International Journal of Science and Research Archive

eISSN: 2582-8185

Cross Ref DOI: 10.30574/ijsra

Journal homepage: https://ijsra.net/

(RESEARCH ARTICLE)

Check for updates

Data synchronisation strategies for distributed web applications using MySQL, MongoDB and AWS Aurora

Sumit Pillai * and Vasudhar Sai Thokala

Independent Researcher, USA.

International Journal of Science and Research Archive, 2023, 09(01), 779-793

Publication history: Received on 02 April 2023; revised on 17 May 2023; accepted on 28 June 2023

Article DOI: https://doi.org/10.30574/ijsra.2023.9.1.0349

Abstract

A very important part that must be addressed when developing web applications with a distributed approach is data synchronisation. The data can be used across multiple platforms, services, and business locations and remains constant in value. This review paper focuses on the various synchronisation techniques used with MySQL, MongoDB, and AWS Aurora databases which are some of the most popular ones in web-oriented architectures of the present day. It starts with explaining the three basic types of databases and their architectural models since these two factors play a role in the synchronisation process and different synchronisation techniques. It considers the trade-off between eventual and strong consistency models and their impact on application performance and user experience. Other hybrid synchronisation strategies that combine several databases are also considered for their potential value in improving data accuracy and resilience. It suggests development practices that would be useful to website developers and designers focused on achieving integrated data synchronisation across platforms and adding, the debate of how best practices in data management can be advanced in the context of global networks.

Keywords: Distributed database; Data Synchronization; Distributed web apps; MySQL; MongoDB; AWS Aurora

1. Introduction

With their great reliance on synchronisation and orchestration during benchmark runs, databases provide unique issues. This is particularly true in micro-service-oriented technologies and dynamic business models like DBaaS [1][2]. Database benchmark setup or, in the case of DB as a Service, the possibility of having to study multitenancy concerns, both of which need synchronisation. Interference effects will be produced when separate database instances execute simultaneously on the same node or cluster [3]. For this reason, it is important to include and enforce appropriate synchronisation points to guarantee that all deployed DBs start their measurement phase simultaneously [4].

Database synchronisation ties into the evolution of database technologies over the past three decades[5]. Numerous database systems have been put out for various applications and dataset sizes. Big data presents unique issues in terms of both diversity and size, which are clearly beyond the capabilities of traditional relational database systems [6]. The MYSQL database is quickly becoming the go-to for handling large data issues because of its simple API, eventual consistency, schema-free nature, and ability to accommodate massive amounts of data[7].

While relational databases like MySQL and Amazon Aurora offer foundational solutions in structured data management, they have adapted to integrate features for modern demands[8][9]. MySQL is document-based, yet it is organised like a relational database, with tables and columns. MySQL stores information as documents using key-value pairs; it is not a relational database [10]. Database developers may use document-based MySQL to manage SQL relational tables and JSON collections without schema. Users have the most leeway when creating NoSQL and conventional SQL relational

* Corresponding author: Sumit Pillai

applications using document-based MySQL [11]. Document database applications that do not rely on schemas; hence, a distinct NoSQL document database is unnecessary [12].

Regarding cloud computing, one of the most unique aspects of Amazon Aurora[13], a relational database management system inside AWS[14], that sets it apart from other relational databases is its ability to offload redo processing to a multi-tenant scale-out storage service that was specifically designed for Aurora. This way, network traffic is reduced, crash recovery and checkpoints are avoided, data loss is prevented during failovers to replicas, and database participation is avoided during fault-tolerant storage's healing process. Distributed consensus techniques for commits, reads, replication and membership modifications would be used in traditional distributed storage implementations, increasing the underlying storage cost [15].

The following paper is organised as: Section II provides the Background of Web Applications for Data Distributed, Section III gives the Database Architectures for Distributed Systems, Sections IV, V and VI discuss the Synchronization Strategies for MySQL, MongoDB and Amazon Aurora, Section VII provide the Literature Review and Section VIII give the Conclusion and Future Work.

2. Background of Web Applications for Data Distributed

A database is just a collection of data that has been organised somehow. Database management systems manage all aspects of databases, including data, transactions, issues, and more. The first database management system was a relational database that used SQL. The most recent fad in business is the NoSQL database, which is not relational [16].

There is a growing demand to transition from SQL-Like to NoSQL DB Sable to effectively handle massive amounts of diverse data across various industries and fields, such as smart cities, industry 4.0, marketing, transportation, healthcare, genomics, and more, due to the arrival of future Internet applications[17][18]. As a result, a great deal of research and literature has been developed to compare and contrast SQL-like and NoSQL database systems [19]. Distributed applications face a data management challenge due to the ever-increasing volume of data generated by modern Web and software services. Because of this, non-distributed RDBMS is not widely used. Figure 1 displays the CAP theorem that Eric Brewer proposed to meet this restriction. The following three qualities cannot be concurrently provided by a distributed storage system, according to the claim [20]:

- **C as Consistency:** Every node in a distributed storage system has to see the effects of any changes made to the system.
- **A as Availability:** There is no downtime in the storage system's ability to fulfil requests.
- **P as Partition-tolerance:** If network issues arise, the storage system will continue functioning and reply to enquiries.

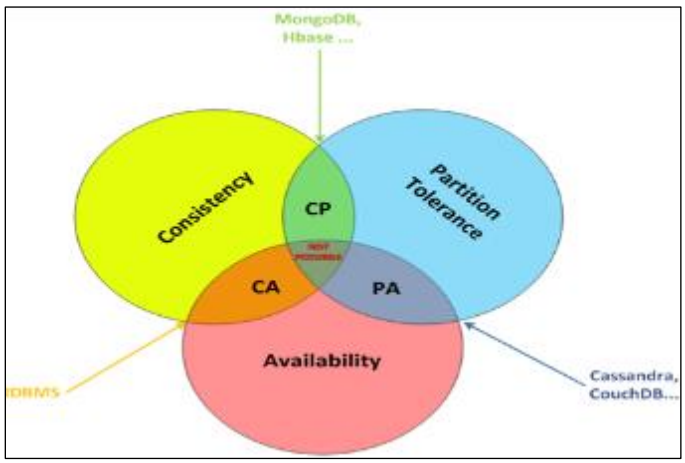

Figure 1 CAP Principle

All databases have had to adapt to the new reality of massive data produced by web-based applications. Cloud databases seem to be an excellent choice for managing this kind of data. In addition, not every business can afford to manage its own databases via the costly infrastructure of a data centre. A new age of databases is dawning with the meteoric rise of cloud databases. Even though cloud databases do not meet ACID standards, they may manage heavy loads from online

applications[21], that do not need such assurances. Features and ideas like as schema-free databases, easy application programming interfaces, eventual/timeline consistency, scalability, synchronous/asynchronous replication, etc., are shared by both [22]. Each database, however, has its specific API, query interface, data model, and database operations. For the sake of their improved development, these ideas should be standardised. Relational databases primarily facilitated the management of transactional data. In subsequent years, data mining applications saw relational database analytics features introduced by market players like Oracle and IBM[23][24]. Many databases, including object-oriented databases and column databases, entered the market in the meantime. However, the relational databases were too strong for them to overcome. Then, the web 2.0 apps and the Internet revolution generated vast amounts of unstructured and sparse data. Using RDBMS to manage large, sparse data collections with poorly specified schemas is impossible. As Cloud databases, NoSQL databases were defined by the need to store and handle such massive amounts of data[25]. The RDBMS and NoSQL database comparison are shown in Table 1.

