High Availability Ultra-Large Cloud-Oriented Storage Systems: A Review, Mind Mapping and Open Research Directions

AJAY KUMAR, University of Petroleum and Energy Studies, Dehradun, India
SEEMA BAWA, Thapar University, Patiala, India
NEERAJ KUMAR, Thapar University, Patiala, India
RAJKUMAR BUYYA, Cloud Computing and Distributed Systems (CLOUDS) Laboratory, Department of Computing and Information Systems, University of Melbourne, Melbourne, Australia
ABDULLAH MOHAMMED ALMUHAIDEB, Imam Abdulrahman Bin Faisal University, Dammam, Saudi Arabia

Cloud computing paradigm enables the delivery of computing, storage, and applications as services as subscription-oriented services to end users over the Internet. It reduces the need for highly expensive computing infrastructure, dedicated storage servers, and software applications to be procured, deployed, and managed as in house capabilities. The amount of data generated by enterprise and business applications has increased exponentially. Addressing and dealing with enormous datasets is a challenging and very time-consuming task that necessitates a high level of computing infrastructure to enable proper data processing and analysis. This study aims to compare and analyze the different aspects of ultra-large data storage systems in cloud computing. The description, features, and classification of cloud storage systems are discussed, along with certain cloud computing considerations. Relationships between big data and cloud computing are also explored, as are ultra-large storage systems and Hadoop technology. Furthermore, research concerns such as dynamicity, scalability, availability, data integrity, self-organizing, data quality, data diversity, privacy, and legal and regulatory issues have been investigated. The sole motivation of this study has been thoroughly explained with the help of a mind-mapping diagram of cloud-oriented data storage (CODS) elements. The seven major elements of CODS—the data model, data consistency, data replication, data scalability, data integration, data virtualization, and transaction—have been systematically discussed in this study. This study also highlights some open research issues related to ultra-large storage systems, which summarize sufficient research efforts.

CCS Concepts: • **Cloud Computing → Cloud-Oriented Storage Systems**; *Ultra-Large Storage Systems*; High Availability Storage System; • **Networks** → Network reliability.

Additional Key Words and Phrases: cloud computing, cloud storage, hadoop, rdbms, nosql

1 Introduction

Cloud computing is an emerging computing paradigm that comprises six distinct phases. These phases represent the transition from early mainframes to cloud computing, as recognized and depicted in Fig. 1. During Phase 1, users connected to mainframes shared by several users via their dumb terminals. In phase 2, these dumb terminals were replaced with powerful PCs, eliminating the need to share mainframes. Phase 3 established the basis for computer networks to connect many computers, allowing users to share resources across local networks. Phase 4

Authors' Contact Information: Ajay Kumar, University of Petroleum and Energy Studies, Dehradun, UK, India; e-mail: ajaycpp@gmail.com; Seema Bawa, Thapar University, Patiala, Punjab, India; e-mail: seema@thapar.edu; Neeraj Kumar, Thapar University, Patiala, Punjab, India; e-mail: neeraj.kumar@thapar.edu; Rajkumar Buyya, Cloud Computing and Distributed Systems (CLOUDS) Laboratory, Department of Computing and Information Systems, University of Melbourne, Melbourne, Victoria, Australia; e-mail: rbuyya@unimelb.edu.au; Abdullah Mohammed Almuhaideb, Imam Abdulrahman Bin Faisal University, Dammam, Eastern, Saudi Arabia; e-mail: amAlmuhaideb@iau.edu.sa.

ACM 1557-7341/2026/7-ART
https://doi.org/10.1145/3831665

employs the Internet to build a global network that connects several local networks. Phase 5 introduced the notion of grid computing, which allows users to transparently share computer power and resources. Cloud computing is the era of Phase 6. There are many similarities between cloud computing and grid computing, such as scalability,

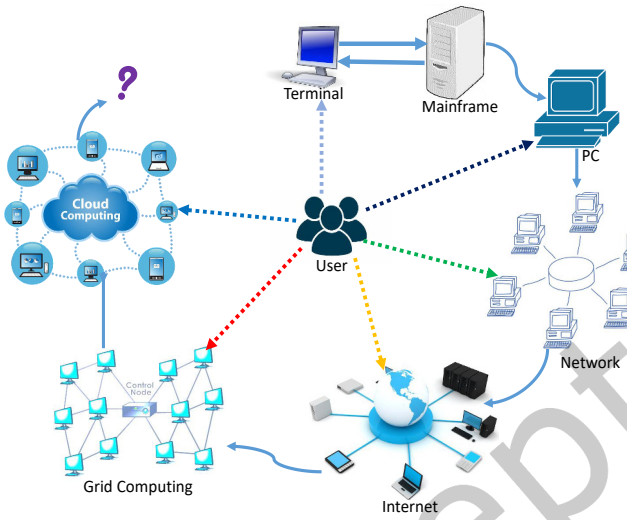

Fig. (1) Evolution of computing paradigms towards cloud computing

adaptability, security, agility, and serviceability, but the size of cloud computing necessitates a distinct application architecture. This new approach is essential if businesses are to embrace cloud computing as a new paradigm for computing. The Internet is used to deliver on-demand, virtualized, and scalable resources known as cloud computing. In a cloud environment, all resources can be considered as services including Infrastructure as a Service (IasS), Platform as a Service (PaaS), Software as a Service (SaaS), Data as a Service (DaaS), Communication as a Service (CaaS), Database as a Service (DBaaS), Security as a Service (SECaaS), etc. The major research contributions of this review are listed below:

- Highlight the cloud-oriented storage management goals and challenges.
- Mind mapping of Cloud-Oriented Data Storage (CODS) elements: *data model, data consistency, data replication, data scalability, data integration, data virtualization, data transaction.*
- Providing a mind mapping diagram for data model in five aspects: *access level, data source, content-based, data staging and abstraction level.*
- Explanation of data consistency mapping brunches like *consistency level, consistency model, consistency type, consistency measures, consistency protocols and consistency maintenance.*
- Elaboration on data replication, integration, scalability and virtualization with the help of individual mind mapping diagrams.
- Mind mapping of data transaction *type, isolation, correctness and dependency* and analyzing the performance issues based on resulted mind mapping.
- Highlights the features and capability of existing cloud-oriented data storage systems and comparison between them all.

The rest of the paper is organized as follows. Section 2 highlights the cloud-oriented data storage goals, challenges and elements mind mapping. Section 3-8 describes the mind mapping of *data model, data consistency, data replication, data scalability, data integration, data virtualization, data transaction* respectively. Section 9

highlights the open research issues and future research directions.Finally this review is ended with some concluding remarks.

2 Systematic Review Technique

This study used the PRISMA guidelines and AMSTAR checklists [86] in order to access the methodological quality of the systematic review technique. Five different phases, including *identification, screening, eligibility, inclusion, and qualitative synthesis* have been selected in order to complete this study. Identification is the first phase of this systematic review technique. Initially, 837 research articles have been shortlisted from different sources, including journal databases, magazines, book chapters, web reports, etc. Around 87% titles have been accessed from well-known digital libraries, as listed below. The rest of the additional records have been identified from other sources, including books, web-based blogs, search engines, expert advice, etc.

- Springer (https://link.springer.com/)
- ACM (https://dl.acm.org/journals)
- IEEE eXplore (https://ieeexplore.ieee.org/)
- Taylor & Francis Online (https://www.tandfonline.com/)
- Google Scholar (https://scholar.google.com/)
- ResearchGate (https://www.researchgate.net/)
- Elsevier (https://www.sciencedirect.com/)

In the screening phase, all the duplicate articles have been filtered from the title database. Furthermore, an overall screening process has been conducted based on research titles and some of the most frequent keywords, including "cloud computing," "ultra-large data," "big data," and "cloud storage databases." In this phase, 578 research titles have been selected for the next phase. In the screening phase, around 25% of the articles have been eliminated based on duplicates and paper tiles. Eligibility is the very crucial phase in which overall shortlisting is done on the basis of the abstract, conclusion, and objective of each article. Here, 254 titles were found eligible for further investigation. In the fourth phase, 126 titles have been included on the basis of full-text assessment. The "Synthesis Analysis" is the last phase of this survey technique, and finally 54 titles are synthesized and qualitatively analyzed based on the common challenges and references. All these titles have been critically reviewed and studied for this study. Moreover, about 57% of papers are taken in the years between 2018 and 2022, and 15% of papers are from the current year.

Table 1 presents a comparative qualitative analysis of existing survey studies on cloud-oriented storage systems, focusing on seven core architectural and operational aspects: data model, data consistency, data replication, data scalability, data integration, data virtualization, and data transaction support. Each survey is evaluated from two complementary perspectives: Mind Mapping View (MMV), which reflects conceptual and architectural coverage, and Research Conclusion (RC), which indicates the extent to which conclusions and insights are analytically synthesized.

Several surveys emphasize foundational storage paradigms such as relational, NoSQL, and object-based models. Early works, such as surveys in [50] and [73], partially discuss data models with limited architectural depth. In contrast, [66], [68], and [40] provide more comprehensive treatment, particularly highlighting schema-less and multi-model storage approaches. However, the discussion in many surveys remains fragmented, lacking an integrated perspective across heterogeneous cloud storage environments. Consistency models (e.g., strong, eventual, and causal consistency) are unevenly addressed across prior surveys. Studies like [15] and [66] offer relatively strong conceptual explanations but provide limited conclusions regarding real-world trade-offs. Surveys such as [88] and [35] only partially cover consistency mechanisms, often focusing on specific platforms rather than generalized cloud storage architectures. Replication strategies for fault tolerance and availability are discussed more consistently across the literature. Surveys in [73], [89], and [68] provide strong analytical conclusions on

Table (1) Related surveys on cloud-oriented storage system

Survey	Data Model		Data Consistency		Data Replication		Data Scalability		Data Integration		Data Virtualization		Data Transaction	
	MMV	RC	MMV	RC	MMV	RC	MMV	RC	MMV	RC	MMV	RC	MMV	RC
[50]	○	◖	○	○	○	○	○	○	○	○	○	◖	○	○
[73]	○	◖	○	○	●	●	○	○	○	○	○	◖	○	○
[15]	○	◖	●	●	○	○	○	○	○	○	○	○	◖	◖
[88]	●	●	○	◖	○	◖	○	○	○	○	○	○	○	○
[35]	○	◖	○	◖	○	○	○	○	○	○	○	○	○	○
[66]	●	●	●	●	◖	◖	◖	◖	◖	◖	○	◖	●	●
[65]	◖	◖	○	◖	○	○	◖	●	○	◖	◖	◖	○	○
[38]	○	◖	○	◖	○	◖	○	◖	○	○	○	○	○	◖
[40]	●	●	◖	◖	○	○	◖	◖	○	○	○	○	○	○
[89]	◖	◖	○	○	●	●	◖	◖	○	○	○	○	○	○
[68]	●	●	◖	◖	◖	●	●	◖	◖	◖	○	◖	○	◖
[37]	○	◖	◖	◖	○	○	○	○	○	○	○	◖	○	○
This survey	●	●	●	●	●	●	●	●	●	●	●	●	●	●

MMV : Mind Mapping View, RC : Research Conclusion, ●: Full cover, ◖: Partial cover, ○: Does not cover

replication techniques, while works such as [50] and [35] mainly introduce replication at a conceptual level. Nevertheless, cross-layer replication challenges (storage, network, and application layers) are insufficiently synthesized in most surveys. Scalability is a central theme in cloud storage research, yet its coverage varies significantly. Surveys such as [66], [68], and [40] provide strong MMV and RC coverage by addressing horizontal scaling, elastic storage, and performance bottlenecks. Conversely, earlier surveys like [50] and [73] treat scalability only superficially, without rigorous performance or architectural analysis. Data integration across distributed, heterogeneous, and multi-cloud environments is one of the least explored aspects. Most surveys, including [50], [73], [35], and [38], either ignore this dimension or cover it marginally. Only a few studies, such as [66] and [68], partially address integration challenges, primarily from a middleware or interoperability standpoint. Virtualization techniques enabling abstraction, location transparency, and resource pooling are inconsistently discussed. Surveys like [66], [65], and [68] demonstrate stronger coverage, particularly in relation to virtual storage pools and software-defined storage. However, many surveys, including [50] and [42], provide minimal or no analytical conclusions on virtualization impacts. Transactional support (e.g., ACID, BASE, and hybrid models) receives the least attention overall. While [15] and [66] partially explore transactional mechanisms in distributed storage systems, most surveys such as [50], [35], and [38] do not adequately address transaction management, especially in large-scale, geo-distributed cloud settings.

As indicated in the final row, this survey provides full coverage across all seven aspects in both MMV and RC dimensions. Unlike prior works, it offers a holistic and integrated analysis that connects architectural foundations with operational challenges and research conclusions. This comprehensive scope addresses critical gaps identified in earlier surveys, particularly in data integration, virtualization, and transaction management, thereby advancing the state of knowledge in cloud-oriented storage systems.

3 Background

Cloud computing has emerged as a pervasive platform which provides Internet computing services. Cloud-Oriented Storage as a Service (COSaaS), a cloud storage service model is one of the significant emerging field in which user can get access storage space in the form of Virtual Machines (VMs). Efficient management of ultra-large storage is a challenging task that aims to optimize available resources and enhance the performance of COSaaS. A cloud-based ultra-large storage system refers to the huge and scalable infrastructure that cloud service providers offer to meet the growing demands of businesses and individuals for data storage. These systems are designed to store and manage massive amounts of data efficiently, securely, and cost-effectively. In the big data era, enterprises and organizations are moving toward cloud environments for scalable and effective storage solutions due to the increase in demand for ultra-large data storage solutions. Cloud storage is the best approach

for businesses handling enormous datasets since it provides an adaptable, affordable, and scalable method of managing enormous volumes of data. Comparative analysis of various cloud-oriented storage system's features, mediums and their capabilities have also been highlighted in this section. At the end, this section concludes with research issues and cloud-oriented storage taxonomy.

3.1 Ultra-Large Cloud-Oriented Storage Systems

Cloud storage is a cloud computing model in which data is stored on remote servers accessed from the Internet. It is maintained, operated and managed by a cloud storage service provider on a storage servers that are built on virtualization techniques. The salient features of all 21 well-known cloud-oriented storage systems are shown in Table 9 (Appendix A). This table provides a comprehensive comparison of various cloud-oriented storage systems, focusing on their key characteristics and capabilities. The table is divided into several columns representing seven different storage capabilities including self-organizing, scalability, high Availability, Replication, Durability, Consistency, and Storage Types have been critically reviewed. The table illustrates the diversity of cloud storage systems in terms of their capabilities and intended use cases. The details information of cloud storage capabilities are listed below:

i **Self-organizing:** This aspect involves the system's ability to automatically organize and manage data without manual intervention. Self-organizing systems can optimize data placement, distribution, and retrieval. This feature is particularly beneficial in cloud environments where data volumes can be large and constantly changing. Self-organizing describes the system's capacity to arrange its parts in the organizational structure of the task without requiring outside assistance. System components can be changed by self-organizing systems without the assistance of humans. The following overlay structures have been used to categorize self-organizing systems:

- **Fully decentralized:** In a fully decentralized overlay, all nodes in the cloud storage system are equal and have the same responsibilities and capabilities. There is no central authority or coordination point; instead, each node independently manages its data and communicates directly with other nodes to store and retrieve data. It is highly resilient to failures, as there is no single point of failure [7].
- **Partially centralized:** A partially centralized overlay incorporates elements of both centralization and decentralization. In this structure, some nodes act as coordinators or supernodes, managing specific functions like routing, indexing, or resource allocation, while other nodes operate more independently. It balances scalability and efficiency by reducing the overhead of fully decentralized systems while avoiding the single points of failure found in fully centralized systems[85].
- **Mess overlay:** In a mesh overlay, each node in the network is connected to several other nodes, forming a mesh-like structure. Nodes can communicate directly with their neighbors and indirectly with more distant nodes through multiple hops. This overlay does not rely on a central authority, and nodes can dynamically discover and maintain connections with other nodes. It offers high fault tolerance, as multiple paths exist between any two nodes, ensuring that data can be routed around failures [56].
- **Hybrid overlay:** It combines features from different overlay types, tailoring the structure to the specific needs of the system. It offers flexibility, allowing the system to leverage the strengths of different overlay types while mitigating their weaknesses. For instance, it can achieve efficient data retrieval and management through centralized components while maintaining fault tolerance and scalability through decentralized elements [28].
- **Tree overlay:** In a tree overlay, nodes are organized into a hierarchical tree structure, with each node having a parent node (except the root) and potentially multiple child nodes. This structure is often used for multicast or broadcast operations, where data needs to be distributed from a source node to many receivers.

