Performance of Database Execution Time for Enhancing Big Data Storage in e-Libraries

Monali Chaudhari
School of Computer Sciences, K. B. C. North Maharashtra University, Jalgaon, Maharashtra, India. Indira University, Pune, Maharashtra, India

Snehalata Shirude
School of Computer Sciences K. B. C. North Maharashtra University, Jalgaon, Maharashtra, India

Varsha Narayanrao Ikhe
School of Information Indira University, Pune, Maharashtra, India

DOI: 10.29322/IJSRP.16.02.2026.p17035
https://dx.doi.org/10.29322/IJSRP.16.02.2026.p17035

Paper Received Date: 9th January 2026
Paper Acceptance Date: 4th February 2026
Paper Publication Date: 12th February 2026

Abstract

Advanced library platforms use large amount of data, referred as Big Data, play a vital role in most transactions, which involve the delivery of data such as course materials and eBooks to students and staff, etc other academic users. The purpose of this paper is to propose an ArangoDB based cohesive model for digital library by evaluating of this multimodel with other data models using query execution time. The study put focus on review of existing data models used in library systems along with their limitations in managing big data. The proposed model eliminates the need for separate databases, reduces query overhead, increases searching, and personalized recommendations. The ArangoDB database consists of approximately 20 million records, out of which testing was done on 10000 records. By analyzing execution times at different scales, results showed that ArangoDB is 60% faster in insert, update and delete operations compared to PostgreSQL, 98% faster with respect to CouchDB but Neo4j has superior execution speed across all operations. The current work focuses primarily on academic library and does not yet evaluate scalability cloud-based repositories. The execution of complex queries become easier using AQL (Arango Query Language). This approach can help institutions modernize their library by integrating structured, unstructured, and graph-based data to optimized backend technologies and unified access mechanisms. Improving the technological backbone of e-libraries enhances information access and supports equitable education by providing faster, more organized access to academic resources for students and faculty. It offers practical guidance to researchers and developers in designing scalable and flexible multi-data models for digital libraries which are suitable for e-learning and digital content management.

Keywords- Big Data, NoSQL, Cohesive Multi-Data Model, ArangoDB, Document data model, Graph Data Model

1. Introduction

"The next generation is full of large volume of data, big data ", As truly said by John Maccsaey (1985), today's world is facing the problem of storage and memory space in the computing world due to increased usage of the internet and computers [1]. This space problem is due to a large volume of data, which is termed as big data [2].

In many organizations, Systems are processing and storing structured, semi-structured and unstructured data. This data is used by organizations to analyse and extract meaningful information and insights in such a way that helps them to improve operational efficiency, provide better customer service, create personalized marketing campaigns and take other actions that can increase revenue and profits.[1] It has been observed that organizations not using big data are not able to make faster and more informed business decisions because they might be using data warehouses which rely on relational databases and contain only structured data. Big data is often stored in a data lake that can support various data types and are using various technologies such as Hadoop clusters, cloud object storage services, Not Only SQL (NoSQL) databases or other big data platforms.[3]

Big data consists mainly of three categories: Volume, Velocity, and Variety. The first V is Volume refers to the size of the data sets is huge compared to regular data, generally it can be in terms of GBs and more. Therefore, the size might vary based on the disciplines. The second V is velocity, which refers to the situation where data is created dynamically and fast. The data is generated rapidly every fraction of second. The third V, refers to variety, which makes big data sets harder to organize and analyze. The regular type of data collected by researchers or business is strictly structured, such as data entered into a spreadsheet with specific rows and columns or into relational databases. However, big datasets might have unstructured data and different types of data, such as email messages or notes. [4]

Big data is not only present in various domains such e-commerce, healthcare but also is present in libraries. Not only this, but also data in the library Big Data includes structured and unstructured data. Unstructured data or Heterogeneous data refers to a dataset composed of different data types, structures, formats or sources.[2] Heterogeneous data can be in terms of user transactional data, session information, books, journals, reports, notes, maps, films, pictures, audios, videos, twitter messages, reviews and recommendations in different categories, which includes various formats and structures. In business contexts, heterogeneous data could emerge from diverse sources like databases, text files, multimedia content, and data streams, among others.[2] Heterogeneous data systems such as data warehousing, data lakes, or hybrid architectures, often utilizing big data technologies like Hadoop, Spark, and NoSQL databases are designed to handle diverse data formats.