Table 1 Comparison of RDBMS and NoSQL databases

RDBMS	**NoSQL Databases**
Data within a database is treated as a "whole"	Each entity is considered an independent unit of data and can be freely moved from one machine to the other
RDBMS supports centrally managed architecture.	They follow distributed architecture.
They are statically provisioned.	They are dynamically provisioned.
It is difficult to scale them.	They are easily scalable.
They provide SQL to query data	They use API to query data (not feature-rich as SQL).
ACID (Atomicity, Consistency, Isolation and Durability) Compliant; DBMS maintains Consistency.	Follow BASE (Basically Available, Soft state, eventually consistent); The user accesses are guaranteed only at a single-key level.
They support online Transaction Processing applications.	They support Web 2.0 applications.
ORACLE, MySQL, SQL Server etc. are popular RDBMS.	Amazon Simple DB, Yahoo's PNUTS, CouchDB etc. are popular NoSQL Databases.

2.1. Selection of Database for Distributed Systems

2.1.1. SQL Database

SQL Database stores data according to the relational data model. Data is organised in a tabular fashion using rows and columns in this paradigm. There is a way to connect related tables. Database management systems such as MySQL, Oracle, SQL Server, and others are accessible.

2.1.2. NoSQL Databases

The non-relational data paradigm is followed by NoSQL. Data in many formats, including documents and graphs, may be stored schema-free using a non-relational paradigm. The capabilities of NoSQL make it a viable option for data storage, including horizontal scalability, schema-less storage, and support for unstructured data [26].

Among various databases, selecting the appropriate one is essential from the perspective of distributed applications. The next section discusses the variations among different database types [27]:

2.1.3. Scalability

SQL databases provide for vertical scalability, but NoSQL databases may scale horizontally. "Vertical scalability" means that a single node may enhance its performance by directly adding more memory or processors.

- MySQL generally accommodates vertical scaling, resource addition of the same node which could allow moderate growth but has constraints within distributed environments. With horizontal scaling,
- MongoDB can add nodes to share the load, hence making it suited for dynamic IoT applications that are targeted to grow in the future without heavy financial investments at the start.

- AWS Aurora which is also compatible with MySQL supports horizontal scaling and allows automatic adding of read replicas in different Availability zones taking advantage of both ordered principles and distributed dynamics.

2.1.4. Data Retrieval

A faster data retrieval function will be necessary when a user needs to get data from a database for further processing.

- MySQL employs JOIN to code different tables, which complicates the task of dealing with intensive queries.
- MongoDB, related entities are kept and retrieved within documents, which are faster and do not utilise joins.
- AWS Aurora, although being relational on the level of MySQL, provides optimised cache layers and enhanced speed of replication, contributing to faster data retrieval suitable for highly demanding distributed environments.

2.1.5. System Maturity

Most of the problems have been resolved since SQL is an experienced technology. The SQL includes security features such as authentication, data confidentiality, and integrity.

- MySQL has been around for quite some time and is quite mature and secure, offering great features like authentication and data integrity, which makes it well-suited for sensitive applications as well.
- MongoDB, on the other hand, is constantly evolving but is currently missing some of these conventional security measures and could need further protection regarding sensitive information.
- AWS Aurora is a mix of maturity offered by MySQL and modern security capabilities from AWS, such as encryption and IAM integration, making it excellent suited for scalable modern distributed systems.

3. Database Architectures for Distributed Systems

The technology known as the Distributed Database (DDB) System is a hybrid of two very different techniques for data processing: databases and their networking. Many variables are involved in this method [28]. Data allocation, fragmentation, and duplication are among the most prevalent. The term "distributed database" describes a system in which many databases are physically located at different places and linked together over a computer network. Distributed database systems aim to fool end users, terminal emulators, or user terminals into thinking the system is centralised.

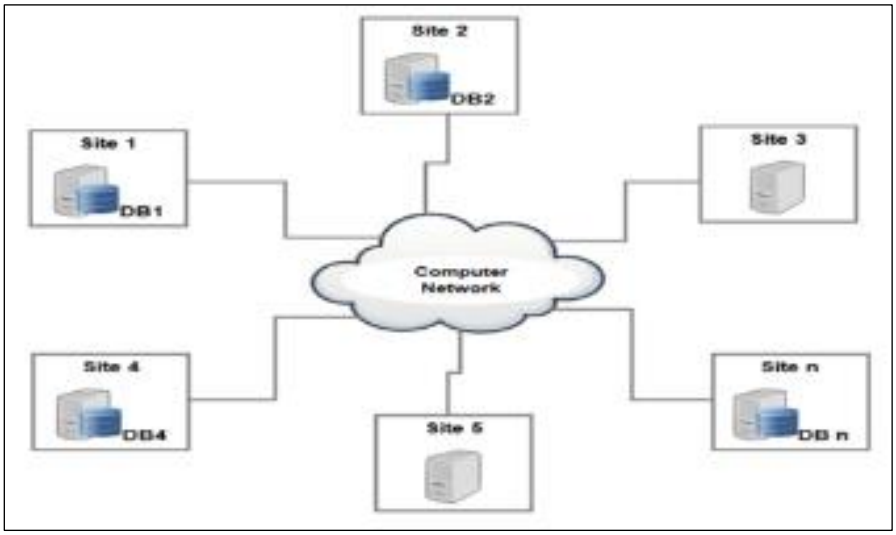

Figure 2 Distributed Database Architecture

Database management systems that are distributed are responsible for overseeing the operations of databases that are distributed. The Distributed Database design is shown in Figure 2.

3.1. Types of Distributed Databases applications

In general, there are two categories of Distributed Database Systems[29]:

- Homogeneous Distributed Database Systems: DBMS store and retrieve data from several servers that share the same hardware, operating system, and database schema as the physical database.
- Heterogeneous Distributed Database Systems: Different sites within these systems may use various DBMSs, operating systems, and hardware, all based on distinct physical database schemas. You may access data stored in various places since these database systems are continually networked.