It is efficient for distributing data in a controlled manner, as each node forwards data to its children, reducing redundant transmissions [55].

All these overlay structure have their own set of rules to perform different operations[4][75]. Some well-known self-organizing storage models are summarized in Table 10 (Appendix B).

ii **Scalability:** Scalability is the system's ability to handle an increasing amount of data and workload. A scalable storage system can grow or shrink based on demand without compromising performance. This is especially important in cloud environments where data volumes can fluctuate greatly. By providing scalability, storage solutions can effectively handle the increasing demands of data storage and processing, ensuring that system performance remains optimal even as the workload grows. Furthermore, a scalable storage system allows for cost-effective resource allocation, as organizations can easily adjust their storage capacity to match their current needs without unnecessary expenses.

iii **High Availability:** High availability means that the storage system is reliably and consistently accessible. It ensures that the data is always available, minimizing downtime and service interruptions. In addition, high availability also provides fault tolerance, as it allows for redundancy and data replication across multiple storage devices. This means that even if one device fails, the data can still be accessed from another device, ensuring continuous operation and preventing data loss.

iv **Replication:** Replication involves creating and maintaining duplicate copies of data across multiple locations or servers. This is done to enhance data availability, fault tolerance, and disaster recovery. In replication, the duplicate copies of data are synchronized in real-time or at regular intervals to ensure consistency and integrity. This allows for seamless failover and load balancing, as requests can be redirected to other servers with replicated data. Additionally, replication also facilitates efficient data distribution and scalability, enabling organizations to handle increasing workloads and accommodate growing user demands.

v **Durability:** Durability measures the enduring nature of data over time. A durable storage system ensures that data is not lost or corrupted, even in the face of hardware failures or other issues. It can be achieved through techniques such as data backups, redundancy, and error detection and correction mechanisms. By ensuring the durability of data, organizations can have confidence in the long-term availability and reliability of their stored information. This is particularly important for critical systems and applications where data loss or corruption could have severe consequences.

vi **Consistency:** Consistency refers to the uniformity of data across the storage system. In a consistent system, all nodes or replicas of data provide the same view to clients, ensuring that users see the latest and correct information. A consistent system is crucial for applications where data loss or corruption could have severe consequences.

vii **Storage Types:** This includes the different types of storage solutions offered by each system, such as object storage, file storage, block storage, etc. Each type serves specific use cases and has its own characteristics. Each type serves specific use cases and has its own characteristics, allowing clients to choose the most suitable storage solution for their needs. The comparative analysis of various storage mediums and their objectives, advantages, and disadvantages are highlighted in Table 11 Appendix C.

3.2 Ultra-Large Storage Management Capabilities

The ultra-large storage management in cloud computing refers to the ability to store, manage, and scale huge amounts of data in distributed systems. Mainly, it focuses on scalability, elasticity, performance, and cost-efficiency while ensuring data security, integrity, and regulatory compliance. These systems leverage distributed architectures and advanced data management techniques to handle massive amounts of data in a flexible and resilient manner. The Cloudera Manager, NoSQL databases, and workflow management tools are key components in a cloud storage and big data processing ecosystem. They serve distinct purposes but work together to provide

Table (2) Cloud-Oriented Storage Integration Tools

Tools	Service Category	Service Model	Objectives	Ref.
CloudFuice	SaaS	Web engineering model and mashup development	Task-based execution approach that allows parallel execution of of data-flows in the cloud	[92]
iProX	IaaS	Hyper-converged model	To enable the computing, storage requirements using unified and fully-virtualized resources pool. To achieve high scalability	[64]
SnapLogic	PaaS	HR, CRM	Provide both data integration as well as application multi-point connection strategies. Handle a more multifarious set of use-cases	[76]
WorkDay	SaaS	Legacy application of HR, CRM	To provide pre-built connectors, intuitive user interface and reporting structure	[47]
Adeptia	SaaS	B2B, SOA, Social Network	To provides application and data integration services that include connections, data mapping, conversion and extractions	[55]
Scribe	PaaS	Mobile application model	To improve application life span. To increase the cloud adaption in the market segment.	[62]
Nginx	PaaS	Microservice model	Integrate API broker and API getways and provides microservice users	[58]

a comprehensive solution for managing, storing, and processing large volumes of data. Cloudera Manager manages and monitors the distributed storage ecosystem, including HDFS. It provides a centralized interface for administrators to configure and maintain HDFS clusters [33] . In a cloud storage and big data ecosystem, NoSQL databases might be used alongside HDFS to accommodate different data storage and retrieval requirements [69]. Workflow management tools can integrate with HDFS to schedule and coordinate data processing tasks and workflows. They can also interact with Cloudera Manager to monitor the health and performance of Hadoop clusters. Additionally, workflow management tools may coordinate tasks involving NoSQL databases within the broader data processing pipeline. There are many challenges faced by large-scale data analysis, including data transfer, integrating data, managing data, controlling data, standardizing data, etc[83] [19]. Some new solutions are readily available to address all kinds of challenges of this nature. Such a solution includes their large-scale data management capabilities.

3.3 Cloud-Oriented Storage Management and Integration Services

Cloud computing services have gained a place in the enterprise based on their low cost and unlimited virtual scaled data to meet the user's needs. The cloud application integration is dependent on synchronization, data migration, quality, and replication. Most of the cloud integration tools are based on SaaS providers. Some major state-of-the-art cloud data integration tools are summarized in Table 2. Each tool in the table aligns with a specific service model that dictates its architecture and the level of control it offers to users. Tools like Nginx and Scribe are highly specialized, focusing on microservices and mobile application management, respectively. In contrast, tools like Adeptia and SnapLogic offer more generalized integration services, supporting a wide array of business processes. Integration of cloud applications is dependent on synchronization, data migration, quality, and replication. Most of the cloud integration tools are based on SaaS providers. The overall framework for ultra-large storage and integration management is illustrated in Fig. 2.

3.4 Cloud Storage Management: Goals and Challenges

This section highlights some major goals and challenges for deploying the ultra-large cloud-oriented storage systems. In general, ultra-large cloud data storage systems face specific goals and challenges due to their scale and complexity. All these challenges have been identified from well qualified sources including research articles, books, magazine, etc.

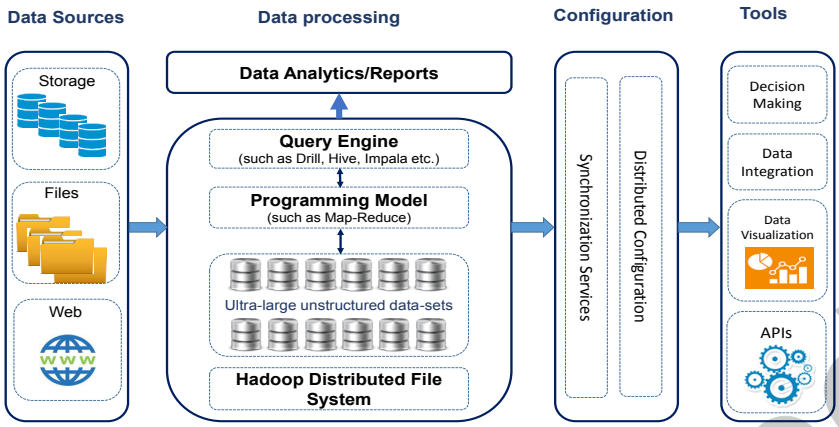

Fig. (2) Ultra-large data storage and integration management

Goals: Cloud-oriented data storage systems are designed to satisfy the following requirements:

i **Data availability** is the ability to make sure that important data is still available in an organization's IT infrastructure even if there is a problem. It emphasizes ensuring that the stored data is reliably and consistently accessible for retrieval and processing. Achieving high data availability involves addressing issues such as data replication, distribution, and fault tolerance.

ii **Elasticity** defines the degree to which a system is capable of making provisions for changes in workload and favoring the resources by de-provisioning, such that the available resources at all times match the current demand as closely as possible.

iii **Multi-tenancy** is a cloud computing architecture that enables customers to share computing resources in either the public or private cloud. Each tenant's data is separated and is invisible to other tenants.

iv **Load balancing** enables the organisation to manage workload demands or application needs by distributing resources on different networks or servers.

v **Fault-tolerance** provides the continuity of service after the failure of one system or any part of a system.

vi **Flexibility** increases the level of customization and change as needed at any time.

vii **High data confidentiality** refers to protecting information from being accessed by unauthorised users.

viii **Scalability** refers to the ability to handle increased loads.

ix **High availability** refers to dynamic resource provisioning as per user needs. It focuses on minimizing downtime and ensuring that the entire system remains operational and accessible, not just the data. Achieving high availability involves addressing various components of the system, including hardware, software, networking, and infrastructure.

x **High durability** confines the long-life survival of the system.

Challenges: Deploying ultra-large cloud-oriented data storage systems are not trivial or straightforward task. Some major challenges are listed below:

i **Data Confidentiality:** Cloud storage services typically involve storing data on remote servers owned and maintained by third-party providers. Some key aspects and measures to consider for ensuring data confidentiality in a cloud storage system include encryption, access controls, authentication, audit logging, security audits, data loss prevention etc.

ii **Data Lock-in:** It refers to a situation in which a user or organization becomes heavily dependent on a particular cloud service provider, making it challenging to migrate data to another provider or back to an on-premises solution.

iii **Data Transfer Bottlenecks:** Data transfer bottlenecks in cloud storage systems can impact performance and user experience. Identifying and addressing these bottlenecks is crucial for optimizing data transfer speed and efficiency.

iv **Application Parallelism:** Parallelism in cloud data storage refers to the ability to perform multiple operations simultaneously, which can significantly improve performance and scalability. However, there are several challenges associated with achieving effective application parallelism as shown in Table 13 (Appendix E). This Table outlines the key challenges associated with achieving application parallelism in cloud storage systems and suggests mitigation strategies to address these challenges. The table is organized by different parallelism factors, which are essential for optimizing the performance, reliability, and scalability of cloud storage systems.

v **Application Debugging:** Debugging an ultra-large distributed storage system can be a complex and challenging task due to its scale, distributed nature, and potential for various types of issues. Some well-known strategies and best practices for debugging applications in such systems are: distributed tracing, profiling, fault injection, static analysis, etc. These strategies and best practices help developers identify and isolate issues in different components of the system, such as network latency, data corruption, or resource contention [104].

vi **Strong Consistency:** Strong consistency in the context of cloud storage systems is a level of consistency where every node in the system always sees the data from the same perspective. In other words, all read operations performed after a write operation will return the most recent written value. Strong consistency must be achieved for applications where data integrity and accuracy are critical. In distributed systems, consistency models vary, with high consistency being the most stringent. It can be difficult to achieve, because distributed settings have inherent problems, including high availability requirements, node failures, and network delays [14].

vii **Interoperability issue:** This issue arises in cloud storage systems when there are difficulties in seamlessly exchanging data and services between different cloud platforms, applications, or systems. These issues can hinder the flexibility, efficiency, and collaboration potential that users expect from cloud environments [12].

viii **Data integrity issues:** Ensuring data integrity in ultra-large cloud storage systems is a critical challenge due to the scale, complexity, and distributed nature of such environments. Data integrity refers to the accuracy, consistency, and reliability of data stored in the cloud.

Successfully addressing these goals and overcoming these challenges requires a combination of advanced technologies, architectural design considerations, and continuous optimization efforts in ultra-large cloud data storage systems.

3.5 Cloud-Oriented Data Storage Elements

This section highlights the overall cloud-oriented data storage elements alongside their mind map as shown in Fig. 3. Mind mapping of the CODS elements including data model, data consistency, data replication, data scalability, data integration, data virtualization, and data transaction. Each of these elements is crucial for the effective management of ultra-large data storage systems in a cloud environment. The strengths of these elements highlight the benefits of using cloud storage for scalable, reliable, and efficient data management. However, their weaknesses underscore the complexities and challenges that need to be addressed when designing and implementing a cloud storage solution. For better readability, comparative and insight analysis of these elements are explained in Table 12 (Appendix D) and Appendix G respectively. Mainly, this table highlights the strengths and weaknesses of each CODS element. All these elements have been systematically explained in subsequent sections. Cloud computing provides an infinite pool of scalable, reliable, and flexible computing services, storage spaces, and network resources on a payment basis as per user needs. In general, these services are categorized

under IaaS, where Cloud-Oriented Storage as a Service (COSaaS) is one of its most critical parts. COSaaS provides a diverse range of data storage on a global scale.

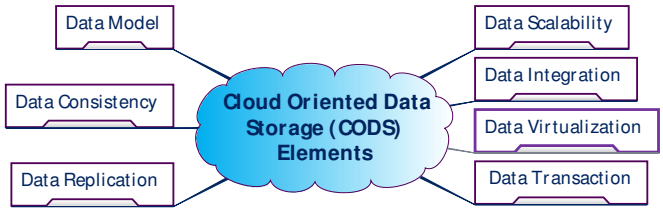

Fig. (3) Mind Mapping of CODS Elements

4 Data Model

The data model in a cloud storage system plays a very critical role in determining how data is structured, stored, accessed, and managed across distributed environments. In cloud storage, the data model needs to support various use-cases, from simple file storage to complex multi-tenant applications. The main key aspects of the data model include its ability to handle unstructured, semi-structured, and structured data efficiently, as well as its support for operations like querying, indexing, and data consistency. A well-designed data model in cloud storage should also accommodate scalability, ensuring that as the volume of data grows, the performance and reliability of the system are maintained. This discussion would examine the trade-offs involved in choosing between different data models, such as key-value stores, object stores, and relational databases, each offering unique benefits and challenges in terms of performance, scalability, and flexibility. A data model shows how information is logically arranged, kept in storage, accessed, and changed there. The significance of data modeling, which is sometimes disregarded as a component of cloud computing, should increase as cloud computing obtains traction in the context of data warehousing and business intelligence. This section describes the various components of the data model and their comprehensive mind maps, which are displayed in Fig. 4.

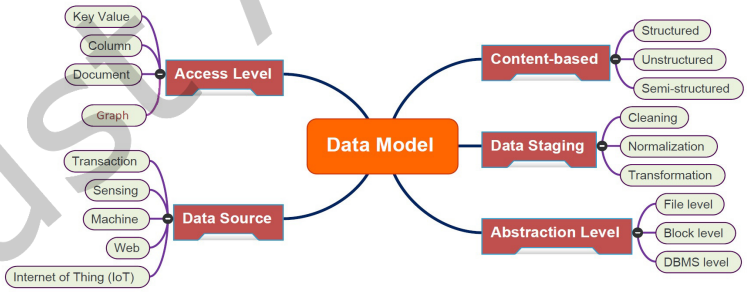

Fig. (4) Data Model Mind Mapping

4.1 Access Level

In general, cloud databases do not employ the relational model because they are associated with several operational models. Currently, five major access levels are available in cloud storage systems, comprising key-value, column-based, document [93], and Graph-based [38][71]. The various use-cases and some major examples of each access level are highlighted in Table 3. It provides a comprehensive overview of various data access levels, their associated

use-cases, and examples of databases that support these access levels. This table categorizes databases into five main access levels: column-oriented, key-value, document-oriented, and graph-based. Each level has tailored to different data management requirements. The table highlights how different database architectures are optimized for specific use-cases, reflecting the importance of selecting the right database model based on application requirements.