Due to the advancement in technology, it led to the rise of a variety of data called heterogeneous data. This in turn creates few issues in such transactions. Issues like delay, data processing and retrieval, security, and space management. Among these, storage is one of the major issues due to a variety of data. [5]

Traditionally for addressing these storage issues, the existing solution is to use Structured Query Language (SQL). SQL is the primary solution contributed by major researchers. In this approach, SQL can only handle the structured portion of data. secondly, their concept is based on the clear line between Online Transaction Processing (OLTP) and Online Application Processing (OLAP). Mostly, data is organized by using Relational data models before storing them as transaction data in SQL databases. These data are retrieved, transformed, and summarized to store in a separate data warehouse for the analytical process. However, when users want to retrieve insight from transaction data or raw data directly, they are unable to do so because of the rigorous design restrictions in dominant relational SQL databases. [4]

In today's era 70% of data is unstructured. To manage with this unstructured data NoSQL came into picture in 1998 by Carlo Strozzi [reference]. NoSQL databases were specifically designed to address the needs of Big Data. It supports unstructured or non-relational data types (nested structure, column families, document, JSON (JavaScript Object Notation), BSON (binary serialization of JSON), and graph). It is schema-less (schema-on-read, implicit schema). It supports scalability, high performance, and fault-tolerance, and is designed for real time, non-uniform big data.[6] Broadly there are four main categories of NoSQL databases, which are Key-Value databases, Document databases, Wide Column databases, and Graph databases.

Mostly, NoSQL databases have their own query language, Application Programming Interface (API) bindings with various programming languages and data models. To rule out these problems, several solutions exist like providing transparent access to heterogeneous data-stores. Adapting query language for each NoSQL data-store is an additional overhead, but at the same time provides flexibility in querying underlying data in an expressive way. Absence of a uniform query language to adopt NoSQL data-stores is one of the biggest hurdles.

A polyglot persistence is one of the solutions in which an application that integrates multiple database models. Multi-model databases provide an elegant and better solution to the challenge of managing heterogeneous data. In contrast to polyglot persistence, a multi-model database naturally supports multiple data models in their native form using a single, integrated backend. Multi-model database management systems unify multiple database systems into one. Multi-model databases store, query, and index data from different models.

This paper aims to review related work of multi-data models and to provide a better solution to overcome the problem of storage in library systems. Rest of the paper is divided as, Section 2: Related work, Section 3: Research Problem, Section 4: Research Objectives, Section 5: Proposed Methodology, Section 6: Results and Analysis, Section 7: Discussion and Interpretation, Section 8: Key Findings, Section 9: Future Enhancement and Section 10: Conclusion.

2. Related Work

Wang et al. (2016) had focused on a comprehensive review and a hybrid approach of existing frameworks and architectures of libraries to expose their data effectively with big data tools for analysis and user interaction. The paper provides detailed descriptions on the practical challenges of integrating big data technologies, such as cost, technical expertise, and user adoption. Library data scalability and data diversity issues were addressed in this paper.

Teets & Goldner (2013) have highlighted that libraries go beyond traditional roles and adopt advanced technologies to manage and expose big data effectively. The methodology proposed by authors gives a theory and discussion on libraries role in data curation and sharing. This paper lacks practical implementations of libraries in managing big data. Research could focus on building library-specific big data tools.

Kaladhar et al. (2018) adopted a review using machine learning algorithms and shown how library management can be transformed by use of big data technologies in better decision-making and resource optimization by focusing on theoretical discussions and proposing potential applications. This paper lacks experimental work , also absence of empirical validation makes it difficult to assess the feasibility of their recommendations. Future work could involve implementing the proposed framework in a real-world library system.