3.2. Advantages of Distributed Databases

Distributed databases have several benefits, some of which are:

- Robust– The operation of other parts of the organisation will not be impeded by a problem in one section.
- Security- Access may be limited so that only authorised staff members can access their respective databases.
- The bandwidth cost is decreased as a result of less network traffic.
- If the business network is momentarily down, the local database can still access the data.
- High Performance– There is no network bottleneck since queries and updates are mostly performed locally.
- Rather than affecting the whole organisation, mistakes in distributed systems are more easily contained at the local level

3.3. Requirements of Future Distributed Systems

- Several critical needs of distributed computing systems must be met by any environment that facilitates the creation, deployment, and administration of distributed applications. This environment must also be service-based and built on architecture. m. Some general criteria that think are crucial are as follows. While they have not started explicitly addressing all of these criteria, they have influenced our study. Here are a few examples [30].
- Despite concerns about price and performance, other factors, like transparency and reusability, will be crucial.
- The distributed environment's services must facilitate creating and maintaining distributed applications as sets of interdependent peer components [31]. Development tools and languages will be necessary to facilitate the synthesis of components to create complicated applications tailored to individual users.
- Current programs and services will need to adapt instead of being rebuilt to facilitate the transfer of legacy apps to a completely distributed environment [32]. Current apps and services need to be able to communicate with dispersed apps and services.
- Future technologies, including high-speed networks, will need adaptable dispersed settings. Administrators of computer systems shouldn't have to settle for the "lowest common denominator" when it comes to the services and technology available to them.
- for security and privacy. Authenticated access to data and computer resources is a top priority for most companies. Controlling who may access what in centralised settings is easier with clear authorisation and access policies.

4. Synchronization Strategies for MySQL

The data stored in the same network is distributed on different nodes in the distributed database system. How to synchronise the database information of each node is the key problem to be solved. Unless all the nodes obtain the synchronised information in the database, distributed database may sustain integrity[33]. The platform would have a negative impact if the data is not synchronised. So that, data synchronisation is the thing must firstly solve in the distributed database system. Various strategies are discussed below for data synchronisation in distributed web applications[34]:

4.1. Single Master and Multi-Master replication

The replication technique ensures that data is consistent across databases by duplicating and distributing objects and items from one database to another [35]. Data consistency may be automatically achieved via replication, a coping mechanism for databases and a component of object database management on a computer network. This replication approach allows data transmission to remote locations over internal and local network connections[36]. Database processing across several servers, online transaction processing, and DSS (Decision Support System) are only a few examples of how replication facilitates application performance and dissemination of biological data according to its intended usage. Database reliability may be enhanced by using this approach to distribute data to many servers[37].

4.1.1. Single Master Replicated Technique

This method uses one computer as the master and another as the slave. This action will include the server computer accessing and writing to the database [38]. Comparatively, the slave computer will just access the database for reading purposes. Automatic replication of changes made to the master to the slave is a key feature of this system. Master database will not be affected by changes made to slave database[39].

4.1.2. Multi-Master Replication

Two computers are involved in this process; one acts as the master server, and the other acts in the same capacity. Every computer will be able to access and update the database in the next year. Changing anything on controller server will affect master server two as well. The same holds for the master server database; any changes made to that database will instantly reflect on the other database. Both master 1 and 2 will therefore be able to update and contribute to the distributed database. Figure 3 shows the Multi-master Replication.

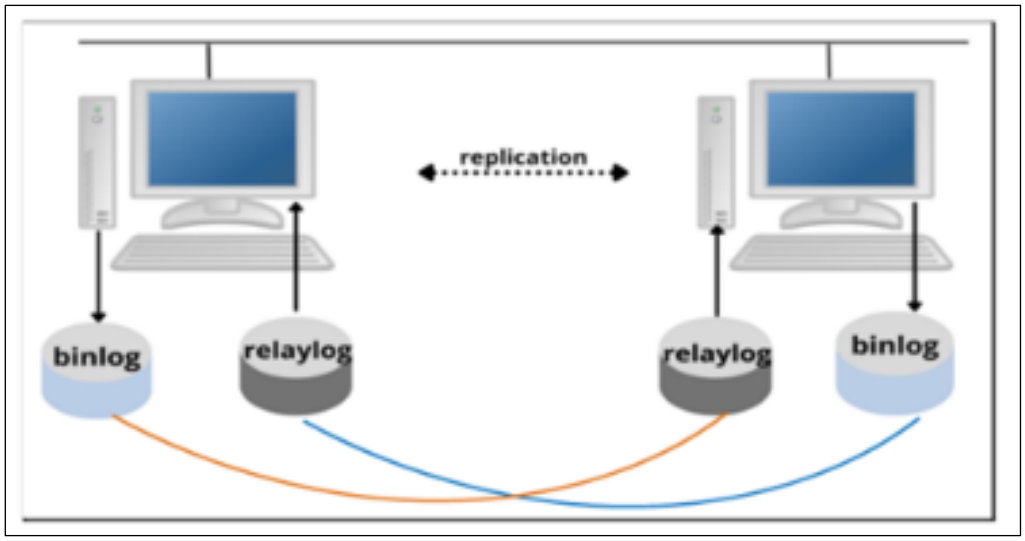

Figure 3 Multi-master Replication

4.2. MySQL Cluster

Figure 4 depicts MySQL Cluster, a technique that offers auto-sharding and shared-nothing clustering for the database management system. There isn't a weak link in MySQL Cluster. The architecture prioritises high availability, low latency, and throughput with the capacity to scale almost linearly. Each data node in MySQL Cluster may accept write operations, and the system is designed to work as a distributed multi-master database, so any changes made by an application or SQL node are immediately accessible by all other nodes in the cluster[40].

Read/write demanding workloads, accessible via SQL and NoSQL interfaces, may be served by MySQL Cluster's horizontal scaling on commodity hardware with auto-sharding. The ability to add nodes to a live cluster with minimal downtime, automated data partitioning with load balancing, and support for both in-memory and disk-based data enable linear database scalability to manage the most unexpected workloads. To ensure the system can continue to function if a node has an issue, such as a network outage, it comprises several nodes dispersed between devices[41].