4.1.1 Key-value. Key-value databases work by managing and storing associative arrays. These arrays are based on dictionaries and hash tables, which consist of various key-value pairs. Here, the key serves as a unique identifier in order to fetch the associated data. These data may be simple objects, complex objects, or JSON structures. Key-value databases store data in the repository without any structure or relationship. It treats any data as an opaque blob. It is also dependent on the application and how it is structured. The key in a key-value pair should be unique enough to allow retrieval of the associated data. Kay may be either an integer key (iKey) or a string key (sKey). The hashing approach can sometimes be used to generate iKey. The pair is a combination of key and value, while the bucket contains all such pairs. The value can be saved as images, text, markup text (such as HTML or XML code), programme codes, numbers, files, audio, videos, and so on. Some key-value databases are also allowed to store lists, sets, and arrays of associated values.

4.1.2 Column-oriented. The column-oriented database stores data in columns instead of rows. It accesses more precisely data access rather than discarding or scanning data in rows. Data has been inserted into the table and internally it assigns a unique identifier, which is known as row_id. This row_id is totally independent from user-assigned unique ids such as customer_id, employee_id, etc. It serializes all the column values together and similarly for all the next columns. It could be about the efficiency of storage device access on a given workload. It is more suitable for highly complex queries over large datasets.

4.1.3 Document-oriented. Document-oriented is a widely used cloud database for storing, retrieving, and managing information. It is also known as a semi-structured database. Additionally, it makes use of the XML attributes and is a sub-type of a key-value database. The internal organization of the document is used by the document-oriented system to extract metadata. Generally, document-oriented databases are used for content management, real-time data analytics, resource management, business analytics, and so on.

4.1.4 Graph-based. Graph-based databases are closer to document-based databases. It stores data in documents and does not follow that data as a predefined schema. It adds another layer to a document-based database by emphasizing the relationship over individual documents. Major use-cases include fraud detection, recommendation systems, context-aware services, etc. Some important components of a graph-based database are pointed out below:

i **Node:** It represents the individual entity. It is based on relational model.
ii **Edge:** Establish the relationship among nodes. Mainly, edge can be either *un-directed* or *directed*.
iii **Property:** It specifies the actual information related to particular node.

4.2 Data Source

In an ultra-large cloud storage system, data models are designed to handle diverse data types and sources efficiently. These data models help organize, structure, and manage data within the storage infrastructure. The data sources for these models can vary widely, reflecting the diverse nature of data in such systems. The different sources of data and their detail information are tabulated in Table 14 (Appendix F).

Table (3) Use-cases of different data access levels

Access Level	Use-cases	Database Examples
Column-oriented	Data-ware housing, Transaction Processing, Business Intelligence Analysis, Self Indexing, Data Compression, etc.	Apache Kudu, HBase, MariaDB, MonetDB, GreenplumDB, ClickHouse, CrateDB, etc.
Key-value	Message queuing, caching, session management etc.	Redis, Dynamo, Aerospike, Berkeley DB, Cassandra, ArangoDB, Voldemort, Keyspace, Hibari, Venti, Oracle NoSQL Database, etc.
Document-oriented	Content Management, Real Time data analytic, Resource Management, Business Analytics, etc.	ArangoDB, CouchDB, Cloudant, DocumentDB, MarkLogic, MongoDB, Cosmos DB, Qizx, Sedna, SimpleDB, Solr, Elasticsearch, BaseX, etc.
Graph-based	Recommendation engine for E-Business, Symbolic AI, Context-Aware Services, Fraud Detection, Semantic Search, etc.	Neo4J, ArangoDB, Amazon Neptune, DGraph, MarkLogic, OrientDB, RedisGraph, TigerGraph, TerminusDB, etc.

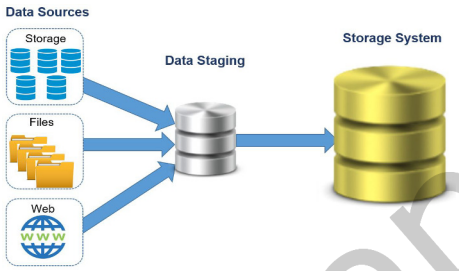

Fig. (5) Data Staging

4.3 Content-Based

Structured, semistructured, and unstructured content refer to different types of data organization and representation, each with its own characteristics and implications. These terms are commonly used in the context of databases, information retrieval, and data management to describe the organization and representation of data. Structured content refers to data that is organized in a highly organized and predefined manner. It is characterized by a fixed schema or data model where information is organized into well-defined fields. There is a predefined structure with specific data types for each field, and data is often organized in tables with rows and columns. Traditional relational databases like MySQL, PostgreSQL, and Oracle store are examples of structured data. In contrast, unstructured data refers to information that does not have a predefined structure or format. It can include text documents, images, videos, social media posts, and more.

4.4 Data Staging

Data staging is a type of landing space where data can be copied before being sent to a storage system. It acts as an interface between external sources of data and the storage system, as mentioned in Fig. 5. All such operations like insert, delete, and update have been done quickly. While copying the data, some changes will be made to the structure of the data. Cleansing, normalization, and transformation operations have been applied in the second copying step—data staging to the storage system. As soon as the data is copied, all the data in the data staging space is deleted immediately. Data staging is temporary storage space that is usually small compared to external data sources and storage systems.

4.5 Abstraction Level

In general, database systems are more complex and use different types of data structures. Due to the large number of complexities, many irrelevant details associated with the database are hidden so that the users do not have to

face problems. Therefore, three different abstraction levels have been classified in the whole system: file level, block level, and DBMS level. The file is the lowest abstraction level that contains the actual storage system. It is also known as the "physical view of the storage system. A "block" is an intermediate-level abstraction that specifies what data can be stored in the storage system. At this level, all the properties of a particular object have been defined with their types, relationships, etc. Block level abstraction specifies the logical view of the system.

4.6 Summary

The effectiveness of a cloud storage system largely hinges on the choice and implementation of its data model. A robust data model can significantly enhance the system's ability to scale, manage data efficiently, and provide quick access to users across distributed environments. Conversely, a poorly designed data model can lead to inefficiencies, such as bottlenecks in data access, difficulties in scaling, and increased complexity in managing data consistency and availability. Therefore, the conclusion underscores the importance of selecting a data model that aligns with the specific requirements of the application, the anticipated data growth, and the need for high availability and durability.

5 Data Consistency

Over the last decade, rapidly expanding Internet-based services such as blogging, social networking, search, and e-commerce have dramatically altered how individuals communicate, access resources, exchange information, and conduct transactions. Traditional RDBMS, on the other hand, are facing new challenges as the demand for scalability and new application requirements increases. Not Only SQL (NoSQL) is a new class of low-cost, high-performance database systems that have emerged to threaten relational databases' supremacy. [15]. Data consistency in cloud storage systems is a fundamental aspect that influences how reliably and accurately data is stored and retrieved across distributed nodes. In cloud environments, maintaining consistency becomes challenging due to factors like network latency, system failures, and the distributed nature of data storage. Various consistency models, such as strong consistency, eventual consistency, and causal consistency, offer different trade-offs between performance, availability, and reliability. This section explores the aspects of data consistency model over application design, user experience, and the overall efficiency of the cloud storage system. The mind mapping of data consistency (as shown in Fig. 6) are discussed in detail, including its levels, models, types, measures, protocols, and maintenance.

5.1 Consistency Level

The data consistency level identifies how many replicas in a cluster must be accepted before accepting read or write operations. In general, consistency levels affect the availability, latency, and throughput of the database. The main aim of the cloud storage system is to provide high availability and scalability, which can sometimes come at the expense of strong consistency. Here, the choice of data consistency levels is crucial and is often influenced by factors such as system scale, distribution, and performance requirements.In general cloud database use four consistency levels including, strong, session, bounded staleness, and weak. The impact of different consistency levels are described in Table 4.

5.1.1 Strong consistency. Strong consistency provides a linearity guarantee, which means that service requests are performed concurrently. It is also known as "immediate consistency. Here, data can be viewed immediately after an update so that data will be consistent for all users.

5.1.2 Session consistency. Session consistency is the most widely used consistency level for both single and globally distributed applications. It provides the consistency guarantees that suit the needs of applications

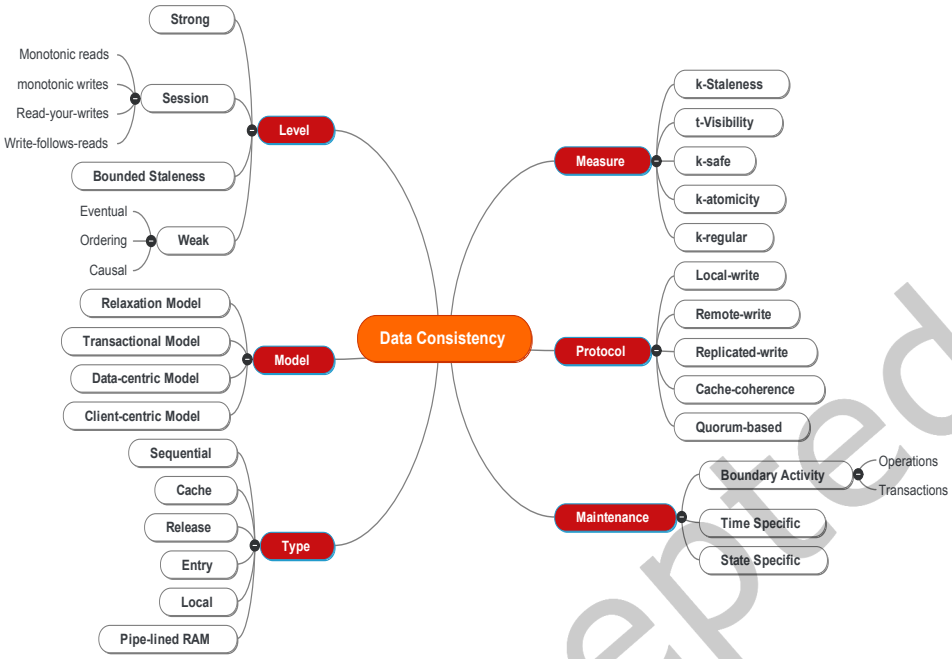

Fig. (6) Mind Mapping of Data Consistency

written to operate in the context of a user. Within a single client session, reads are guaranteed to honor the consistent-prefix, monotonic reads, monotonic writes, read-your-writes, and write-follows-reads guarantees.

5.1.3 Bounded staleness consistency. In bounded staleness consistency, the reads are guaranteed to honor the consistent-prefix guarantee. Generally, "bounded staleness" can be configured in two types: (i) Number of version and (ii) Time interval. It offers total global order outside of the staleness window.

5.1.4 Weak consistency. Weak consistency is also known as eventual consistency. There is no ordering guarantee for reads and further in the absence of any writes, the replicas eventually converge.It is the weakest form of consistency level because a client can read the values that are older than the ones it had read before. Sometime it may be ideal if the application does not require any ordering guarantees.

Table (4) Consistency levels vs. availability, latency and throughput

Consistency Level	Availability	Latency	Throughput
Strong	Low	High	Low
Session	Medium	Medium	Medium
Bounded staleness	Medium	Medium	Medium
Weak	High	Low	High

The Table 4 illustrates the trade-offs between consistency, availability, latency, and throughput, emphasizing that achieving higher consistency often comes at the cost of lower availability and higher latency. Conversely, weaker consistency models can improve availability and throughput, but at the expense of consistency. Strong Consistency offers the most rigorous data consistency guarantees, but at the cost of low availability and high latency. The system ensures that all users see the same data simultaneously, which requires substantial coordination across

the system, leading to reduced throughput. Session consistency provides a balance where data consistency is maintained within a single session, leading to medium availability, latency, and throughput. Bounded Staleness offers medium levels of availability, latency, and throughput. It allows data to be slightly out of date (or "stale") within a defined period, striking a balance between consistency and system performance. Weak consistency emphasizes availability and low latency, resulting in high throughput.

5.2 Consistency Model

A consistency model is a contract between a globally distributed data storage and computation in which the computation agrees to follow certain rules to store promises to work perfectly. In general, a consistency model basically refers to the degree of consistency that should be maintained for the shared memory data [16]. The consistency model can be classified under four categories: i) Relaxation model, ii) Transaction model, iii) Data-centric model and iv) Client-centric model.

5.2.1 Relaxation. Relaxation is one of the less strict consistency model in which sequential consistency requirements are relaxed. In another way, relaxing the program order or writing the need for atomicity to improve the performance of the system. In case of relaxing program order, following operational ordering pairs can be relaxed: write-after-write, read-after-write and read/ write-after-read. A process can view its own writes before any other processes under the relaxed write atomicity paradigm.

5.2.2 Transaction. The transaction model combines the ideas of cache coherency and memory consistency. Typically, a transaction is often a set of activities performed by a process to convert data from a stable state to a stable state. A transaction commits or aborts when there is no dispute. All changes are accessible to all other processes after a transaction is finished, whereas aborts erase all changes. A relaxed consistency method is more difficult to utilise than a transactional consistency model. It also outperforms a sequential consistency method in terms of performance.

5.2.3 Client-centric. The client-centric is one of the cost effective model which specifies the concurrent operations rather than sequential. Some examples of client-centric consistency models are - i) Monotonic-Read (MR) , ii) Monotonic-write (MW) , iii) Read-your-writes (RYW), and iv) Writes-follows-reads (WFR). The first model, monotonic-read, ensures that a client who has read version n will always read version n after that. This is useful because, while data visibility may not be immediate in an application, versions are at least visible in chronological sequence. Monotonic-write consistency ensures that two updates from the same client are serialized in a specific order in which they are received by the storage system. The read-your-write ensures that a client who has written version n will be able to read a version that is at least as new as n in the future. It prevents scenarios where a user or programme sends the same request several times because it believes the initial attempt failed. The last model, write-follows-read ensures that after a read of version n, an update will only run on replicas that are at least version n.

5.2.4 Data-centric. Access to the storage system's real replicas is required for all data-centric metrics. A test application adds to the system's burden. The results are then obtained by mining replica logs, which should contain the following information for each request: start and end timestamps at each replica, a unique request id, and the kind of request (read, write). It is therefore feasible to determine t-Visibility, k-Staleness, the number of ordering violations, and the related consistency model using this information. The relationships between client-centric and data-centric can be specified in the following points:

(α) Single requests may occasionally be guaranteed, but only by chance.
(β) For a high number of requests, a guarantee may be fulfilled.
(γ) The guarantees from single client are fulfilled.

(δ) A guarantee is extended for all clients at same moment.

Each of these data consistency models offers a unique trade-off between consistency, performance, and complexity. The choice of model depends on the specific requirements of the application, such as the need for strong consistency, the importance of low latency, or the ability to scale across distributed systems. Understanding these trade-offs is key to designing systems that meet the desired levels of reliability, efficiency, and user experience.

Table 5 shows the relationship between client-centric and data-centric consistency models. The table compares four client-centric operations—monotonic-read, monotonic-write, read-your-writes, and writes-follow-reads—across four data-centric consistency models: weak, causal, sequential, and linearizability (LIN). For weak consistency, the guarantees are minimal. Under causal consistency, the operations are generally fulfilled when many requests are made, indicating a higher likelihood of consistency. Sequential consistency provides stronger guarantees ensuring consistency for single clients. Finally, the strongest model, LIN, offers the highest level of consistency, ensuring that all clients experience the same data consistency at the same moment.

Table (5) Relationship between client-centric and data-centric consistency model

Client-centric	Weak	Casual	Sequential	LIN
Monotonic-read	α	β	γ	δ
Monotonic-write	α	β	γ	δ
Read-your-writes	α	β	γ	δ
Writes-follows-reads	α	β	δ	δ

Table 6 highlights the analytical comparison of data consistency models along side recommended deployment scenarios and insights. From an analytical perspective, no single consistency model is universally optimal; rather, the choice depends on workload characteristics, failure tolerance, geographic distribution, and application semantics. Strong transactional consistency remains essential for correctness-critical domains, but its scalability limitations make it unsuitable for modern geo-distributed cloud systems. In contrast, relaxed and client-centric models dominate large-scale web, IoT, and edge deployments where latency and availability are prioritized. Emerging data-centric and hybrid approaches represent a shift toward adaptive consistency, enabling systems to dynamically balance correctness and performance based on data importance and access patterns. This trend aligns with contemporary cloud-native and microservices-based architectures, where heterogeneous consistency requirements coexist within the same system. Recent systems increasingly adopt hybrid and adaptive consistency strategies, combining transactional guarantees for critical data with relaxed or client-centric consistency for performance-sensitive operations.