Harrington & Christman (2019) put focus on a comparative analysis of relational databases and NoSQL systems, along with their advantages and limitations in handling library data by describing features of NoSQL such as schema-less databases, scalability, and high availability, can be a better choice for handling growing library datasets. Further research could involve testing NoSQL systems with library datasets to evaluate performance improvements. No practical implementation or experimental validation was demonstrated to show the benefits of NoSQL in a real library environment.

Ruldeviyani & Aji (2016) had demonstrated NoSQL's ability to handle distributed data can overcome the limitations of relational databases in terms of query performance, particularly for large-scale library systems and then put focused on improving query performance in library management systems by implementing NoSQL databases through practical demonstration. This paper shows comparative analysis of NoSQL and traditional relational databases, presenting performance benchmarks (e.g., query response time). They used a case study approach, implementing NoSQL in the library information system of Universitas Indonesia. The findings are specific to one library system and may not generalize to others. Further research could explore scalability, security, and integration with existing systems.

Mehmet Zahid Ercan & Michael Lane (2014) used query performance, data consistency, and scalability as benchmarking techniques to evaluate the suitable NoSQL databases for managing electronic health records (EHR) with a detailed evaluation framework, conducting experiments with healthcare-specific datasets and analyzing the trade-offs between different NoSQL databases, providing actionable insights. The study overlooks critical aspects like data security and compliance along with performance metrics.

Fotache & Cogean (2013) provided detailed experimental results by conducting a mobile application-based case study using MongoDB (NoSQL) and PostgreSQL (SQL), which measures performance of NoSQL and SQL databases for through a hybrid database approach, offering practical recommendations based on specific use cases. The study lacks a broader scope beyond the two database systems tested. Future research could explore additional database technologies and scenarios.

Jong Sung Hwang et al. (2015) Proposes a systematic method for selecting the most suitable database system in big data environments. Performance metrics such as latency, throughput, and scalability were used for evaluation. Additional case studies in other industries could enhance the generalizability of the proposed method.

Srikrishna Prasad & Nunifar Sha M.S (2013) Introduces a polyglot persistence pattern further focusing on healthcare data management challenges, addressing issues like performance optimization and data heterogeneity, etc to handle diverse healthcare data types efficiently. It outline the benefits of polyglot persistence by using conceptual discussions and hypothetical scenarios in healthcare systems, but did not include a working implementation or experimental results. Further work could involve implementing the pattern in a real healthcare environment.

Srikrishna Prasad & Avinash S.B (2014) demostrated a conceptual framework based on polyglot persistence to the energy sector, addressing specific challenges like handling large datasets, optimizing query performance and enhancing effectiveness in energy data management. Combines relational and NoSQL databases to store and process energy data more efficiently. Limited testing under diverse real-world conditions. The study could benefit from detailed performance benchmarks and scalability analysis.

André Magno Costa de Araújo et al. (2016) focused on electronic health records (EHR) and proposed a polyglot persistence framework (PolyEHR) tailored for the healthcare sector which manages the diverse requirements of EHR systems, such as scalability, performance, and data integrity. The framework was designed to handle different types of healthcare data, providing a scalable and efficient solution for EHR management and implemented a proof-of-concept prototype to evaluate its feasibility. The study lacks extensive testing under real-world healthcare scenarios, such as large-scale hospital systems or compliance with legal regulations like HIPAA.

Flores, Ramirez, Toasa G., & Lavín (2018) focused on comparing SQL Server and MongoDB for e-government applications, emphasizing response time differences. The study highlighted how MongoDB initially outperformed SQL Server in query execution but noted that response times converged after repeated queries. However, the research lacked real-world implementation and only used a controlled experimental setup. Further research could involve testing NoSQL and SQL databases on real-time government datasets to validate these findings.