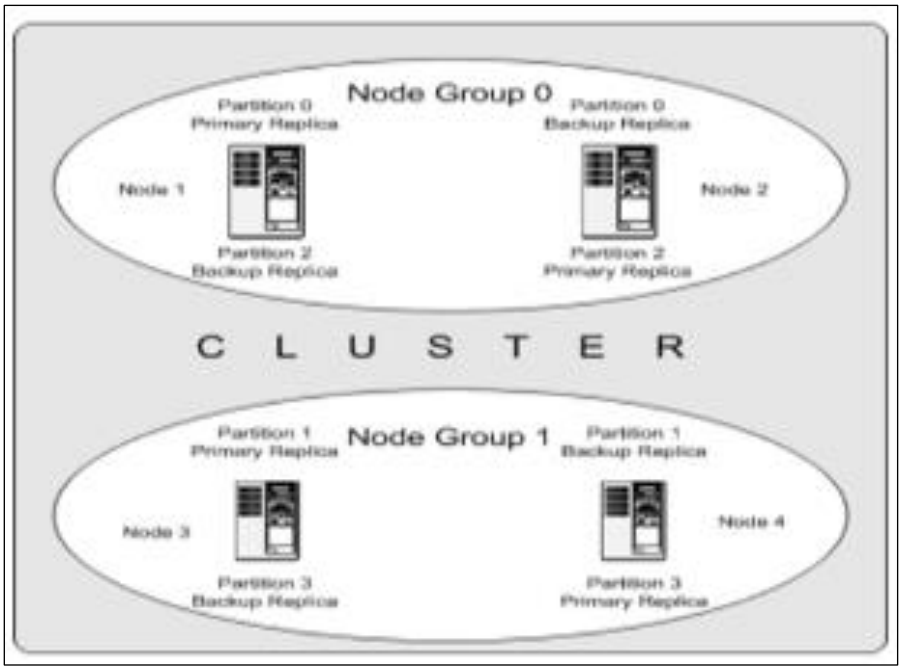

Figure 4 MySQL Cluster Environment

5. Synchronization Strategies for MongoDB

MongoDB is a free and open-source database that focuses on documents. The code is written in C++. In 2007, the software business 10gen created it to serve as a service for other programs. The next year, in 2009, the business shifted its focus to open-source development. Many websites have been using MongoDB as their backend software since then. Although it differs from RDBMS, the document database model uses some structures comparable to RDBMS [42]. Figure 5 displays the MongoDB Sharding.

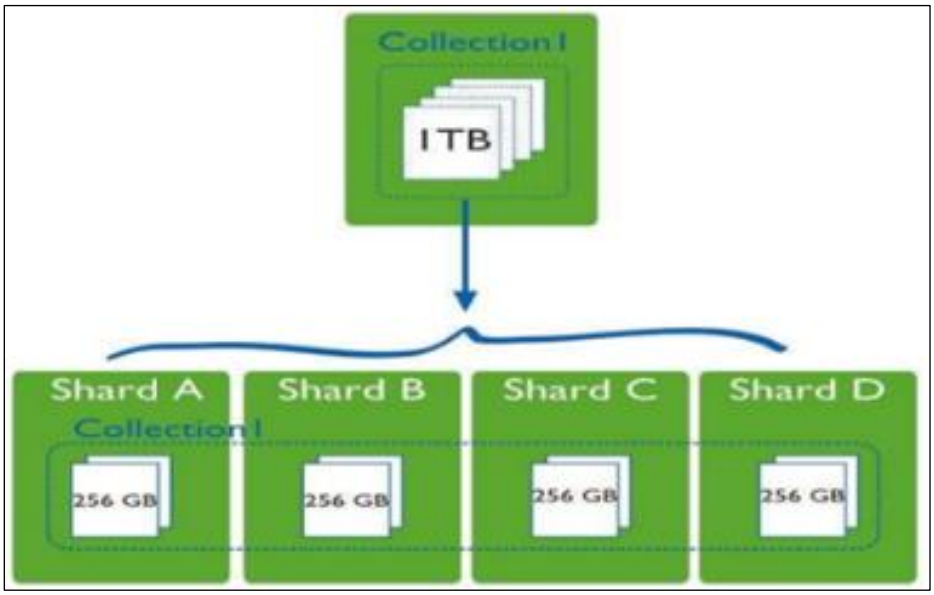

Figure 5 Sharding of MongoDB

5.1. Types of MongoDB

Relational, Graph, Key, Queue, FTS, Reduce, and other database types are "as opposed to" this. Topics covered include indexing possibilities, data organisation about query patterns, coding for polymorphic objects, and client-side manual joins. Similarly, MongoDB's Replica Sets and Sharding feature emphasise the use of several servers

5.1.1. Sharding in MongoDB

The Query Routers or Mongos and Config Server are the primary tools for implementing sharding in MongoDB. Auto-sharding and shard balancing are features of MongoDB. By using Query routers and Config servers, Shard Clusters ensure the smooth operation of all processes.

5.1.2. Query Routers

The application server uses this procedure to determine which shard should execute a query. It then provides the result after processing the query to the relevant shard. There might be more than one query router on a single shard cluster. If necessary, it spreads the workload of client-requested queries. Query routers are responsible for processing all queries.

5.1.3. Config Server

Shard metadata, also known as mapping information, is stored by the configuration server. With this data, a query router may direct the query to the correct shard for processing. A 3 Config server has to be accessible to make changes to or build a new shared cluster. Information from sharded clusters becomes read-only if the number is less than 3. For a single-sharded cluster to move or divide shards, at least three configuration servers must be operational. A technique called Balancer is also included in Sharding [43].

5.1.4. Change Data Capture (CDC)

Data must be synchronised whenever it changes. The data synchronisation component of the web service adapter module is responsible for achieving this function: The source database and data synchronisation:

- The capture component obtains the incremental update log table and sends it to the source database.;
- While synchronising data, either manually or by event-driven transmission, use the Purpose Insert Book method of the Purpose Insert Book class to ensure it is being saved in the central database. This approach is a rational procedure for synchronising data. Used in synchronous are the synchronised data table and the source log table.
- Notify the central database of the successful synchronisation and then remove the incremental synchronisation log table.
- The capture component sends the incremental update log table to the central database.
- In addition to calling the source insert book method in the source database, data synchronisation may be accomplished by event-driven transmission. The procedure follows a rational pattern of data synchronisation. Database type, purpose log table, synchronised data table, and table information will all be used synchronously;