Table (6) Analytical comparison of data consistency models with deployment scenarios

Consistency model	Key strengths	Limitations	Deployment scenarios	Analytical insight
Relaxation model	High throughput, low coordination overhead and high scalability	Temporary inconsistencies and non-deterministic state convergence	Cloud object storage, social platforms, IoT data ingestion	Best suited for read-heavy, latency-sensitive workloads where freshness is less critical.
Transactional model	Guarantees strict correctness and data integrity	High coordination cost, limited scalability, risk of deadlocks under contention	Financial systems, banking transactions, inventory management, mission-critical enterprise systems	Suitable for correctness-critical workloads but poorly aligned with geo-distributed or high-latency environments.
Data-centric	Enables fine-grained consistency control based on data criticality	Increased design and operational complexity	Microservices architectures, multi-tenant SaaS, hybrid cloud platforms	Reflects modern system design by selectively enforcing strong consistency only where required, improving overall efficiency.
Client-centric	Improves perceived consistency, low read latency and reduces user-visible anomalies	Limited global guarantees and cross-client coordination challenges	Mobile apps, collaborative systems, edge and user-centric services	Effective in geo-distributed user environments; commonly layered over eventual consistency at the backend.

5.3 Consistency Type

Consistency in cloud data storage refers to how data consistency is maintained across distributed systems and multiple copies of data stored in the cloud. The consistency type in ultra-large cloud storage system is closely related to the concept of cache consistency in distributed systems, but it also encompasses broader considerations specific to cloud environments. [82] There are six types of data consistency which are listed below [60][3][11]:

Sequential: It is low strict consistency model developed by Lamport. Writes to variables by different processors have to be viewed in the same order by all processors, but not be seen in a write to a variable. All actions must seem to execute instantly or atomically with regard to every other processor in order to maintain the sequence of execution amongst processors.

Cache: It refers to the synchronization of data stored in different caches that might be part of a distributed computing environment. Ensuring consistency in a distributed system is essential to avoid conflicts and discrepancies in data. There are several cache consistency models, each with its own approach to managing the coherence of cached data. There are some common cache consistency types: i) Write-Through, ii) Write-Behind, iii) Write-Once, iv) Write-Invalidating, v) Read-Your-Write. Choosing the appropriate cache consistency type depends on the specific requirements and characteristics of the distributed system and the trade-offs between consistency, availability, and partition tolerance.

Release: It relaxes the weak consistency model by separating the entry synchronization operation from the exit synchronization operation. When a synchronisation operation is observable under weak ordering, all activities in all processors must be visible before the synchronisation operation is completed and the processor continues.

Entry: The release consistency model is a subset of the release consistency model. It also necessitates the use of acquire and release instructions to clearly express a vital section's entry or exit. Every shared variable, on the other hand, is given its own synchronisation variable under entry consistency. It enables the functioning of several crucial sections of various shared variables at the same time.

Local: Local consistency refers to a scenario where consistency is maintained within a local context or a subset of the system, often ignoring the global state. This can occur in distributed systems where local operations are consistent but may temporarily diverge from the global state. For example-In a distributed database, transactions within a single node may be consistent locally, even if the global consistency across all nodes is not immediately achieved.

Pipe-lined RAM: One of the earliest consistency models named pipe-lined RAM (PRAM), which was introduced by Lipton and Sandberg in 1988. All processes in PRAM consistency observe the operations of a single process in the same order as they were issued by that process, but operations issued by separate processes might be viewed in various orders. The consistency of PRAM is less reliable than that of the CPU. PRAM eliminates the requirement for all of its processors to maintain coherence to a single place.

5.4 Consistency Measure

In general, there are two prospective of consistency measurement:client and provider. The client's perspective focuses on the assurances of consistency that a client will be able to see. Client-centric consistency is the name given to this perspective. The provider perspective is also known as data-centric consistency because it relies on internal synchronizing mechanisms and communication between replicas. It totally depends on the perspective to measure different aspects.

5.4.1 k-Staleness. According to consistency models based on the idea of staleness, reads might return old, stale written data. They provide more assurances than eventually consistent semantics, but they are still insufficient to allow more efficient implementations than linearizability. It determines how many versions a certain read fell behind on. It is also dependent on the current system workloads. Reproducibility is on of the main limitation of

k-Staleness consistency measure. It can be reproduced only if there is any possibility to repeatedly regenerate the same workload.

5.4.2 t-Visibility . It reports the distribution function of inconsistency windows and represents staleness in terms of time. It also calculates the chance of a particular inconsistency window length for a large number of executions. t-Visibility measures are repeatable because they give a distribution function of staleness values rather than a single measurement. Storage providers are interested in t-Visibility because it provides comprehensive information on how long it takes for replicas to synchronize after an update. Because any distribution function may be used, t-Visibility is both a continuous and a fine-grained metric because it just measures staleness with no ordering.

5.4.3 Other measures. It's typical to believe that operations within the same memory location are atomic and that they happen in a particular manner. But , this assumption, can be relaxed, allowing for simultaneous operations within the same memory location. In this regards, Lamport has defined three classes of consistency measures - k-Safe, k-Atomicity and k-Regular. Where, k specifies the positive integer. The k-safe option is the most fragile, as it assumes that a read that is not concurrent with any writing would obtain one of the k most recently written values. Except that the value acquired by a read that overlaps a write must be one of the register's possible values, no assumptions are made about the value gained by a read that overlaps a write.

For more information please refer a research paper published by L. Lamport(1985) titled "On Interprocess Communication"[57]. Basic requirements of these measures are shown in Table 7.

Table (7) Consistency measure and its requirements

Measure	Objective	Ordering	Staleness	Continuous	Prospective
k-Staleness	To use the client-observable and data-centric version lag for distribution function.	×	√	√	Both client and data-centric
t-Visibility	To use the both client and data-centric inconsistency windows for distribution function	×	√	√	Both client and data-centric
k-Safe	Maximum version log in violation	√	√	×	Data-centric
k-Atomicity	Maximum version log in violation	√	√	×	Data-centric
k-Regular	Maximum version log in violation	√	√	×	Data-centric

5.5 Consistency Protocol

Consistency protocol provides a systematic way to implement the consistency model. It implements a consistency model by controlling the sequence of operations in accordance with the consistency model. The consistency protocol can be categorized under two parts: i) Primary-based protocols and ii) Replicated-write protocols Both are implemented with specific consistency model.

5.5.1 Primary-based protocols. The primary-based protocols are easier to put into practice. Sequential ordering is an example of the primary-based protocol. The local-write and remote-write both can be considered as primary-based protocols. Each data item has a primary server which is responsible for coordinating its changes. All read and write operations have been performed on data items at the same server. In local-write protocols, primary copy moves between processes willing to perform an update.

5.5.2 Replicated-write protocols. In this protocol, write operations are propagated to all replicas, while reads are performed locally. The writes can be propagated using either point-to-point communication or multicast. In distributed systems, achieving consistency in the face of data replication involves defining protocols for how writes are handled across multiple replicas. There are several protocols, and the choice depends on the specific requirements of the system. Some common replicated write protocols include the Quorum-based and Consensus-Based protocols. These protocols ensure that updates are properly synchronized across replicas to

Table (8) Comparison between Primary-Based and Replicated-Write Protocols

Criteria	Primary-Based Protocols	Replicated-Write Protocols
Consistency	Easier to ensure strong consistency via the primary	It requires complex consistency mechanisms like quorum or consensus
Write Handling	Centralized-handled by a primary node	Decentralized-handled by multiple replicas
Complexity	Simpler to implement but requires failover mechanisms	More complex due to distributed coordination
Fault Tolerance	Primary failure requires failover	Higher fault tolerance, as multiple nodes can accept writes
Use Cases	Traditional relational databases, single-master systems	Distributed systems, NoSQL databases, large-scale distributed databases

maintain consistency in the distributed system. The choice of protocol depends on factors such as fault tolerance requirements, performance considerations, and the desired level of consistency guarantees.

For better readability, the comparison between both Primary-based and Replicated-based protocols are tabulated in Table 8. The primary-based protocols are simpler to implement and offer strong consistency but can suffer from bottlenecks or availability issues if the primary fails. Replicated-write protocols distribute the responsibility for updates across multiple replicas, providing better fault tolerance and availability. However, they are more complex to manage and require mechanisms like quorum or consensus to ensure consistency.

5.6 Major challenges in Data Consistency

The selection of appropriate consistency metrics that meet the requirements outlined in this section normally takes place. However, there are still a lot of challenges when looking at the consistency behavior which are mentioned below:

i Reproducibility: Under the premise that the storage system performs similarly, the measurement component should produce same values when repeating a system benchmark.
ii Wide applicability: A measuring strategy should not be restricted to a single system. Instead, it should be able to cover a wide range of systems in order to ensure that the findings are comparable.
iii Resolution:A method for evaluating and measuring a system should not only utilise a measure with sufficient resolution, but also be able to use it.
iv Accuracy: Measurement mistakes are inevitable, but they should be minor in comparison to the values being measured.
v Microservices Architecture: One logically atomic activity in microservices can commonly span several microservices. Multiple databases or communications technologies may be used by even a monolithic system.

5.7 Summary

The choice of data consistency model in a cloud storage system has profound impacts on the system's performance, scalability, and user satisfaction. While strong consistency ensures data accuracy, it can impose limitations on availability and responsiveness, making it less ideal for certain cloud applications that prioritize speed and uptime. Conversely, eventual consistency provides greater flexibility and efficiency in distributed environments but requires applications to handle the possibility of temporary inconsistencies. The conclusion highlights that there is no one-size-fits-all solution; instead, the optimal consistency model depends on the specific requirements of the application, such as the need for real-time data accuracy versus the need for high availability and low latency.

6 Data Replication

Data replication in ultra-large cloud storage systems is a critical strategy for ensuring data availability, fault tolerance, and disaster recovery across a distributed infrastructure. By creating multiple copies of data across different geographical locations or storage nodes, replication enhances the system's resilience against hardware failures,

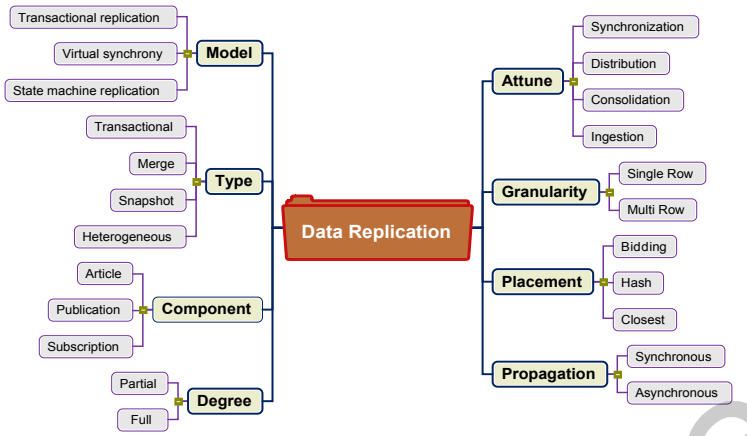

Fig. (7) Mind Mapping of Data Replication

network outages, and data corruption. However, implementing data replication at scale introduces challenges such as managing the trade-offs between consistency, latency, and storage costs. Synchronous replication, which requires all copies to be updated simultaneously, ensures strong consistency but can lead to increased latency and reduced system performance, particularly in geographically dispersed environments. Asynchronous replication, on the other hand, offers better performance and lower latency by allowing updates to propagate across replicas with a delay, but it may lead to temporary inconsistencies. Data is replicated to enable high availability, backup, and/or disaster recovery in organizations. The mind mapping of data replication has been shown in Fig. 7

6.1 Replication Model

6.1.1 Transactional replication. A snapshot of the published database objects and data is usually taken before transactional replication begins. Following the initial snapshot, the publisher typically sends subsequent data updates and schema changes to the subscriber as they happen (near real time). The data updates are applied to the subscriber in the same sequence and within the same transaction boundaries as they were applied to the publisher; hence, transactional consistency is assured within a publication.

6.1.2 Virtual Synchrony. Virtual synchrony is a term used to describe dynamic process groups with self-managed membership. Applications can join or leave process groups at any time: a process group is similar to a networked replicated variable. The second requirement has to do with performance. The virtual synchrony replication concept has been extensively used, with applications ranging from air traffic control and stock market systems to IBM and Microsoft's data centre management platforms [18].

6.1.3 State Machine Replication. State machine replication is a generic approach for creating a fault-tolerant service by duplicating servers and coordinating client interactions with replicas. In addition, the method provides a foundation for comprehending and creating replication management methods [84]. It is made up of state variables that store information about its current state and instructions that change it. Each command is implemented by a deterministic programme; the command's execution is atomic in comparison to other commands, and it alters state variables and/or generates some output. A state machine's client requests that a command be executed. The request identifies a state machine, specifies the command to be executed, and includes any further information that the operation may require. Request processing can send data to an actuator (for example, in a process control system), a disc or a terminal, or clients who are waiting for replies to previous requests.

6.2 Replication Type

There are several forms of replication used by organizations today, depending on the data replication technologies used. The following are some of the most prevalent replication types:

Transactional replication: In this approach,the data replication software creates entire initial copies of data from origin to destination, after which the subscriber database receives updates anytime data is updated. Because fewer rows are duplicated each time data is updated, this replication option is more efficient. In server-to-server situations, transactional replication is most common.

Merge replication: This type of replication is common in server-to-client scenarios and allows both the publisher and the subscriber to make dynamic data changes. Merge replication adds to the technique's complexity by combining data from two or more databases to form a single database.

Snapshot replication: Data is duplicated exactly as it appears at any given point in time with Snapshot replication. Snapshot replication, unlike other techniques, is unaffected by data changes. When data changes are rare, such as when performing initial synchronizations between publishers and subscribers, this style of replication is employed.

6.3 Replication Component

Today, the amount of data stored, handled, and accessible is unparalleled. Businesses want their IT departments to maintain data online and make it available indefinitely, placing a lot of strain on the systems that store and handle it. We need to replace obsolete and inefficient legacy procedures with new, more agile approaches to suit today's demands. One approach to meeting these objectives is data replication. Data replication involves creating and maintaining multiple copies of data across different storage systems or locations. By implementing data replication, businesses can enhance data availability, improve disaster recovery capabilities, and reduce the risk of data loss. Additionally, it allows for faster access to information and enables better scalability to accommodate the growing volume of data in today's digital landscape.

6.4 Replication Degree

Data replication entails the continuous duplicating of transactions such that the duplicate is always up to date and synced with the source. In data replication, however, data is available in several locations, but a specific relation must reside in only one. The degree of replication can be classified either full or partial. Full replication, in which the whole database is stored at each location, is possible. The most severe example involves replicating the whole database across all of the distributed system's sites. Because the system can continue to run as long as at least one site is up, the system's availability will increase. Partial replication is also possible, in which some frequently used database fragments are replicated while others are not.

6.5 Replication Attune

The database is synchronized in synchronous replication such that all replications have the same value. In all of the sites, a transaction seeking a data item will have access to the same value. A transaction that changes a data item is extended to make the modification in all copies of the data item to guarantee consistency. In most cases, the two-phase commit technique is employed.

6.5.1 Synchronization. Synchronization implies that two or more copies of data are being kept up-to-date, but not necessarily that each copy contains all of the data. Generally, it will be followed after a certain predetermined period of time and amounts to the re-merging of the changes made to the different replicas of the database into the master database. It is used in distributed databases and systems to ensure that changes made on one node are immediately and consistently replicated on one or more other nodes. It also helps to maintain data consistency and integrity across the distributed system [72].

6.5.2 Distribution. Distributed replication is the process of replicating data across multiple nodes or different locations in a distributed system. The main goal is to ensure data availability, fault tolerance, and load distribution. It is commonly used in various distributed computing environments, including distributed databases, cloud storage systems, and content delivery networks, to improve performance and scalability.