Kanchan, Kaur, & Apoorva (2021) analyzed NoSQL and relational database systems empirically, emphasizing performance trade-offs. The study covered data insertion, retrieval, and scalability, showing that SQL databases excel in structured queries while NoSQL databases offer better scalability. However, it lacked practical testing on large-scale applications. Future studies could extend the analysis to distributed environments and hybrid database models for more comprehensive results.

Sandell, Asplund, Ayele, & Duneld (2024) examined ArangoDB, MySQL, and Neo4j, focusing on connected data queries. Their findings showed Neo4j performing better than MySQL and ArangoDB in handling complex relationships. The study effectively measured execution time and resource consumption but did not explore real-world deployment scenarios. Further research could involve integrating these databases in a multimodal application and evaluating long-term performance.

Truică, Rădulescu, Boicea, & Bucur (2018) investigated CRUD performance in MongoDB, CouchDB, and Couchbase in both single-instance and distributed environments. The study found MongoDB to be the most efficient in distributed setups. However, it did not compare NoSQL databases with relational alternatives, limiting the scope of its conclusions. Future research could expand the study by including SQL databases and testing across different workloads.

Howard (2024) evaluated NoSQL (Cloud Firestore) and SQL (Cloud SQL MySQL) in an energy marketplace context, incorporating GraphQL optimization. The study demonstrated that Firestore provided scalability benefits, and GraphQL improved query efficiency. However, it lacked comparisons with other NoSQL systems and did not analyze the cost-effectiveness of deploying such a hybrid model. Further research could extend the study by testing multiple NoSQL and SQL databases in different cloud environments.

3. Research Problem

In today's era data sources in libraries are heterogeneous and in large volume where storing such data in RDBMS would become more complex and slower. Also, research shows that only a few library information systems are using NoSQL data models. It has been observed that libraries using a single data model for storage and having heterogeneous data are inefficient in query processing as well as lack in performance. Also, scalability issues persist which can degrade the performance evaluation of a single data model.

4. Research Objectives

To propose the cohesive Multi Data Model for enhancing heterogenous Big Data Storage in e-Libraries.

5. Proposed Methodology

In, Library Information System, data for Book, Supplier and Journal are best stored in a document store, as they efficiently handle structured and semi-structured metadata, enabling flexible searches. User data benefits from a document store, ensuring fast authentication and session tracking. To maintain data for borrowing transactions, a document store is used, ensuring data integrity and consistency in lending records. Social relationships, including friendships and teaching interactions, are best managed with a graph database, which efficiently handles complex connections between users. Similarly, feedback and reviews require a combination of graph and document stores, as they involve both textual data and relational links between users, books, and ratings. Using multi-model architecture in the Library Information System ensures efficient storage, fast retrieval, accurate transactions, and intelligent recommendations, ultimately enhancing user experience and operational effectiveness. We proposed a cohesive multi-model which satisfies the requirement for enhancing data storage in the Library Information System.

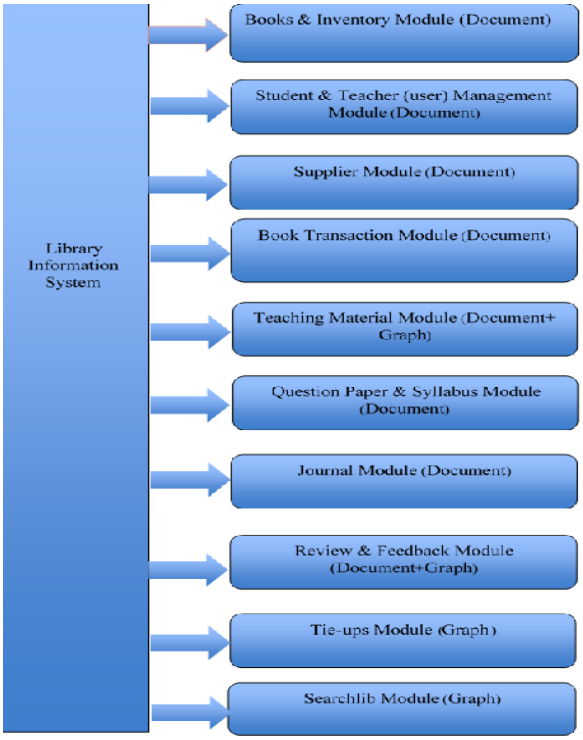

Fig 1 - Cohesive multi-model for Library information System using ArangoDB database