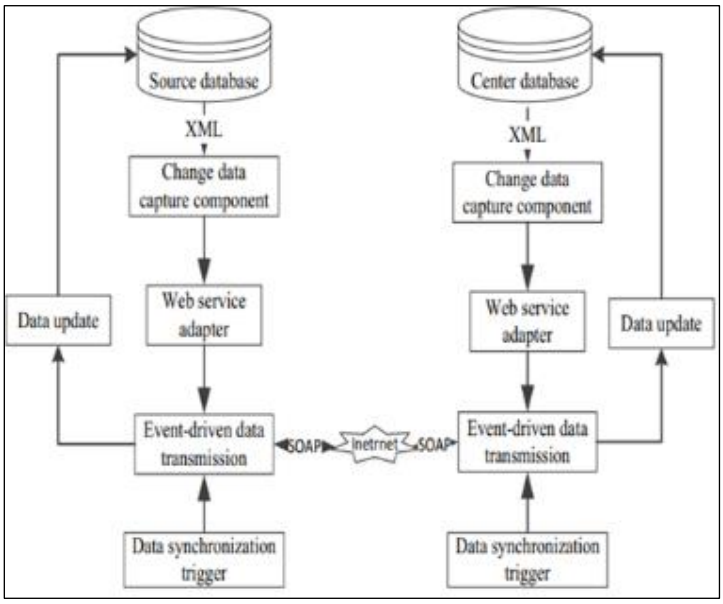

Figure 6 Change Data capture using Web Technologies

Figure 6 shows the synchronisation sequence; after a successful synchronisation, you should get a notification from the source database and be able to remove the incremental synchronisation log table that corresponds to it[44]:

5.2. Applications of MongoDB

The following content management system uses MongoDB as its back-end, as shown below [45]:

- Node.js and MongoDB were used to construct this system.
- Node.js and MongoDB are also used in the construction of this system.
- The front end is built using PHP.
- It's a free and open-source Rails content management system.
- Forward is an open-source e-commerce platform. Strong templates are supported.
- The back-end database is MongoDB.
- To keep track of its billions of photographs, it used MongoDB.
- All of the information is stored and processed automatically using MongoDB.
- For search recommendations, it utilises MongoDB

6. Synchronization Strategies for Amazon Aurora

Amazon Aurora is a relational database service that excels in cloud settings. It is compatible with MySQL and PostgreSQL and offers great performance. Companies may concurrently enable read/write activities across multiple database instances by setting Aurora up as a multi-master cluster [46][47]. The distributed design of this system improves speed and makes it more resilient by reducing the effect of any node failures. By using the Aurora API, crucial components like as failover, scaling, and load balancing can be easily managed inside the multi-master cluster. This guarantees that the cluster will remain available and scalable regardless of load fluctuations or unforeseen occurrences.

6.1. Aurora Replicas

The design of the Aurora replicas is seen in Figure 7. Aurora allows for the mounting of shared storage volumes by a single writer and up to fifteen read replicas. So, read replicas don't cost anything extra regarding storage consumption or disc write operations. All read replicas get the log stream the writer sends to the storage nodes to keep latency to a minimum. The database processes each log record individually as it consumes this stream of logs in the reader.

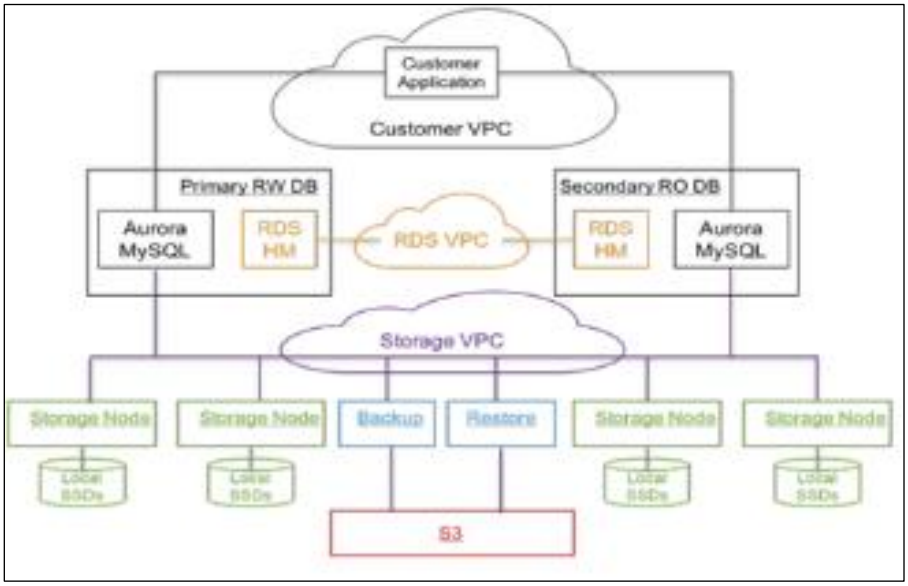

Figure 7 Aurora Architecture

A page in the reader's buffer cache may have the specified redo operation applied to it using the log applicator if the log record specifies that it should be executed. Failing that, the log entry is just discarded. Regarding the writer's point of view, which recognises user contributions independently of the replica, it is important to note that replicas consume log data asynchronously.

Two crucial principles are followed by the replica while applying log records.

- Log records will only be used if their LSN is equal to or lower than the VDL.
- To ensure the replica has the same view of all objects in the database, the log entries that are part of a single mini-transaction are applied atomically in its cache. Each copy follows the author's work by a little margin in practice.

6.2. Amazon Aurora Global Database

Applications built for AWS's globally dispersed cloud may use Amazon Aurora Global Database. Its capacity to fail over to another AWS region is how it offers high availability and database resilience. It comprises of one main and up to five subsidiary regions in a global database cluster, enabling a database to span numerous regions. The primary area is capable of reading and writing, whereas the second zone is limited to reading. Figure 8 depicts the architecture of the Amazon Aurora Global Database.

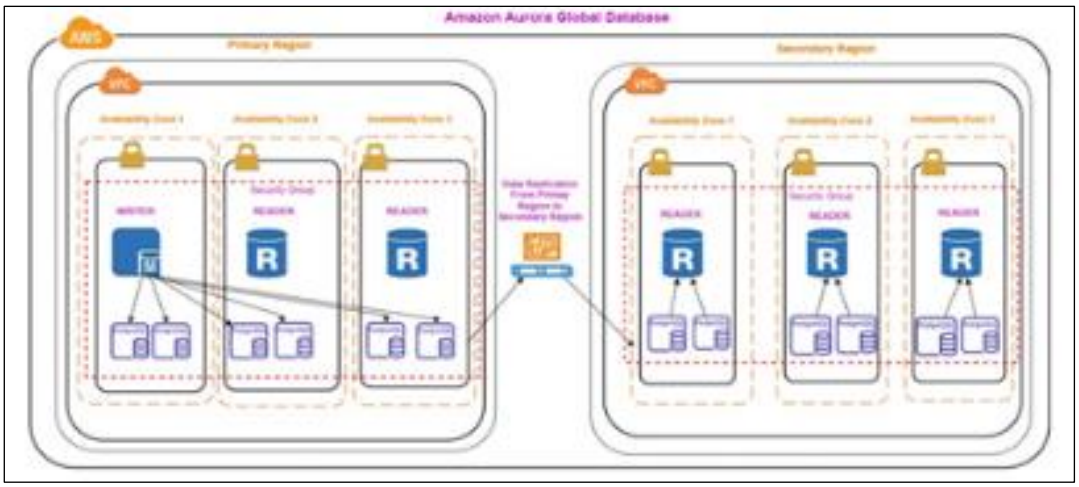

Figure 8 Amazon Aurora Global Database architecture

AWS makes this capability possible by turning on writer endpoints in the main region and turning them off in the secondary regions. Additionally, Aurora duplicates data in less than a second from main to secondary locations.