6.5.3 Consolidation and Ingestion. Consolidation replication centralizes and consolidates their data in a single location, making it easier to analyze and derive insights from. By ingesting data from various sources, businesses can gain a comprehensive view of their operations and make data-driven decisions. Ingestion replication is the process of moving and replicating data from data sources to a destination, such as a cloud data lake or cloud data warehouse.

6.6 Replication Granularity

Granularity in data replication refers to the level of detail or size of the data units that are replicated between different nodes or locations in a large scale distributed storage system. It involves determining what portions of the data are replicated and how often the replication process occurs. Some levels of granularity are- Record-Level, Table-Level, Database-Level , File-Level and Object-level. The choice of replication granularity depends on various factors, including the nature of the data, the frequency of updates, the size of the data units, and the performance requirements of the system. Finer granularity (e.g., record-level replication) can provide more granular control and real-time consistency but may incur higher overhead, while coarser granularity (e.g., database-level replication) may simplify the replication process but could lead to unnecessary data transfer in some scenarios.

6.7 Replication Placement

Data replication placement refers to the strategic decisions made about where to store replicated copies of data in a distributed system. Replication is often used in distributed systems to improve fault tolerance, availability, and performance. Deciding where to place these replicated copies is a crucial aspect of designing a system that meets its goals efficiently. There are various factors that need to be considered when determining the placement of replicated data. These factors include network latency, data access patterns, and the overall system architecture.

6.8 Replication Propagation

In general, replication is set up between data nodes using two different data replication strategies: synchronous and asynchronous. Many copies of the same data are made and distributed across multiple nodes. The sole motivation behind of replication propagation are listed below:

- Keep data geographically close to the user.
- Improve data availability.
- Improve performance by taking advantage of parallel reads in modeled data.
- Prepare for disaster recovery.

In Synchronous Replication, the Master node begins the write operation on its replicas after updating its own copy of the data. When the Replicas get the update, they make the necessary changes to their copy of the data and then confirm it to the Master. The Master responds to the client and completes the action after it has received confirmation from all of the Replicas.

6.9 Summary

Data replication is indispensable for the reliability and efficiency of ultra-large cloud storage systems, as it plays a vital role in maintaining data availability and integrity across distributed environments. The choice between

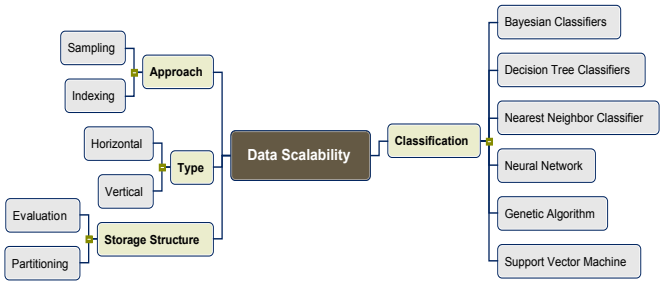

Fig. (8) Mind Mapping of Data Scalability

synchronous and asynchronous replication methods is often dictated by the specific needs of the application, with some prioritizing strong consistency and others valuing performance and cost-effectiveness.

7 Data Scalability

Data scalability is a fundamental concern in ultra-large cloud storage systems, where the ability to efficiently handle increasing amounts of data and users is crucial. Scalability involves not only adding storage capacity but also ensuring that performance metrics such as latency, throughput, and data access times remain consistent as the system grows. It is the process of adding or withdrawing compute, storage, and network services in cloud computing to satisfy the demands a workload places on resources to maintain availability and performance as usage grows. It encompasses the capability to expand resources, accommodate additional data storage needs, and deliver consistent performance even in the face of increasing workloads. The mind mapping of data scalability has been visualized in Fig. 8

7.1 Approach

In the context of data scalability, a sampling approach involves using a subset of the data to perform analysis or operations instead of working with the entire dataset. This technique is particularly useful when dealing with large datasets, as it helps reduce computational and storage requirements while still providing insights into the overall characteristics of the data. Indexing is a crucial component of data scalability, as it significantly influences the efficiency of data retrieval operations. Proper indexing can enhance query performance and enable systems to scale effectively with growing datasets. By using indexing, the system can quickly locate and retrieve specific data points, reducing the time and resources required for searching through the entire dataset.

7.2 Type

Data scalability refers to the ability of a system to efficiently handle and manage a growing volume of data as the demands on the system increase. Different types of data scalability address various aspects of system design and architecture to ensure that the system can scale effectively. Scalability can be achieved by two important aspects:

i Horizontal scaling - It connects multiple clouds to integrate and creates single logical cloud. A cloud with a computational affinity might be accessed data from heterogeneous platform. The relationship between cloud data integrates with compute-data is more crucial.

ii Vertical scaling: It improves the cloud storage and computing capacity by magnifying the existing systems and enhancing the communication bandwidth. It has ability to increase the current resources by adding a new database, processing power, etc.

7.3 Storage Structure

Evaluation: The storage structure in cloud storage systems is a critical factor that influences how data is stored, managed, and accessed at scale. Evaluating the storage structure involves assessing how data is organized across different storage layers, including object stores, block storage, and file systems. The evaluation process also considers how the storage structure supports data redundancy, fault tolerance, and data access patterns, ensuring that the system can handle both current and future data needs. Key metrics in this evaluation include storage efficiency, latency, throughput, and the system's ability to manage large volumes of data without performance degradation.

Partitioning: It involves dividing a large dataset into smaller, manageable pieces, or "partitions," which can be distributed across multiple storage nodes. This approach enables the system to scale horizontally, as additional partitions can be allocated to new nodes when the system needs to grow. Partitioning can be implemented using various techniques, such as range-based partitioning, where data is divided based on key ranges, or hash-based partitioning, where data is distributed according to the output of a hash function.

7.4 Classification

Data scalability classifications revolve around strategies and approaches to handle increasing data volumes, user loads, and performance requirements. Cloud storage systems leverage various scalability classifications to ensure flexibility, efficiency, and reliability. Some major classification algorithms are Bayesian classifiers, decision trees, k-nearest neighbor, neural networks, genetic algorithms, support vector machine, etc. These algorithms are commonly used in cloud storage systems to classify and organize data based on its characteristics and patterns. By implementing these scalability classifications, cloud storage systems can effectively handle large amounts of data while maintaining optimal performance and meeting user demands. These scalability classifications enable cloud storage systems to efficiently allocate resources and distribute data across multiple servers.

7.5 Summary

Data scalability is essential for the long-term viability of ultra-large cloud storage systems, as it directly impacts their ability to support growing datasets and user bases without sacrificing performance. The conclusion underscores that a scalable storage system must be designed with flexibility in mind, allowing for seamless expansion while maintaining reliability, availability, and efficiency. The architecture should be modular, enabling incremental scaling through the addition of storage nodes or resources without significant reconfiguration. Moreover, the system should incorporate intelligent data distribution and load balancing mechanisms to ensure that the increased scale does not lead to bottlenecks or resource imbalances.

8 Data Integration

This section highlights the various aspects of data integration schemes, approaches and stages. For a better understanding of data integrity schemes, we made a thematic taxonomy based on the type of data they deal with, the type of metadata they use, their capabilities, the implementation primitives they use, and the deployment scenarios they are used in. Due to their low cost and potentially infinite scalability to satisfy ever-increasing data requirements, cloud computing services have established a footing in the industry. The mind mapping of the data integration is visualized in Fig. 9.

8.1 Data Integration Approach

The two well-known methods for ensuring data integrity are statistical and deterministic techniques. The statistical techniques employ random selected blocks of data to evaluate the integrity and provide less than a 100% assurance. Whereas, the deterministic technique requires access to the entire file in order to assess the

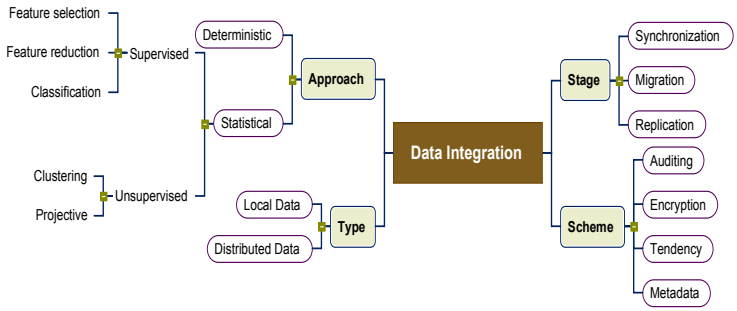

Fig. (9) Mind Mapping of Data Integration

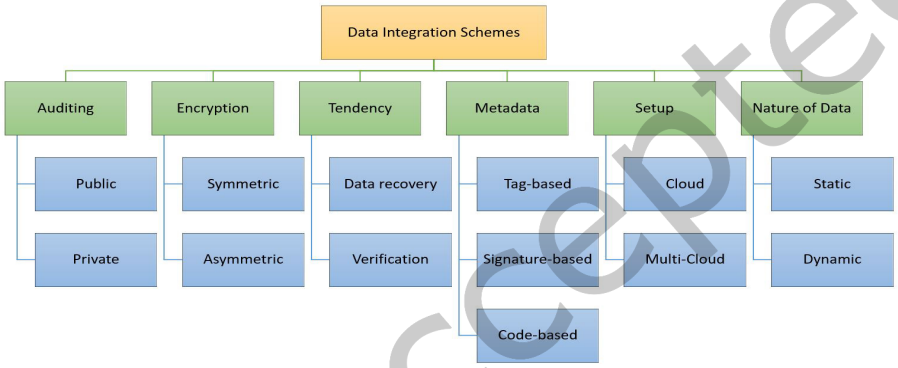

Fig. (10) Data Integration Schemes

integrity and provide a 100% possession guarantee[36]. Deterministic techniques are not appropriate for huge files, such as archives with data sizes in the GBs or TBs, as it would take too long to verify the integrity of such a file. As a result, the user's ability to perform integrity verification may be limited. Therefore, for huge datasets, statistical techniques are more suitable.

8.2 Data Integration Schemes

Any data corruption or deletion may be discovered quickly with the use of data integrity schemes, and required steps can then be done to recover the data[102]. To help people better understand data integrity schemes, we created a thematic taxonomy based on the type of data they deal with, the type of metadata they use, their capabilities, the implementation primitives they use, and the deployment scenarios they leverage. Fig. 10 visualized the detail taxonomy of various data integration schemes.

8.2.1 Auditing. Data integrity plans allow for both public and private audits. Public auditing raises privacy issues related to user anonymity and data leakage. Public auditing strategies that protect privacy are further broken down into groups. Krzywiecki and Kuty[53], provide broadcast encryption with open auditability and secure pseudorandom numbers (SPRN) with Lagrangian interpolation. Public auditing is permitted while keeping anonymity thanks to traceability, which enables the origin user to track a signature into a block and identify the signer. Another important features and ability studied by Wang et at.[95] to support a large number of users without compromising the efficacy of the verification process. According to Shen and Tzeng's[87] approach,

delegation of auditing authority is encouraged; re-delegation is not, however, authorised. An auditing user who has been given authorization cannot provide it to another user. Wang[96] has described about the pros and cons of the proposed method for cloud storage called Proprietary Secure and Efficient Proxy Provable Data Possession (PPDP).

8.2.2 Encryption. The process of data integration involves gathering information from many sources and presenting it as a single, cohesive dataset. Over the year's data, integrity schemes have utilized both symmetric and asymmetric encryption for security. Authors Ateniese et al.[10] achieved scalability and efficiency by using symmetric key encryption and cryptographic hash functions. In 2009, Wang et al.[97] suggested a method for retrievability proof that makes use of BLS signatures that enable both public verifiability and operations on dynamic data. A cooperative PDP (CPDP) system from bilinear pairings is presented by Zhu et al [105] .'s suggested technique to offer knowledge soundness. It is based on homomorphic verifiable response (HVR) and hash index hierarchy (HIH). An effective data integrity verification methodology was presented by Lee and Chang[59] utilising a hybrid approach that uses both symmetric and asymmetric encryption.

8.2.3 Tendency. A data integrity scheme's inclination can either be verification solely or verification plus data recovery. Both of these tendencies are possible. As per tendency, data integrity schemes can be divided into two categories: ownership of the data and proof of retrievability[99]. Ownership schemes are based on probabilistic models because they pick random blocks to check instead of reading the whole file. The initial data are preprocessed to produce some information. These metadata are kept with the original data and then used to verify the veracity of user data.

8.2.4 Metadata. The majority of data integrity schemes combine the original data with some extra metadata, which is used throughout the integrity verification process. These metadata might be "tags" having homomorphic, verifiable qualities that aid in the production of an aggregate value throughout the verification process[101][103]. The metadata might be "signatures," which are used in place of tags, such as algebraic signatures to advance methods like Reed-Solomon codes. Finally, network codes that also permit data recovery might be utilised as metadata in place of tags and signatures, which could be used to detect minor data[9][20].

8.2.5 Setup. The setup describes the type of scenario where the data integrity technique will be implemented. Since data is stored in the cloud, which can be straightforward, "hybrid-cloud," or "multi-cloud," data integrity schemes must meet some extra requirements as a result of the setup. Data integration in a cloud storage system involves leveraging cloud services and tools to seamlessly combine and manage data from various sources. The key objectives of the integration setup are: cloud-based data extraction, data transformation in the cloud, loading data into cloud storage, data quality assurance, metadata management, and integration workflow. The authors (Curtmola et al.)[32] proposed a scheme named "Multiple Replicas Provable Data Possession (MR-PDP)" which makes new copies of the data on demand if the originals are lost or changed.

8.2.6 Nature of Data. This attribute describes the nature of data, on which the scheme is applicable. Data can be either of a static or dynamic nature[101]. The nature of data also plays an important role in scheme evaluation[31][103]. The scheme proposed by Ateniese et al.[9] is only suitable for static data, because it requires re-computation of complete file tags when new data is inserted in between existing data. Similarly, Ateniese et al.[10], provide support for modification, deletion, and appending operations.

8.3 Summary

Data integration is critical for unlocking the full potential of ultra-large cloud storage systems, enabling organizations to leverage comprehensive insights and drive better decision-making. Effective integration strategies ensure

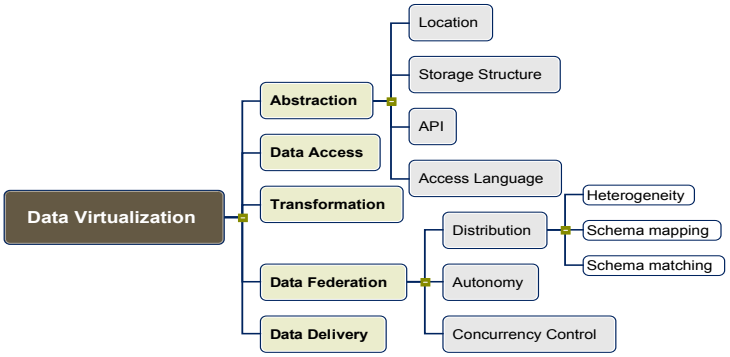

Fig. (11) Mind Mapping of Data Virtualization

that data from various sources is seamlessly merged and made accessible across the entire cloud infrastructure, supporting everything from routine operations to advanced analytics.

9 Data Virtualization

Data virtualization is a logical data layer that integrates all business data from different systems, controls unified data for centralized security and governance, and distributes it to business users in real-time. It refers to technologies that offer a layer of abstraction between the layers of hardware and software, as well as between layers of software, allowing for more straightforward interactions with it. The mind mapping of data virtualization is shown in Fig. 11. It exhibits the various aspects of data virtualization technology, including abstraction, data access, transformation, data federation, and delivery. The act of expressing important functionalities while keeping the background information hidden from users and developers is known as abstraction. Abstraction and virtualization are comparable, however virtualization does not necessarily conceal the specifics of the base layer. In the field of computers, the word "abstraction" is employed on many levels. The location, storage architecture, API, and access language are the sole foundations of an abstraction of data virtualization. To build an autonomous data platform, we will need tightly integrated clouds, including those from third parties, but most of all, we will need a Cloud Native Framework made with the CNCF community in mind[77]. Virtualization is the foundation of cloud computing because it enables more efficient use of actual computer hardware. It enables a business to obtain a greater return on the investment it makes in its hardware.

