”. In: *Proceedings of the International Conference on Music Information Retrieval*. Citeseer. 2005.

[210] forcepoint. *Forcepoint Advanced Classification Engine (ACE)*. https://www.forcepoint.com/product/add-on/advanced-classification-engine-ace?utm_source=Websense&utm_medium=Redirect&utm_content=websense-advanced-classification-engine%3Fcmpid%3Dslblog]. [Online; accessed 19-November-2019].

[211] PSIGEN. *PSIGEN Releases Accelerated Classification Engine*. https://www.psigen.com/?s=Accelerated+Classification+Engine. [Online; accessed 19-November-2019].

[212] Gauri Shah et al. “Ace: Classification for information lifecycle management”. In: *NASA Mass Storage Systems and Technologies* (2006).

[213] Oliver Giudice et al. “A classification engine for image ballistics of social data”. In: *International Conference on Image Analysis and Processing*. Springer. 2017, pp. 625–636.

[214] Mehdi Bahrami and Mukesh Singhal. “The role of cloud computing architecture in big data”. In: *Information granularity, big data, and computational intelligence*. Springer, 2015, pp. 275–295.

[215] Eli Collins. “Big Data in the Public Cloud”. In: *IEEE Cloud Computing* 1.2 (2014), pp. 13–15.

[216] Jinting Ren et al. "Archivist: A Machine Learning Assisted Data Placement Mechanism for Hybrid Storage Systems". In: *2019 IEEE 37th International Conference on Computer Design (ICCD)*. IEEE. 2019, pp. 676–679.

[217] Ridwan Rashid Noel, Rohit Mehra, and Palden Lama. "Towards self-managing cloud storage with reinforcement learning". In: *2019 IEEE International Conference on Cloud Engineering (IC2E)*. IEEE. 2019, pp. 34–44.

[218] Gartner. *AIOps (Artificial Intelligence for IT Operations)*. https://www.gartner.com/en/information-technology/glossary/aiops-artificial-intelligence-operations. [Online; accessed 29-June-2020]. 2020.

[219] Anna Levin et al. "AIOps for a Cloud Object Storage Service". In: *2019 IEEE International Congress on Big Data (BigDataCongress)*. IEEE. 2019, pp. 165–169.

[220] IBM. *IBM Cloud Object Storage*. https://www.ibm.com/cloud/object-storage. [Online; accessed 29-June-2020]. 2020.

[221] Ling-Fang Zeng, Dan Feng, and Ling jun Qin. "SOSS: smart object-based storage system". In: *Proceedings of 2004 International Conference on Machine Learning and Cybernetics (IEEE Cat. No. 04EX826)*. Vol. 5. IEEE. 2004, pp. 3263–3266.

[222] Karamjit Kaur and Rinkle Rani. "Managing data in healthcare information systems: many models, one solution". In: *Computer* 48.3 (2015), pp. 52–59.

[223] Kishan Trivedi, Sambhav Shah, and Kriti Srivastava. "An Efficient E-Commerce Design by Implementing a Novel Data Mapper for Polyglot Persistence". In: *Advanced Computing Technologies and Applications*. Springer, 2020, pp. 149–156.

[224] Michael Schaarschmidt, Felix Gessert, and Norbert Ritter. "Towards automated polyglot persistence". In: *Datenbanksysteme für Business, Technologie und Web (BTW 2015)* (2015).

[225] Anindita Sarkar, Kshitij Pant, and Samiran Chattopadhyay. "DRSQ-A Dynamic Resource Service Quality Based Load Balancing Algorithm". In: *International Conference on Computational Intelligence, Communications, and Business Analytics*. Springer. 2018, pp. 97–108.

[226] Ling-Fang Zeng et al. "Object replication and migration policy based on OSS". In: *2005 International Conference on Machine Learning and Cybernetics*. Vol. 1. IEEE. 2005, pp. 45–49.

[227] Jason Weston et al. "Feature selection for SVMs". In: *Advances in neural information processing systems*. 2001, pp. 668–674.

[228] Yi-Wei Chen and Chih-Jen Lin. "Combining SVMs with various feature selection strategies". In: *Feature extraction*. Springer, 2006, pp. 315–324.

[229] Jun Liu, Shuiwang Ji, and Jieping Ye. "Multi-task feature learning via efficient l 2, 1-norm minimization". In: *Proceedings of the twenty-fifth conference on uncertainty in artificial intelligence*. AUAI Press. 2009, pp. 339–348.

[230] Padraig Cunningham and Sarah Jane Delany. "k-Nearest neighbour classifiers". In: *Multiple Classifier Systems* 34.8 (2007), pp. 1–17.

[231] Bernhard Scholkopf and Alexander J Smola. *Learning with kernels: support vector machines, regularization, optimization, and beyond.* MIT press, 2001.

[232] Donald F Specht. "A general regression neural network". In: *IEEE transactions on neural networks* 2.6 (1991), pp. 568–576.

[233] Marina Sokolova and Guy Lapalme. "A systematic analysis of performance measures for classification tasks". In: *Information processing & management* 45.4 (2009), pp. 427–437.

[234] Nathalie Japkowicz. "Why question machine learning evaluation methods". In: *AAAI workshop on evaluation methods for machine learning*. 2006, pp. 6–11.