6.3. Applications of AWS Aurora

Here's a breakdown of applications of synchronisation strategies for AWS Aurora, with a focus on managing distributed data effectively

- Suitable for globally distributed applications like social media, where users in various regions need fast access to the latest content.
- Suitable for globally distributed applications like social media, where users in various regions need fast access to the latest content.
- E-commerce sites and content-heavy applications with high read volumes can balance loads across multiple read replicas, ensuring responsiveness.
- Critical data management for industries like banking or government services, where data integrity and recoverability are paramount.
- Real-time analytics for IoT platforms that collect and process large sensor data require immediate data synchronisation.
- Financial systems where strict data consistency is critical, such as in banking transactions or stock trading.

7. Literature Review

This section provides related work, a systematic review of data synchronisation strategies for distributed web applications using MySQL, MongoDB, and AWS Aurora. Also, Table 2 provides the summary of these literature reviews discussed below:

This Study, Ezéchiel, Kant and Agarwal (2019) data fragmentation, data allocation, and data replication are the three distribution techniques that are the basis of this comprehensive study on distributed database systems. Certain issues have been brought to light regarding the design and implementation of these initiatives. The join optimisation issue arises due to data fragmentation whenever a query has to combine several fragments stored in distinct locations. The result is a rapid reaction. This issue is recognised as an NP-Hard problem, and heuristic techniques have been considered for a solution. Another specific issue is data allocation, which entails determining the best way to distribute pieces to sites [48].

In this study, by AlShammari (2020) developers control problems throughout development and maintenance by understanding the evolution and present condition of distributed system issues; this source of knowledge may be valuable. Under 12 overarching topics, this research examined various problems with distributed systems. Some of the most prominent areas of study are privacy and security, service quality, resource management, and synchronisation [49].

This Study, Parvej et al. (2018) an effort to study the frameworks that provide device data consistency and synchronisation. These frameworks facilitate synchronisation with several clients by providing a solution to the unreliable connection issue via tailored replication and synchronisation procedures. Consistency and data model support parameters, as well as synchronisation protocol and conflict resolution approaches, are used to compare the frameworks[50].

In this study, Győrödi et al. (2022) MongoDB and document-based MySQL, two prominent document-based NoSQL databases, were examined for their CRUD operations' complexity and performance, particularly about query operations. A comparative investigation of the effects of each database on application performance while executing CRUD queries is the article's primary goal. The investigation was carried out using a case-study application built using MongoDB and MySQL document-based databases [51].

This Study, Gamero et al. (2022) to examine the consequences and design compromises among two well-known DBMSs, SQL and NoSQL, inside a decoupled architecture. The following databases are considered: MongoDB, MySQL, and AWS Aurora. They look at the technologies and their related design restrictions, then compare them side by side using simulated stress testing based on high-fidelity industrial data and metrics from a big US manufacturer[14].

This Study, Nam, Bang and Choi (2008) a distribution strategy and web application paradigm for portable web apps that can adapt to the latest developments. The method's key features include offline execution via a web server and a DBMS on portable discs like USB memory sticks and a click-and-run interface for installing, configuring, and running web applications. Developing a basic application platform and package builder that supports Ruby and PHP apps has validated the approach[52].

Table 2 summarises the relevant work by outlining the main points and stages.

Table 2 Summary of the related work with key areas/phases discussed

Reference	Topic	Tools, Techniques, Databases	Challenges Addressed	Comparison Parameters	Applications/ Implications	Future Work
[48]	Distributed Database Strategies	Heuristic approaches, distributed database systems	Join optimisation, high response time, data allocation	Distribution strategies: fragmentation, allocation, replication	Insights for database designers in optimising distributed database performance	Explore advanced optimisation techniques and AI-driven allocation models
[49]	Issues in Distributed Systems	Distributed systems frameworks, resource management techniques	Security/privacy, quality of service, resource management	Themes: security, privacy, QoS, synchronisation	Developers gain a better understanding of distributed system challenges and management strategies.	Develop more robust frameworks for resolving synchronisation and scalability issues.

[50]	Data Consistency & Synchronization Frameworks	Synchronisation protocols, conflict resolution techniques, replication frameworks	Unreliable connections, conflict resolution	Consistency, data model support, synchronisation protocol, conflict resolution	Synchronisation of data for multiple clients in distributed environments	Investigate real-time synchronisation for edge devices under highly unreliable conditions
[51]	NoSQL vs SQL Database Behavior	MongoDB, MySQL, CRUD analysis tools	CRUD query performance	Complexity and performance of CRUD operations	Assessment of the impact of database choice on application performance	Extend to other NoSQL databases and analyse in different application domains
[14]	Decoupled Architecture in DBMS	MongoDB, MySQL, AWS Aurora, industrial data simulation tools	Design trade-offs, technological constraints	Metrics: performance, scalability, design constraints	Identification of design trade-offs for industrial database management	Evaluate new decoupled architectures using cloud-native technologies
[52]	Portable Web Applications	Ruby, PHP, embedded DBMS, package builder tools	Offline execution, installation ease	Offline functionality, portability, ease of use	Improved portability and execution of web applications in environments with limited online access	Explore integration with modern containerisation technologies for enhanced portability.

8. Conclusion and Future Work

The efficiency of data synchronisation is crucial for a positive user experience, good reliability, and performance of distributed web applications utilising MySQL, MongoDB, and AWS Aurora. This review has established that each system of the databases has its distinct features which determine its synchronisation, although MySQL has the advantage of high consistency due to the capability of strong transactions, it has limited flexibility in document-based environments. On the other hand, AWS Aurora provides advantages such as effortless scaling and high availability thanks to its cloud design. Varying consistency models have trade-offs, which are also important in determining how an application will function and the overall experience the end user has. Hybrid synchronisations are gaining ground as they focus on strengths of different databases to improve consistency and fault tolerance. As society changes, the need for smart synchronisation techniques that adapt to the load and usage patterns is more evident. In terms of future perspectives, looking at more advanced synchronisation mechanisms with artificial intelligence elements enabling auto-detection and auto-resolution of data conflicts in real time is suggested. It will also be useful to explore the role of decentralised structures and edge computing in synchronising distributed systems in solving the problems of contemporary web applications.