10 Data Transaction

A "transaction" is a set of activities carried out by one or more entities. Each transaction is guaranteed to be atomic, which means that transactions are never partially applied. Either all or none of the transaction's activities are executed. The mind mapping of data transaction has been exhibited in Fig. 12. The data transaction taxonomy highlights the four different aspects of the data transaction in a cloud environment: type, isolation, correctness, and dependency. The three main kinds of transaction types are read-write, read-only, and write-only. Each transaction type serves different purposes and has specific requirements in terms of access and modification of data.

Web services today typically use scalable storage systems that partition data over several servers since their data has gone beyond the capacity of a single machine. Transactional isolation is necessary for the reads in order to read data consistently across several shards. Fekete et al.[43] propose criteria for avoiding non-serializable transaction execution in order to ensure serializability when the database system ensures strong SI. They propose

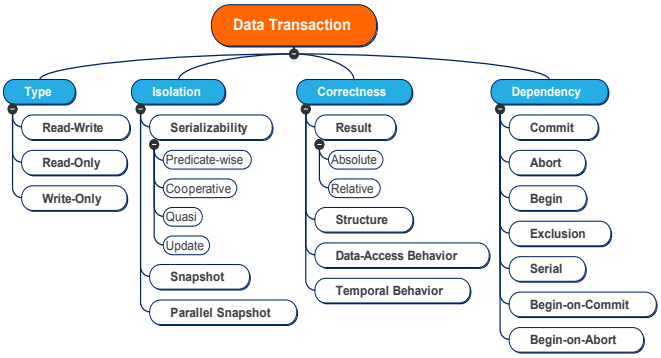

Fig. (12) Mind Mapping of Data Transaction

statically altering the workload, such as by generating conflicts between transactions such that they are effectively serialized under the first-committer-wins rule. Schenkel et al. investigate how 1SR may be assured for transactions running on a database system that provides strong SI.

Transactional dependencies are categorized by type as ordered conflicts that may occur in multi-version histories. For instance, one type relates to two transactions that produce successive versions of the same data item, and another type relates to a write that creates an item version that is seen by an item read that occurs later[43]. Adya et al.[2] introduced the dependence definitions, which may be used with any multi-version history and can even be applied to different locking isolation levels. In fact, in this article, the authors have suggested using the dependencies they established in their ANSI definition of isolation level, which would be impartial toward the particular concurrency management mechanism. Adya et al.[2] have defined the transaction dependency, which is based on an ordering of the versions of each data item, i.e., knowing which prior version each version replaces.

11 Open Research Issues and Directions

Cloud computing plays an important role in sharing data or moving information from one place to another. It shares data to the users in efficient way and gives quick response to the users. Despite this, there are many flaws in cloud computing. All these flaws move towards research itself. In this section, some major research issues has been addressed related to cloud storage systems in cloud computing.

11.1 Dynamicity and Scalability Issues

A large number of organizational units that share a shared computing infrastructure implement their special blend of each task, and their overall demand and patterns of arrival are more dynamic. Such environment must be characterized by a large number of on-demand requested resources within short intervals with high variation. There will be a number of scheduling results, including the dynamic Virtual Network Function (VNF) [80] rapid task scheduling decision, the amendment of the previous assignment decisions and the long-running jobs include the resident of Dynamic Resource Request which, with the help of time, predicts the intervention of resources difficult to predict. Some issues include: SLA violation, poor prediction, resources constraints and workload heterogeneity and variability.

11.2 Ultra-Large Data Integration Issue

Cloud data integration is more challenging and time consuming process to coordinate different resources, infrastructures and applications. Before migrating a particular cloud domain to other environment; it creates

a plan for how data integration service will be assessed the right technology for the cloud storage systems. There are some critical issues related to data integration include: Moving data tracking [13], data migrating, lack of performance, firewall mediation, maintenance and up-gradation [63]. With little planning and some good techniques, data integration issues get resolved quickly, and the cloud storage system becomes so obvious.

11.3 Computation Efficiency

Before outsourcing the data to a cloud storage server, all data integrity techniques preprocess the information. Similarly, metadata is created from the original data on the cloud storage server. The cost of carrying out such processing might reduce how well data integrity algorithms compute. For small datasets, the preprocessing phase's computational efficiency doesn't matter, but for large datasets, it has a significant influence. The frequency with which a user can check the accuracy of outsourced data is constrained by the processing cost on the server side for the proof creation. The data integrity scheme's usage of primitives as metadata affects computation times as well.

11.4 Infrastructure Issue

Massive amounts of data require infrastructure and space for storage, which usually means purchasing state-of-the-art servers that will take up a lot of space in the office or facility. One of the easiest fixes is to employ cloud hosting and cloud storage, which utilize the infrastructure of another business to save space and the headache of setting things up themselves.

11.5 High Energy Consumption Issue

Scientific computing, business applications and other application domains are continuous improvement demands in performance which must be taken care during deployment of cloud storage systems [91]. Reasons of high energy consumption includes equipment's cooling purpose and transferring power to whole infrastructure [90]. The energy consumed by the servers contributes to 70% of the total consumption of a cloud data center. However, such improvement has increased the energy consumption limit. According to Amazon energy consumption cost for cloud-oriented storage server is 47%, cooling infrastructure is 23%, network devices is 21% and remaining 9% of others. Thus, the objective is to minimize the total power consumption of all the cloud-oriented storage systems. Energy consumption in storage systems are mainly calculated from server, network devices and disk storage.

11.6 Complications in managing data Consistency and data replication

Maintaining data consistency across multiple symmetrical nodes is a much more complicated task in distributed data replication systems as compared to centralized system. Another major concern is ensuring consistency while balancing the costs of read and write operations. To address this problem, several protocols have been proposed. But these protocols resolve the issue at some level such as network partitions, node failures, trade-offs between consistency models and complex topology.Therefore, we need an efficient approach to get around this problem while keeping the costs of read and write operations even [65].

11.7 Data loss during normal operations

Network communication and other related issues are invariably associated with distributed environments. Similarly, data replication and management systems in cloud computing are facing the same challenges. Reliable communication channels must be established in order for data to be available and to avoid data loss during normal operations. A great deal of research has gone into developing group communication protocols. Data replication requires a reliable multicast with total order to ensure orderly replica updates, and the presence of

group communication limits the scalability. Furthermore, the group communication layer complicates network configuration and tuning[24][1].

12 Conclusions

Data has been produced in massive quantities over the last few decades via internet-connected devices and applications. The main reasons for data growth are continuous increases in computational power and advancements in recent technologies such as Internet of Things. In this study, we reviewed seven major elements of cloud-based storage systems, including the data model, data consistency, data replication, data scalability, data integration, data virtualization, and data transactions. In this study, additional mind mapping of each element is shown alongside its components. We have conducted an in-depth study on each element, including suitable diagrams and examples. After a regressive study of different aspects of the data modeling, we found lots of major issues that have been analyzed by some well-qualified sources. All of these issues must be addressed in order to conduct further research. Mainly, we observed that data consistency, data integration, and data virtualization have several issues that are tabulated in the corresponding sections. The existing consistency algorithms and protocols are not sufficient for ultra-large storage databases. For the same reason, some efficient and optimized approaches should be implemented. Research must concentrate on these issues and develop efficient techniques.

References

[1] Takoua Abdellatif, Emmanuel Cecchet, and Renaud Lachaize. 2004. Evaluation of a group communication middleware for clustered J2EE application servers. In *OTM Confederated International Conferences" On the Move to Meaningful Internet Systems"*. Springer, 1571–1589.

[2] Atul Adya, Barbara Liskov, and Patrick O'Neil. 2000. Generalized isolation level definitions. In *Proceedings of 16th International Conference on Data Engineering (Cat. No. 00CB37073)*. IEEE, 67–78.

[3] Mustaque Ahamad. 1999. Scalable consistency protocols for distributed services. *IEEE Transactions on Parallel and Distributed Systems* 10, 9 (1999), 888–903.

[4] Mushtaq Ahmed, Kunwar Pal, and MC Govil. 2019. Utilization-based hybrid overlay for live video streaming in P2P network. In *Recent findings in intelligent computing techniques*. Springer, 331–338.

[5] Talal Alsedairy, Yinan Qi, Ali Imran, Muhammad Ali Imran, and Barry Evans. 2015. Self organising cloud cells: a resource efficient network densification strategy. *Transactions on Emerging Telecommunications Technologies* 26, 8 (2015), 1096–1107.

[6] EC Amazon. 2015. Xeround Cloud Database for MySQL Applications. *Available in: https://aws.amazon.com/marketplace/pp/B0085GVT7O(November 2012)* (2015).

[7] Stephanos Androutsellis-Theotokis and Diomidis Spinellis. 2004. A survey of peer-to-peer content distribution technologies. *ACM computing surveys (CSUR)* 36, 4 (2004), 335–371.

[8] Artur Andrzejak, Sven Graupner, Vadim Kotov, and Holger Trinks. 2002. Algorithms for self-organization and adaptive service placement in dynamic distributed systems. (2002).

[9] Giuseppe Ateniese, Randal Burns, Reza Curtmola, Joseph Herring, Lea Kissner, Zachary Peterson, and Dawn Song. 2007. Provable data possession at untrusted stores. In *Proceedings of the 14th ACM conference on Computer and communications security*. 598–609.

[10] Giuseppe Ateniese, Roberto Di Pietro, Luigi V Mancini, and Gene Tsudik. 2008. Scalable and efficient provable data possession. In *Proceedings of the 4th international conference on Security and privacy in communication netowrks*. 1–10.

[11] Hagit Attiya and Jennifer L Welch. 1994. Sequential consistency versus linearizability. *ACM Transactions on Computer Systems (TOCS)* 12, 2 (1994), 91–122.

[12] Gabriel Terna Ayem, Salu George Thandekkattu, and Narasimha Rao Vajjhala. 2022. Review of interoperability issues influencing acceptance and adoption of cloud computing technology by consumers. In *Intelligent Systems and Sustainable Computing: Proceedings of ICISSC 2021*. Springer, 49–58.

[13] Tapas Badal, Neeta Nain, and Mushtaq Ahmed. 2018. Online multi-object tracking: multiple instance based target appearance model. *Multimedia Tools and Applications* 77, 19 (2018), 25199–25221.

[14] Peter Bailis and Ali Ghodsi. 2013. Eventual consistency today: Limitations, extensions, and beyond: How can applications be built on eventually consistent infrastructure given no guarantee of safety? *Queue* 11, 3 (2013), 20–32.

[15] David Bermbach. 2019. *Benchmarking eventually consistent distributed storage systems*. KIT Scientific Publishing.

[16] David Bermbach and Jörn Kuhlenkamp. 2013. Consistency in distributed storage systems. In *International Conference on Networked Systems*. Springer, 175–189.

[17] Christopher D Bienko, Marina Greenstein, Stephen E Holt, Richard T Phillips, et al. 2015. *IBM Cloudant: Database as a Service Advanced Topics*. IBM Redbooks.
[18] Ken Birman. 2010. A history of the virtual synchrony replication model. In *Replication*. Springer, 91–120.
[19] Spyros Blanas, Jignesh M Patel, Vuk Ercegovac, Jun Rao, Eugene J Shekita, and Yuanyuan Tian. 2010. A comparison of join algorithms for log processing in mapreduce. In *Proceedings of the 2010 ACM SIGMOD International Conference on Management of data*. ACM, 975–986.
[20] Kevin D Bowers, Ari Juels, and Alina Oprea. 2009. HAIL: A high-availability and integrity layer for cloud storage. In *Proceedings of the 16th ACM conference on Computer and communications security*. 187–198.
[21] Aron Brand. 2017. Storage device and method thereof for integrating network attached storage with cloud storage services. US Patent 9,614,924.
[22] Marcos M Campos and Gail A Carpenter. 2001. S-TREE: self-organizing trees for data clustering and online vector quantization. *Neural Networks* 14, 4-5 (2001), 505–525.
[23] Bogdan Alexandru Caprarescu, Nicolo Maria Calcavecchia, Elisabetta Di Nitto, and Daniel J Dubois. 2010. Sos cloud: Self-organizing services in the cloud. In *International Conference on Bio-Inspired Models of Network, Information, and Computing Systems*. Springer, 48–55.
[24] Emmanuel Cecchet, George Candea, and Anastasia Ailamaki. 2008. Middleware-based database replication: the gaps between theory and practice. In *Proceedings of the 2008 ACM SIGMOD international conference on Management of data*. 739–752.
[25] Fay Chang, Jeffrey Dean, Sanjay Ghemawat, Wilson C Hsieh, Deborah A Wallach, Mike Burrows, Tushar Chandra, Andrew Fikes, and Robert E Gruber. 2008. Bigtable: A distributed storage system for structured data. *ACM Transactions on Computer Systems (TOCS)* 26, 2 (2008), 4.
[26] Artem Chebotko, Andrey Kashlev, and Shiyong Lu. 2015. A big data modeling methodology for Apache Cassandra. In *Big Data (BigData Congress), 2015 IEEE International Congress on*. IEEE, 238–245.
[27] I-Pao Chen. 2013. Flash storage device, data storage system, and data writing method. US Patent 8,380,919.
[28] Jun Chen, Xing Wu, Shilin Zhang, Wu Zhang, and Yanping Niu. 2012. A decentralized approach for implementing identity management in cloud computing. In *2012 Second International Conference on Cloud and Green Computing*. IEEE, 770–776.
[29] Kristina Chodorow. 2013. *MongoDB: The Definitive Guide: Powerful and Scalable Data Storage*. " O'Reilly Media, Inc.".
[30] Brian F Cooper, Raghu Ramakrishnan, Utkarsh Srivastava, Adam Silberstein, Philip Bohannon, Hans-Arno Jacobsen, Nick Puz, Daniel Weaver, and Ramana Yerneni. 2008. PNUTS: Yahoo!'s hosted data serving platform. *Proceedings of the VLDB Endowment* 1, 2 (2008), 1277–1288.
[31] Reza Curtmola, Osama Khan, and Randal Burns. 2008. Robust remote data checking. In *Proceedings of the 4th ACM international workshop on Storage security and survivability*. 63–68.
[32] Reza Curtmola, Osama Khan, Randal Burns, and Giuseppe Ateniese. 2008. MR-PDP: Multiple-replica provable data possession. In *2008 the 28th international conference on distributed computing systems*. IEEE, 411–420.
[33] Jeffrey Dean and Sanjay Ghemawat. 2010. MapReduce: a flexible data processing tool. *Commun. ACM* 53, 1 (2010), 72–77.
[34] Giuseppe DeCandia, Deniz Hastorun, Madan Jampani, Gunavardhan Kakulapati, Avinash Lakshman, Alex Pilchin, Swaminathan Sivasubramanian, Peter Vosshall, and Werner Vogels. 2007. Dynamo: amazon's highly available key-value store. In *ACM SIGOPS operating systems review*, Vol. 41. ACM, 205–220.
[35] Ganesh Chandra Deka. 2014. A survey of cloud database systems. *IT Professional* 16, 2 (2014), 50–57.
[36] Yves Deswarte, Jean-Jacques Quisquater, and Ayda Saïdane. 2003. Remote integrity checking. In *Working conference on integrity and internal control in information systems*. Springer, 1–11.
[37] Haowen Dong, Chao Zhang, Guoliang Li, and Huanchen Zhang. 2024. Cloud-native databases: A survey. *IEEE Transactions on Knowledge and Data Engineering* 36, 12 (2024), 7772–7791.
[38] M Drake. 2019. A Comparison of NoSQL Database Management Systems and Models. *DigitalOcean, LLC* (2019).
[39] Georgios Drakopoulos, Aikaterini Baroutiadi, and Vasileios Megalooikonomou. 2015. Higher order graph centrality measures for Neo4j. In *2015 6th International Conference on Information, Intelligence, Systems and Applications (IISA)*. IEEE, 1–6.
[40] Elias Dritsas and Maria Trigka. 2025. A Survey on Database Systems in the Big Data Era: Architectures, Performance, and Open Challenges. *IEEE Access* 13 (2025).
[41] Shu Du, Ahamed Khan, Santashil PalChaudhuri, Ansley Post, Amit Kumar Saha, Peter Druschel, David B Johnson, and Rudolf Riedi. 2008. Safari: A self-organizing, hierarchical architecture for scalable ad hoc networking. *Ad Hoc Networks* 6, 4 (2008), 485–507.
[42] Alex Feinberg. 2011. Project voldemort: Reliable distributed storage. In *Proceedings of the 10th IEEE International Conference on Data Engineering*.
[43] Alan Fekete, Dimitrios Liarokapis, Elizabeth O'Neil, Patrick O'Neil, and Dennis Shasha. 2005. Making snapshot isolation serializable. *ACM Transactions on Database Systems (TODS)* 30, 2 (2005), 492–528.
[44] Garth A Gibson and Rodney Van Meter. 2000. Network attached storage architecture. *Commun. ACM* 43, 11 (2000), 37–45.