Compliance with ethical standards

Disclosure of conflict of interest

No conflict of interest to be disclosed.

References

[1] C. Y. Huang, "Learning database through developing database web applications," *Int. J. Inf. Educ. Technol.*, 2019, doi: 10.18178/ijiet.2019.9.4.1207.

[2] W. S. Dewi, E. Utami, and B. Sudaryatno, "Database migration using data synchronisation and transactional replication," *Int. J. Innov. Technol. Explor. Eng.*, 2019, doi: 10.35940/ijitee.J9563.0881019.

[3] J. Xu, S. Ye, and X. Zhang, "Data Synchronization Tool for Distributed Heterogeneous Database," *Ruan Jian Xue Bao/Journal Softw.*, 2019, doi: 10.13328/j.cnki.jos.005694.

[4] G. Kousiouris and D. Kyriazis, "Enabling containerized, parametric and distributed database deployment and benchmarking as a service," in *ICPE 2021 - Companion of the ACM/SPEC International Conference on Performance Engineering*, 2021. doi: 10.1145/3447545.3451188.

[5] A. A. Imam, S. Basri, and R. Ahmad, "Data synchronization between mobile devices and server-side databases: A systematic literature review," 2015.

[6] K. Patel, "Quality Assurance In The Age Of Data Analytics: Innovations And Challenges," *Int. J. Creat. Res. Thoughts*, vol. 9, no. 12, pp. f573–f578, 2021.

[7] H. Hu, Y. Wen, T. S. Chua, and X. Li, "Toward scalable systems for big data analytics: A technology tutorial," *IEEE Access*, 2014, doi: 10.1109/ACCESS.2014.2332453.

[8] P. Kotiranta, M. Junkkari, and J. Nummenmaa, "Performance of Graph and Relational Databases in Complex Queries," *Appl. Sci.*, 2022, doi: 10.3390/app12136490.

[9] R. Goyal, "The Role Of Business Analysts In Information Management Projects," *Int. J. Core Eng. Manag.*, vol. 6, no. 9, pp. 76–86, 2020.

[10] O. Alotaibi and E. Pardede, "Transformation of schema from relational database (RDB) to NoSQL databases," *Data*, 2019, doi: 10.3390/data4040148.

[11] A. Anchalia, A. Paudel, R. Sanjeetha, and A. Kakati, "Transforming NoSQL Database to Relational Database: An Algorithmic Approach," in *2022 IEEE 3rd Global Conference for Advancement in Technology, GCAT 2022*, 2022. doi: 10.1109/GCAT55367.2022.9972168.

[12] C. A. Győrödi, D. V. Dumşe-Burescu, R. Győrödi, D. R. Zmaranda, L. Bandici, and D. E. Popescu, "Performance impact of optimization methods on MySQL document-based and relational databases," *Appl. Sci.*, 2021, doi: 10.3390/app11156794.

[13] A. Verbitski *et al.*, "Amazon Aurora," 2018. doi: 10.1145/3183713.3196937.

[14] D. Gamero, A. Dugenske, C. Saldana, T. Kurfess, and K. Fu, "Scalability Testing Approach for Internet of Things for Manufacturing SQL and NoSQL Database Latency and Throughput," *J. Comput. Inf. Sci. Eng.*, 2022, doi: 10.1115/1.4055733.

[15] A. Verbitski *et al.*, "Amazon Aurora: On Avoiding Distributed Consensus for I/Os, Commits, and Membership Changes," in *Proceedings of the 2018 International Conference on Management of Data*, in SIGMOD '18. New York, NY, USA: Association for Computing Machinery, 2018, pp. 789–796. doi: 10.1145/3183713.3196937.

[16] I. Lungu, M. Velicanu, and I. Botha, "Database Systems - Present and Future.," *Inform. Econ.*, vol. 13, no. 1, pp. 84–99, 2009.

[17] R. Arora, "Mitigating Security Risks on Privacy of Sensitive Data used in Cloud-based Mitigating Security Risks on Privacy of Sensitive Data used in Cloud-based ERP Applications," *8th Int. Conf. "Computing Sustain. Glob. Dev.*, no. March, pp. 458–463, 2021.

[18] M. S. Rajeev Arora, Sheetal Gera, "Impact of Cloud Computing Services and Application in Healthcare Sector and to provide improved quality patient care," *IEEE Int. Conf. Cloud Comput. Emerg. Mark. (CCEM), NJ, USA, 2021*, pp. 45–47, 2021.

[19] Vasudhar Sai Thokala, "Efficient Data Modeling and Storage Solutions with SQL and NoSQL Databases in Web Applications," *Int. J. Adv. Res. Sci. Commun. Technol.*, vol. 2, no. 1, pp. 470–482, Apr. 2022, doi: 10.48175/IJARSCT-3861B.

[20] A. Celesti, M. Fazio, and M. Villari, "A study on join operations in MongoDB preserving collections data models for future internet applications," *Futur. Internet*, 2019, doi: 10.3390/fi11040083.

[21] R. Bishukarma, "The Role of AI in Automated Testing and Monitoring in SaaS Environments," *Int. J. Res. Anal. Rev.*, vol. 8, no. 2, pp. 846–852, 2021, [Online]. Available: https://www.ijrar.org/papers/IJRAR21B2597.pdf

[22] R. Bishukarma, "Adaptive AI-Based Anomaly Detection Framework for SaaS Platform Security," *Int. J. Curr. Eng. Technol.*, vol. 12, no. 07, pp. 541–548, 2022, doi: https://doi.org/10.14741/ijcet/v.12.6.8.

[23] M. S. Rajeev Arora, "Applications of Cloud Based ERP Application and how to address Security and Data Privacy Issues in Cloud application," *Himal. Univ.*, 2022.

[24] S. Bauskar, "Predictive Analytics For Sales Forecasting In Enterprise Resource," *Int. Res. J. Mod. Eng. Technol. Sci.*, vol. 04, no. 06, pp. 4607–4618, 2022, doi: https://www.doi.org/10.56726/IRJMETS26271.

[25] I. Arora and A. Gupta, "Cloud Databases: A Paradigm Shift in Databases," *Int. J. Comput. Sci.*, 2012.

[26] V. K. Y. Mohamed Ali Shajahan, Nicholas Richardson, Niravkumar Dhameliya, Bhavik Patel, Sunil Kumar Reddy Anumandla, "AUTOSAR Classic vs. AUTOSAR Adaptive: A Comparative Analysis in Stack Development," *Eng. Int.*, vol. 7, no. 2, pp. 161–178, 2019.

[27] S. Rautmare and D. M. Bhalerao, "MySQL and NoSQL database comparison for IoT application," in *2016 IEEE International Conference on Advances in Computer Applications, ICACA 2016*, 2017. doi: 10.1109/ICACA.2016.7887957.

[28] R. Goyal, "The Role Of Requirement Gathering In Agile Software Development: Strategies For Success And Challenges," *Int. J. Core Eng. Manag.*, vol. 6, no. 12, pp. 142–152, 2021.

[29] P. Tomar, "An Overview of Distributed Databases," 2014.

[30] R. M. Adler, "Distributed Coordination Models for Client/Server Computing," *Computer (Long. Beach. Calif).*, vol. 28, no. 4, pp. 14–22, 1995, doi: 10.1109/2.375173.

[31] K. V. V. and S. G. Jubin Thomas , Piyush Patidar, "An analysis of predictive maintenance strategies in supply chain management," *Int. J. Sci. Res. Arch.*, vol. 06, no. 01, pp. 308–317, 2022, doi: DOI: https://doi.org/10.30574/ijsra.2022.6.1.0144.

[32] K. Patel, "An Analysis of Quality Assurance Practices Based on Software Development Life Cycle (SDLC) Methodologies," *J. Emerg. Technol. Innov. Res.*, vol. 9, no. 12, pp. g587–g592, 2022.

[33] H. S. Chandu, "A Survey of Memory Controller Architectures: Design Trends and Performance Trade-offs," *Int. J. Res. Anal. Rev.*, vol. 9, no. 4, pp. 930–936, 2022.

[34] S. Ying-Chun, Z. Qiu-Ju, and L. Jing, "A kind of heterogeneous database synchronization mechanism," in *Proceedings - 2016 International Conference on Intelligent Transportation, Big Data and Smart City, ICITBS 2016*, 2017. doi: 10.1109/ICITBS.2016.89.

[35] A. P. A. Singh, "STRATEGIC APPROACHES TO MATERIALS DATA COLLECTION AND INVENTORY MANAGEMENT," *Int. J. Bus. Quant. Econ. Appl. Manag. Res.*, vol. 7, no. 5, 2022.

[36] A. P. A. Singh and N. Gameti, "Streamlining Purchase Requisitions and Orders : A Guide to Effective Goods Receipt Management," *J. Emerg. Technol. Innov. Res.*, vol. 8, no. 5, pp. g179–g184, 2021.

[37] C. Madhavaram, H. K. Gollangi, S. Bauskar, and E. P. Galla, "Unveiling the Hidden Patterns : AI-Driven Innovations in Image Processing and Acoustic Signal Detection," *J. Recent Trends Comput. Sci. Eng.*, vol. 8, pp. 25–45, 2020.

[38] R. Goyal, "Software Development Life Cycle Models: A Review Of Their Impact On Project Management," *Int. J. Core Eng. Manag.*, vol. 7, no. 2, pp. 78–87, 2022.

[39] S. Bauskar, "BUSINESS ANALYTICS IN ENTERPRISE SYSTEM BASED ON APPLICATION OF ARTIFICIAL INTELLIGENCE," *Int. Res. J. Mod. Eng. Technol. Sci.*, vol. 04, no. 01, pp. 1861–1870, 2022, doi: DOI : https://www.doi.org/10.56726/IRJMETS18127.

[40] J. Rao *et al.*, "Data migration in the cloud database: A review of vendor solutions and challenges," pp. 96–101, 2022, doi: 10.33545/27076571.2022.v.

[41] S. Tummalapalli and V. R. Machavarapu, "Managing Mysql Cluster Data Using Cloudera Impala," in *Procedia Computer Science*, 2016. doi: 10.1016/j.procs.2016.05.193.

[42] S. Agrawal, J. Prakash Verma, B. Mahidhariya, N. Patel, and A. Patel, "Survey on Mongodb: An Open-Source Document Database," *Int. J. Adv. Res. Eng. Technol.*, vol. 6, no. 12, pp. 1–11, 2015.

[43] B. M. Abdelhafiz, "Distributed Database Using Sharding Database Architecture," in *2020 IEEE Asia-Pacific Conference on Computer Science and Data Engineering, CSDE 2020*, 2020. doi: 10.1109/CSDE50874.2020.9411547.

[44] Y. Wang, J. Wen, W. Fang, and X. Rao, "Research on Incremental Heterogeneous Database Synchronization Update Based on Web Service," in *Proceedings - 2015 International Conference on Computational Intelligence and Communication Networks, CICN 2015*, 2016. doi: 10.1109/CICN.2015.273.

[45] R. Arora, S. Gera, and M. Saxena, "Mitigating Security Risks on Privacy of Sensitive Data used in Cloud-based ERP Applications," in *2021 8th International Conference on Computing for Sustainable Global Development (INDIACom)*, 2021, pp. 458–463.

[46] M. Z. Hasan, R. Fink, M. R. Suyambu, and M. K. Baskaran, "Assessment and improvement of intelligent controllers for elevator energy efficiency," in *IEEE International Conference on Electro Information Technology*, 2012. doi: 10.1109/EIT.2012.6220727.

[47] S. A. and A. Tewari, "AI-Driven Resilience: Enhancing Critical Infrastructure with Edge Computing," *Int. J. Curr. Eng. Technol.*, vol. 12, no. 02, pp. 151–157, 2022, doi: https://doi.org/10.14741/ijcet/v.12.2.9.

[48] K. Ezéchiel, S. Kant, and D. Agarwal, "A systematic review on Distributed Databases Systems and their techniques," *J. Theor. Appl. Inf. Technol.*, vol. 96, 2019.

[49] M. M. AlShammari, "Evolution of Issues in Distributed Systems: A Systematic Review," in *2020 19th International Symposium on Distributed Computing and Applications for Business Engineering and Science (DCABES)*, 2020, pp. 33–37. doi: 10.1109/DCABES50732.2020.00018.

[50] Y. Parvej, I. Ishak, F. Sidi, and M. A., "A Review of Data Synchronization and Consistency Frameworks for Mobile Cloud Applications," *Int. J. Adv. Comput. Sci. Appl.*, vol. 9, 2018, doi: 10.14569/IJACSA.2018.091284.

[51] C. A. Győrödi, D. V. Dumşe-Burescu, D. R. Zmaranda, and R. Győrödi, "A Comparative Study of MongoDB and Document-Based MySQL for Big Data Application Data Management," *Big Data Cogn. Comput.*, 2022, doi: 10.3390/bdcc6020049.

[52] K. H. Nam, K. S. Bang, and W. Choi, "A method for distributing web applications," in *International Conference on Advanced Communication Technology, ICACT*, 2008. doi: 10.1109/ICACT.2008.4494222.