[45] Hector Gonzalez, Alon Halevy, Christian S Jensen, Anno Langen, Jayant Madhavan, Rebecca Shapley, and Warren Shen. 2010. Google fusion tables: data management, integration and collaboration in the cloud. In *Proceedings of the 1st ACM symposium on Cloud computing*. ACM, 175–180.
[46] Min Gu, Xiangping Li, and Yaoyu Cao. 2014. Optical storage arrays: a perspective for future big data storage. *Light: Science & Applications* 3, 5 (2014), e177.
[47] Henry Hai and Seitaro Sakoda. 2009. SaaS and integration best practices. *Fujitsu Scientific and Technical Journal* 45, 3 (2009), 257–264.
[48] Yin Huai, Ashutosh Chauhan, Alan Gates, Gunther Hagleitner, Eric N Hanson, Owen O'Malley, Jitendra Pandey, Yuan Yuan, Rubao Lee, and Xiaodong Zhang. 2014. Major technical advancements in apache hive. In *Proceedings of the 2014 ACM SIGMOD international conference on Management of data*. ACM, 1235–1246.
[49] Mihai-Dorin Istin, Andreea Visan, Florin Pop, and Valentin Cristea. 2010. Sopsys: Self-organizing decentralized peer-to-peer system based on well balanced multi-way trees. In *P2P, Parallel, Grid, Cloud and Internet Computing (3PGCIC), 2010 International Conference on*. IEEE, 369–374.
[50] Jiehui Ju, Jiyi Wu, Jianqing Fu, Zhijie Lin, and Jianlin Zhang. 2011. A Survey on Cloud Storage. *J. Comput.* 6, 8 (2011), 1764–1771.
[51] Young-Hoon Kim, Sang Hoon Chu, Seong-Woo Kang, Dong-Ho Oh, Yun-Sik Han, and Tae Yeon Hwang. 2006. Servo controller method and apparatus for high tracks per inch hard disk drives using a delay accomodating state estimator. US Patent 7,031,095.
[52] Jayaram Krishnaswamy. 2010. *Microsoft SQL Azure Enterprise Application Development*. Packt Publishing Ltd.
[53] Łukasz Krzywiecki and Mirosław Kutyłowski. 2012. Proof of possession for cloud storage via lagrangian interpolation techniques. In *International Conference on Network and System Security*. Springer, 305–319.
[54] Ajay Kumar and Seema Bawa. 2019. Adjacency Cloud-Oriented Storage Overlay Topology using Self-Organizing M-Way Tree. In *2019 International Conference on Innovative Computing and Communication (ICICC-2019)*. Springer, 76.
[55] Ajay Kumar and Seema Bawa. 2020. DAIS: dynamic access and integration services framework for cloud-oriented storage systems. *Cluster Computing* 23, 4 (2020), 3289–3308.
[56] Yen-Hung Kuo, Yu-Lin Jeng, and Juei-Nan Chen. 2013. A hybrid cloud storage architecture for service operational high availability. In *2013 IEEE 37th Annual Computer Software and Applications Conference Workshops*. IEEE, 487–492.
[57] Leslie Lamport. 1986. On interprocess communication. *Distributed computing* 1, 2 (1986), 86–101.
[58] Walter S Lasecki, Christopher D Miller, Iftekhar Naim, Raja Kushalnagar, Adam Sadilek, Daniel Gildea, and Jeffrey P Bigham. 2017. Scribe: deep integration of human and machine intelligence to caption speech in real time. *Commun. ACM* 60, 9 (2017), 93–100.
[59] Narn-Yih Lee and Yun-Kuan Chang. 2011. Hybrid provable data possession at untrusted stores in cloud computing. In *2011 IEEE 17th International Conference on Parallel and Distributed Systems*. IEEE, 638–645.
[60] David J Lilja. 1993. Cache coherence in large-scale shared-memory multiprocessors: Issues and comparisons. *ACM Computing Surveys (CSUR)* 25, 3 (1993), 303–338.
[61] Weiwei Lin and Deyu Qi. 2010. Research on resource self-organizing model for cloud computing. In *Internet technology and applications, 2010 international conference on*. IEEE, 1–5.
[62] Desheng Liu, Hong Zhu, Chengzhi Xu, Ian Bayley, David Lightfoot, Mark Green, and Peter Marshall. 2016. Cide: An integrated development environment for microservices. In *2016 IEEE International Conference on Services Computing (SCC)*. IEEE, 808–812.
[63] Yang Lu. 2017. Industry 4.0: A survey on technologies, applications and open research issues. *Journal of Industrial Information Integration* 6 (2017), 1–10.
[64] Jie Ma, Tao Chen, Songfeng Wu, Chunyuan Yang, Mingze Bai, Kunxian Shu, Kenli Li, Guoqing Zhang, Zhong Jin, Fuchu He, et al. 2018. iProX: an integrated proteome resource. *Nucleic acids research* 47, D1 (2018), D1211–D1217.
[65] Saif Ur Rehman Malik, Samee U Khan, Sam J Ewen, Nikos Tziritas, Joanna Kolodziej, Albert Y Zomaya, Sajjad A Madani, Nasro Min-Allah, Lizhe Wang, Cheng-Zhong Xu, et al. 2016. Performance analysis of data intensive cloud systems based on data management and replication: a survey. *Distributed and Parallel Databases* 34, 2 (2016), 179–215.
[66] Yaser Mansouri, Adel Nadjaran Toosi, and Rajkumar Buyya. 2017. Data storage management in cloud environments: Taxonomy, survey, and future directions. *ACM Computing Surveys (CSUR)* 50, 6 (2017), 1–51.
[67] Carlo Mastroianni, Michela Meo, and Giuseppe Papuzzo. 2013. Probabilistic consolidation of virtual machines in self-organizing cloud data centers. *IEEE Transactions on Cloud Computing* 1, 2 (2013), 215–228.
[68] Somnath Mazumdar, Daniel Seybold, Kyriakos Kritikos, and Yiannis Verginadis. 2019. A survey on data storage and placement methodologies for cloud-big data ecosystem. *Journal of Big Data* 6, 1 (2019), 1–37.
[69] Rohit Menon. 2014. *Cloudera administration handbook*. Packt Publishing Ltd.
[70] Mike Mesnier, Gregory R Ganger, and Erik Riedel. 2003. Object-based storage. *IEEE Communications Magazine* 41, 8 (2003), 84–90.
[71] Claudia Morgado, Gisele Busichia Baioco, Tania Basso, and Regina Moraes. 2018. A security model for access control in graph-oriented databases. In *2018 IEEE International Conference on Software Quality, Reliability and Security (QRS)*. IEEE, 135–142.
[72] Rajendrani Mukherjee and Pragma Kar. 2017. A comparative review of data warehousing ETL tools with new trends and industry insight. In *2017 IEEE 7th International Advance Computing Conference (IACC)*. IEEE, 943–948.

[73] Rekha Nachiappan, Bahman Javadi, Rodrigo N Calheiros, and Kenan M Matawie. 2017. Cloud storage reliability for big data applications: A state of the art survey. *Journal of Network and Computer Applications* 97 (2017), 35–47.
[74] Daniel Nurmi, Rich Wolski, Chris Grzegorczyk, Graziano Obertelli, Sunil Soman, Lamia Youseff, and Dmitrii Zagorodnov. 2008. Eucalyptus: A technical report on an elastic utility computing architecture linking your programs to useful systems. In *UCSB Technical Report*. Citeseer.
[75] Kunwar Pal, MC Govil, and Mushtaq Ahmed. 2016. Comparative analysis of new hybrid approach for overlay construction in P2P live streaming. In *International Conference on Emerging Research in Computing, Information, Communication and Applications*. Springer, 239–250.
[76] Teykhrib Anton Pavlovich and Teykhrib Genrikh Ivanovich. 2015. ANALYSIS OF CLOUD SERVICES INTEGRATION WITH ENTERPRISE INFORMATION SYSTEMS. *Journal of Theoretical & Applied Information Technology* 82, 2 (2015).
[77] Laura Po, Nikos Bikakis, Federico Desimoni, and George Papastefanatos. 2020. Linked data visualization: techniques, tools, and big data. *Synthesis Lectures on Semantic Web: Theory and Technology* 10, 1 (2020), 1–157.
[78] Evangelos Pournaras, Martijn Warnier, and Frances MT Brazier. 2014. Adaptive self-organization in distributed tree topologies. *International Journal of Distributed Systems and Technologies (IJDST)* 5, 3 (2014), 24–57.
[79] Meikang Qiu, Keke Gai, Bhavani Thuraisingham, Lixin Tao, and Hui Zhao. 2018. Proactive user-centric secure data scheme using attribute-based semantic access controls for mobile clouds in financial industry. *Future Generation Computer Systems* 80 (2018), 421–429.
[80] Pham Tran Anh Quang, Kamal Deep Singh, Abbas Bradai, and Abderrahim Benslimane. 2018. QAAV: Quality of Service-Aware Adaptive Allocation of Virtual Network Functions in Wireless Network. In *2018 IEEE International Conference on Communications (ICC)*. IEEE, 1–6.
[81] P Rabl, D DeMille, John M Doyle, Mikhail D Lukin, RJ Schoelkopf, and P Zoller. 2006. Hybrid quantum processors: molecular ensembles as quantum memory for solid state circuits. *Physical review letters* 97, 3 (2006), 033003.
[82] Daniel J Scales and Kourosh Gharachorloo. 1997. Towards transparent and efficient software distributed shared memory. *ACM SIGOPS Operating Systems Review* 31, 5 (1997), 157–169.
[83] Eric E Schadt, Michael D Linderman, Jon Sorenson, Lawrence Lee, and Garry P Nolan. 2010. Computational solutions to large-scale data management and analysis. *Nature reviews genetics* 11, 9 (2010), 647.
[84] Fred B Schneider. 1990. Implementing fault-tolerant services using the state machine approach: A tutorial. *ACM Computing Surveys (CSUR)* 22, 4 (1990), 299–319.
[85] Pratima Sharma, Rajni Jindal, and Malaya Dutta Borah. 2020. Blockchain technology for cloud storage: A systematic literature review. *ACM Computing Surveys (CSUR)* 53, 4 (2020), 1–32.
[86] Beverley J Shea, Barnaby C Reeves, George Wells, Micere Thuku, Candyce Hamel, Julian Moran, David Moher, Peter Tugwell, Vivian Welch, Elizabeth Kristjansson, et al. 2017. AMSTAR 2: a critical appraisal tool for systematic reviews that include randomised or non-randomised studies of healthcare interventions, or both. *bmj* 358 (2017).
[87] Shiuan-Tzuo Shen and Wen-Guey Tzeng. 2011. Delegable provable data possession for remote data in the clouds. In *International Conference on Information and Communications Security*. Springer, 93–111.
[88] Aisha Siddiqa, Ahmad Karim, and Abdullah Gani. 2017. Big data storage technologies: a survey. *Frontiers of Information Technology & Electronic Engineering* 18, 8 (2017), 1040–1070.
[89] Harcharan Jit Singh and Seema Bawa. 2018. Scalable metadata management techniques for ultra-large distributed storage systems–A systematic review. *ACM Computing Surveys (CSUR)* 51, 4 (2018), 1–37.
[90] Saurabh Singh, Pradip Sharma, Seo Moon, and Jong Park. 2017. EH-GC: an efficient and secure architecture of energy harvesting green cloud infrastructure. *Sustainability* 9, 4 (2017), 673.
[91] Jie Song, Tiantian Li, Xuebing Liu, and Zhiliang Zhu. 2012. Comparing and analyzing the energy efficiency of cloud database and parallel database. In *Advances in Computer Science, Engineering & Applications*. Springer, 989–997.
[92] Andreas Thor and Erhard Rahm. 2011. Cloudfuice: A flexible cloud-based data integration system. In *International Conference on Web Engineering*. Springer, 304–318.
[93] Athena Vakali, Barbara Catania, and Anna Maddalena. 2005. XML data stores: emerging practices. *IEEE Internet Computing* 9, 2 (2005), 62–69.
[94] Mehul Nalin Vora. 2011. Hadoop-HBase for large-scale data. In *Computer science and network technology (ICCSNT), 2011 international conference on*, Vol. 1. IEEE, 601–605.
[95] Boyang Wang, Baochun Li, and Hui Li. 2012. Knox: privacy-preserving auditing for shared data with large groups in the cloud. In *International conference on applied cryptography and network security*. Springer, 507–525.
[96] Huaqun Wang. 2012. Proxy provable data possession in public clouds. *IEEE Transactions on Services Computing* 6, 4 (2012), 551–559.
[97] Qian Wang, Cong Wang, Jin Li, Kui Ren, and Wenjing Lou. 2009. Enabling public verifiability and data dynamics for storage security in cloud computing. In *European symposium on research in computer security*. Springer, 355–370.
[98] Jiyi Wu, Lingdi Ping, Xiaoping Ge, Ya Wang, and Jianqing Fu. 2010. Cloud storage as the infrastructure of cloud computing. In *2010 International Conference on Intelligent Computing and Cognitive Informatics*. IEEE, 380–383.

[99] Jia Xu and Ee-Chien Chang. 2012. Towards efficient proofs of retrievability. In *Proceedings of the 7th ACM symposium on information, computer and communications security*. 79–80.
[100] Tin Tin Yee and Thinn Thu Naing. 2011. Pc-cluster based storage system architecture for cloud storage. *International Journal on Cloud Computing: Services and Architecture(IJCCSA)* 1, 3 (2011), 117–128.
[101] Jiawei Yuan and Shucheng Yu. 2013. Proofs of retrievability with public verifiability and constant communication cost in cloud. In *Proceedings of the 2013 international workshop on Security in cloud computing*. 19–26.
[102] Faheem Zafar, Abid Khan, Saif Ur Rehman Malik, Mansoor Ahmed, Adeel Anjum, Majid Iqbal Khan, Nadeem Javed, Masoom Alam, and Fuzel Jamil. 2017. A survey of cloud computing data integrity schemes: Design challenges, taxonomy and future trends. *Computers & Security* 65 (2017), 29–49.
[103] Yihua Zhang and Marina Blanton. 2013. Efficient dynamic provable possession of remote data via balanced update trees. In *Proceedings of the 8th ACM SIGSAC symposium on Information, computer and communications security*. 183–194.
[104] Qianqian Zhao, Hongwei Kan, Yanwei Wang, Dongdong Su, Kefeng Zhu, and Le Yang. 2021. The deployment of FPGA Based on Network in Ultra-large-scale Data Center. In *Proceedings of the 2021 5th International Conference on Electronic Information Technology and Computer Engineering*. 1050–1055.
[105] Yan Zhu, Hongxin Hu, Gail-Joon Ahn, and Stephen S Yau. 2012. Efficient audit service outsourcing for data integrity in clouds. *Journal of Systems and Software* 85, 5 (2012), 1083–1095.

Appendices

Manuscript Title: High Availability Ultra-Large Cloud-Oriented Storage Systems: A Review, Mind Mapping and Open Research Directions

A Table 9. Comparison of Cloud Oriented Storage System

This table provides salient features of 21 well-known cloud-oriented storage systems focusing on their key characteristics and capabilities.

B Table 10. Self-Organizing storage model in distributed/ cloud environment

This table highlights model name, approach, objectives, complexity and working environment of various existing self-organizing storage models.

C Ultra-Large Storage Management Capabilities

Today, there are many organizations that offer cloud storage services. Most of the providers have used common platforms and approaches. A cloud-oriented database provides the mechanism to store and access large volumes of data in a distributed environment. The features of a cloud-oriented storage system include self-organizing, durability, consistency, storage type, and developer. There are various storage media that are integrated with Network Attached Storage (NAS)[44][21] in cloud computing environment. The Table 11 provides a comparison of various storage mediums based on their objectives, advantages, and disadvantages. Cloud-oriented storage is designed to provide storage as a service with on-demand access. Its main advantage is flexibility, making it ideal for small companies without sufficient storage infrastructure. However, it has issues related to trust and security. Object-based storage stores data in the form of variables and objects. It is easily scalable and secure since the physical location is hard to track. The downside is the complexity in tracking indices. Optical storage is used for storing data in different angles, offering high speed and cost-effectiveness. However, reproducing data on different optical disks can be complex. Solid-state storage is intended for use in various peripheral devices and can store up to 2TBs. It allows for frequent and fast access with minimal start-up time, but it is more expensive for large-scale storage. Local disk storage can store data up to 10TBs and has a high-speed start-up, but it suffers from high latency in reading data. Flash Storage Drive stores data in flash memory with sizes ranging from 8GB to 2TBs. It is faster compared to optical storage, but it has a limited number of write and erase cycles and lacks write protection.

D Table 12. Strengths and weakness of CODS elements

Mainly, this table highlights the strengths and weaknesses of each CODS element.

E Table 13. Application parallelism challenges and mitigation in cloud storage system

F Table 14. Major sources of data

This table exhibits a fascinating visualization of the major sources of data that their organizations can use to forward their business, indicating whether these sources are internal or external, structured or unstructured, their velocity and variety, and whether they leverage APIs or not.

G Insight Analysis of Cloud Oriented Data Storage (CODS) Elements

Data model defines the structure of the data and how it is organized, stored, and accessed. The data model influences the design of other storage elements, such as data consistency and scalability. Different data models

Table (9) Comparison of cloud-oriented storage systems

Cloud Database	Storage capabilities									
	C1	C2	C3	C4	C5	C6	C7	Developer	License Type	Ref.
Apache Hive	Yes	High	Yes	Full	Yes	Yes	Column	ASF	Open source	[48]
BigTable	Yes	High	Yes	Partial	Yes	Yes	Column	Google	CCA License 4.0	[25]
Cassandra	Yes	High	Yes	Partial	-	Yes	Column	ASF	Apache License 2.0	[26][94]
ClearDB	No	High	Yes	-	-	Yes	Column	ClearDB Inc.	Open source	[35]
Cloudant	Yes	High	Yes	-	Yes	-	Key-value	IBM	Proprietary	[17]
CouchDB	Yes	High	Yes	Partial	Yes	Yes	Document	ASF	Open source	[30]
Dryad	Yes	Medium	Yes	Full	Yes	Yes	Column	Microsoft	Open source	[48]
DynamoDB	No	High	Yes	Partial	Yes	Yes	Key-value	Amazon	Open source	[34]
Eucalyptus	-	High	Yes	Partial	Yes	Yes	Document	Eucalyptus Inc.	Proprietary	[74]
FusionTable	No	High	-	-	Yes	-	Column	Google	CCA License 4.0	[45]
Hadoop	Yes	High	-	Partial	Yes	Yes	DFS†	ASF	Open source	[94][33]
Hbase	Yes	Medium	Yes	Partial	Yes	Yes	Column	ASF	Open source	[94]
MongoDB	Yes	High	Yes	Partial	Yes	Yes	Document	MongoDB Inc.	Open source	[29]
Neo4j	Yes	High	Yes	Full	-	Yes	Graph-based	Neo4j, Inc.	Proprietary	[39]
PNUTS	No	Medium	-	-	Yes	-	Column	Yahoo!	Proprietary	[30]
Redis	No	Medium	Yes	Partial	-	Yes	Key-value	Redis Labs	BSD 3-Clause License	[29]
SimpleDB	Yes	High	Yes	Full	-	Yes	Document	Amazon	Proprietary	[35]
SQL Azure	No	Medium	Yes	Full	Yes	Yes	NoSQL	Microsoft	Proprietary	[52]
Voldemort	No	High	Yes	Partial	Yes	Yes	Key-value	LinkedIn	Open source	[42]
Xeround	No	High	Yes	-	No	-	MySQL	Xeround	General Public License	[6]

C1: Self-Organizing; C2: Scalability; C3: High Availability; C4: Replication; C5: Durability; C6: Consistency; C7: Storage Types;

Table (10) Self-Organizing storage model in distributed/ cloud environment

Prevailing Model	Approaches	Objectives	Time Complexity	Environment	Ref.
ACOT	M-Ways Tree Overlay	Improve the self-organizing capability and overall time complexity.	$O(log_k N)$	Cloud Computing	[54]
Adaptive Resource Allocation Model	Ant-based control algorithm	Adaptive resource allocation in dynamic environment	$O(logN^2)$	Distributed Computing	[8]
AETOS	Tree Overlay topology	Improve the performance and enhance runtime estimation strategies.	$O(logN)$	Distributed Computing	[78]
CCPS	M/M/1 queuing model	PC cluster based storage system.	–	Cloud Computing	[100]
Cloud Cell	Fuzzy-logic-based decision framework	Improve the storage capacity and mobility performance.	$O(2^{N+1})$	Cloud Computing	[5]
ecoCloud	Bernoulli trials	Virtual Machine (VM) consolidation Assignment and migration of VMs.	–	Cloud Computing	[67]
Proactive Dynamic Secure Data Scheme (P2DS)	Attribute-based Semantic Access Control Algorithm	Constraints data access and deal with dynamic threats.	–	Cloud Computing	[79]
Safari	Hybrid routing algorithm	Scalable ad hoc network routing and integration services.	$O(logN)$	Ad Hoc Network	[41]
SOPSys	M-ways balanced tree	Reduce the network join and discovery cost.	$O(logN)$	Decentralized Peer-to-Peer Network	[49]
SSA_G	Election algorithm	Resource optimization in dynamic environment and improve the system performance	$O(N^{1/2})$	Cloud Computing	[61]
S-TREE	Supervised learning	Illustrate the data clustering and compression	$O(logN)$	Cluster Computing	[22]
SOS Cloud	Bio-inspired algorithm	Reduce the Service Level Agreement (SLA) violation cost.	–	Cloud Computing	[23]

Table (11) Comparison of different storage mediums

Storage medium	Objectives	Advantages	Disadvantages	Ref.
Cloud-oriented storage	To provide as a service model and on-demand access	More flexible storage system; useful for small company that don't have sufficient storage infrastructure	Lack of trust and security	[98]
Object-based storage	To store data in the form of variables and objects	Easy to access and scale the data; highly secure because physical location cannot track easily	Tracking indices are more complex	[70]
Optical storage	To store data in different angles	High speed and cost effective	More complex ability to reproduce in different optical disk	[46]
Solid-state storage	To store data in various peripheral devices; to store upto 2TBs data	Frequent access; fast movement; start-up time is very less	More expensive for large-scale storage	[81]
Local disk storage	To store data upto 10 TBs	high speed start-up; storage cost per bit	High latency time for reading data	[51]
Flash Storage Drive	It stores data in flash memory with integrated USB interface. Size of this storage device from 8GBs to 2TBs	It is faster as compare to optical storage	Limited number of write and erase cycles; no facility to write protection	[27]

(relational, NoSQL, etc.) may have varying requirements for consistency and scalability. For example, relational data models typically prioritize data consistency and provide strong guarantees for maintaining the integrity of the data. On the other hand, NoSQL data models often prioritize scalability and can handle large amounts of data with high throughput. Therefore, choosing the appropriate data model is crucial to ensuring that the storage system meets the specific needs of an application or organization.

Data consistency ensures that data remains accurate and reliable across distributed systems and multiple replicas. It is closely related to data replication and transaction management. Maintaining consistency becomes more challenging in distributed and scalable cloud environments. In these environments, data is often replicated across multiple nodes to ensure fault tolerance and high availability. However, the process of synchronizing data across these replicas can introduce latency and potential inconsistencies. To address this challenge, various techniques, such as distributed consensus algorithms and conflict resolution mechanisms, are employed to maintain data consistency in distributed systems.

Data replication involves creating and maintaining copies of data in multiple locations for fault tolerance, load balancing, and enhanced performance. Consistent replication strategies are crucial for ensuring data integrity, and scalability may involve distributing data across multiple nodes or regions. By distributing data across multiple nodes or regions, scalability can be achieved as the workload is distributed among different resources. This allows for better utilization of resources and improved performance. Additionally, consistent replication strategies ensure that all copies of data are synchronized and up-to-date, preventing any inconsistencies or conflicts that may arise due to concurrent updates or failures in the system.

Data scalability enables systems to handle increased load and data volume by distributing data and processing it across multiple resources. Scalability is closely related to data replication and consistency. Effective scalability often requires careful consideration of how data is partitioned, replicated, and distributed across the cloud infrastructure. It ensures that as the system grows, it can continue to handle the increased workload without sacrificing performance or data integrity. By distributing data and processing it across multiple resources,

Table (12) Strengths and weakness of CODS elements

Element	Strengths	Weakness
Data Model	A well-designed data model facilitates efficient data retrieval and storage. It supports complex queries and analytics by organizing data logically. Flexibility in adapting to various types of data, such as structured, semi-structured, and unstructured.	Poorly designed data models can lead to inefficiencies and increased latency. Rigid data models may struggle with evolving data requirements, necessitating costly migrations or re-engineering.
Data Consistency	It helps to prevent conflicts and ensures that operations such as read and write are reliable. Essential for maintaining data integrity in distributed environments. Supports compliance with regulatory requirements that demand data accuracy.	High levels of consistency can impact system performance, particularly in geographically dispersed systems. Achieving consistency in real-time can be challenging and may lead to trade-offs in availability or latency.
Data Replication	It provides load balancing by allowing access to data from multiple locations. Supports high availability systems by ensuring that data is always accessible, even in the event of a failure.	Increases storage costs due to the need to store multiple copies of data. May introduce latency due to the time required for data synchronization.
Data Scalability	It supports the growth of applications by enabling seamless expansion of storage capacity. Allows for the dynamic allocation of resources based on demand, improving cost efficiency. Enables businesses to scale their operations without significant infrastructure changes.	Scaling large volumes of data can introduce challenges in managing performance and latency. It requires complex architectures and advanced management tools, leading to increased operational complexity.
Data Integration	Facilitates comprehensive analysis by bringing together data from multiple sources. Supports business intelligence and decision-making processes by providing a holistic view of data. Enables interoperability between different systems and applications.	Can be complex and time-consuming to implement, especially with heterogeneous data sources. Integrating real-time data from various sources can be challenging and may require sophisticated middleware.
Data Virtualization	Simplifies access to data by allowing users to interact with it without knowing where it is physically stored. Reduces the need for data replication, thus lowering storage costs. Speeds up data retrieval by providing real-time access to integrated data.	Can introduce performance overhead due to the abstraction layer. May not be suitable for all types of data or applications, particularly those requiring high-performance access.
Data Transaction	Ensures the integrity of operations, especially in multi-user environments.Provides a mechanism to roll back changes in the event of an error, maintaining data reliability. Critical for financial and other sensitive applications where accuracy is paramount.	Transaction management can introduce performance bottlenecks, especially in high-volume systems. Managing distributed transactions across different systems can be complex and error-prone.

scalability allows for efficient utilization of resources and improved response times. However, achieving effective scalability requires careful planning and design to ensure that data is partitioned and replicated in a way that maximizes performance and minimizes bottlenecks.

Table (13) Application parallelism challenges and mitigation in cloud storage system

Parallelism Factor	Challenges	Mitigation
Data Distribution and Partitioning	Distributing data across multiple storage nodes in a balanced way can be challenging. Uneven data distribution may lead to hotspots, where certain nodes are overloaded, causing performance bottlenecks.	Efficient data partitioning and distribution strategies are required to ensure a balanced workload across storage nodes. Techniques such as sharding and consistent hashing can help achieve better distribution.
Consistency and Coherency	Maintaining consistency and coherency in a parallel environment can be complex, especially when multiple nodes are accessing and modifying data concurrently.	Implementing appropriate consistency models and synchronization mechanisms is essential. Techniques such as distributed locks, versioning, or using eventual consistency models can help manage data integrity.
Network Latency and Bandwidth	In a distributed environment, communication between storage nodes introduces network latency and consumes bandwidth, affecting the overall performance.	Optimizing network communication and reducing the amount of data transferred can help mitigate latency and bandwidth challenges. Techniques like data compression and minimizing unnecessary data transfers are essential.
Fault Tolerance	Cloud storage systems must be resilient to hardware failures, software errors, or network issues.	Implementing fault-tolerant mechanisms, such as data replication, erasure coding, and distributed redundancy, is crucial to ensure the system remains available and reliable even in the face of failures.
Scalability	Ensuring that the storage system can scale horizontally to accommodate growing amounts of data and increased demand.	Designing the storage system with scalability in mind, such as employing a distributed file system or object storage architecture, enables seamless expansion as data volume and user load increase.

Data integration ensures the combining and unifying of data from different sources to provide a unified view for analysis and decision-making. Integration may involve dealing with diverse data models and sources, impacting data consistency and scalability. Data integration solutions need to be designed to work seamlessly with the chosen data model and replication techniques. Additionally, data integration solutions should also consider the potential impact on data consistency and scalability when dealing with diverse data models and sources. By carefully designing the integration process to work seamlessly with the chosen data model and replication techniques, organizations can ensure that their data remains consistent and scalable as they grow.

Data virtualization provides a layer of abstraction over distributed and heterogeneous data sources, making it easier to access and manage data. Virtualization can enhance data integration by presenting a unified view of data. It also plays a role in scalability by facilitating access to distributed data without exposing the underlying complexity. In addition, data virtualization can improve the agility of organizations by allowing them to quickly adapt and respond to changing business needs. It enables users to access and analyze data in real-time, regardless of its location or format. This flexibility can greatly enhance decision-making processes and drive innovation within the organization.

Table (14) Major sources of data

Source	Velocity	Content-based	Description
Transaction Data	Average	Structured	Associated with banking operations, financial activities, and business data that are of strategic value to an organization. Generally, non-volatile and transnational in nature
Sensors Data	High	Structured and Unstructured	Sensor data is collected by devices that measure physical properties in the environment, such as temperature, pressure, humidity, and light, providing valuable insights into the surrounding conditions and is widely used in various industries and technologies.
Machine Data	High	Structure and Unstructured	Machine-generated data plays a crucial role in various fields, including IT operations, cybersecurity, industrial monitoring, and data analytics. Analyzing this data provides valuable insights for optimizing performance, identifying issues, and making informed decisions in diverse domains.
Web and Social Media	High	Unstructured	It is user generated data such as text, images, sound and videos etc. Social media data is information collected from social media networks that demonstrates how users share, view, or engage with business analytic. Raw social data includes comments, views, shares, and likes.
Journalism and News Media	High	Semi-structured and Unstructured	User generated data such as text, images, sound and videos etc. Statistical data for various events and sports.
Medical Data	Average	Structured, Semi-structured and Unstructured	Patient details, diagnostic reports, imagine data like Ultra sound, MRI, CT-Scan etc.

Data transactions ensure the atomicity, consistency, isolation, and durability (ACID) properties of transactions, maintaining data integrity. Transactions are fundamental for ensuring data consistency, especially in distributed environments. They interact closely with data replication to ensure that changes are propagated reliably across the system. Transactions also provide a level of security by allowing for rollbacks in case of errors or failures, ensuring that data remains consistent and accurate. Additionally, the use of transactions can improve performance by reducing the amount of time spent on data synchronization and conflict resolution.

Received 8 December 2022; revised 28 January 2026; accepted 2 July 2026