To handle diverse data formats in the Library Information System we can either use any multi-model databases like ArangoDB, OrientDB, Marklogic, etc or create a cohesive multi-model database using polyglot persistence. Here, using ArangoDB database, Developers have one database to maintain, just one adapter for their application and also they have to learn details about just one database (such as query syntax, architecture or topology).

A) Mathematical model for ArangoDB database

The Library Information System in ArangoDB consists of **Document Collections (Doc)** and **Graph Relations (G)**. Hence, the representation of data for Library Information System will be as follows:-

1) Representation of data in ArangoDB

1. Document Collections (Doc):

The document collections can be defined as:

Doc = {B, J, U, S, T, QS, I, BT} where

B = Set of **Books,** J = Set of **Journals,** U = Set of **Users (Students ∪ Teachers),** S = Set of **Suppliers,** T = Set of **Teaching Materials,** QS = Set of **Question Papers & Syllabus,** I = Set of **Inventory Items,** BT = Set of **Book Transaction Logs**

2. Graph Collections (G):

The Graph collections can be defined as a set of V = Set of all vertices = D (Document collections) and E = Set of edges representing relationships.

The Relations (edges) can be represented as:-

Review (user (id, name, role), book_review (id, rating, comment, date)) ∈ E (wrote, evaluates)

Feedback (user (id, name), feedback (id, text, rating, date)) ∈ E (submitted, about)

TieUp (Institution (id, name, location), teacher (id, name, expertise), provider (id, name, type)) ∈ E (Collaborates with, Affiliated with, Provided by)

SearchLib (material(id, title, type),topic(id, name, description),author(name, user type, email),keyword(id, word)) ∈ E (tagged with, contains)

Teaches (material (id, title, type), topic(id, name, description),users (author) (name, user type, email) ∈ E (features , Belongs to, created by, requires, related to)

Hence the Graph Collections, G can be represented as G = (V, E), where E = {Review, Feedback, Teaches, TieUp, SearchLib}

2) Query Model in ArangoDB

A multimodel query in ArangoDB is represented as:

Q: (S, B, P, G)→ R

where Q maps multimodel data into a result R.

S = Document collection Set , **B** = Set of binding variables used in the query

P = Set of predicates, **G** = Set of graph patterns, **R** = Result set

Formal Query Representation:

Q1 : (S,B,P,G)→R1

S ⊆ Suppliers (Document collection)

B = {supplier, book} (Set of binding variables used in the query)

P = {supplier.status == "active"} (Set of predicates)

G = {TieUp (college, supplier) } (Set of graph patterns)

R1 = List of active suppliers with their supplied books (Resultset)

3) Query Execution Time Model

To evaluate the execution cost or performance, we can use a simplified model:

Cost (Q) = CD + CBP + CEP + CGT + CRS

where:

- CD = Accessing and scanning sources cost (documents, edges)
- CBP = variable binding and parameter substitution cost
- CEP = Evaluating predicates cost (filters, joins)

- CGT = graph traversal cost (path expansion, neighbors, etc.)
- CRS = Cost of result construction (aggregation, sorting, returning)

a) Cost of accessing and scanning sources (documents, edges) as CD is

$$CD = \sum_{i=1}^{n} \left(access(si) + scan(si)\right)$$

b) Cost of variable binding and parameter substitution as CBP is

CBP = O(v) (v = number of variables bound)

c) Cost of evaluating predicates (filters, joins) as CEP is

$$CEP = \sum_{i=1}^{n} (filter(pi) + join\ cost)$$

d) Cost of graph traversal (path expansion, neighbors, etc.) as CGT is

$$CGT = d \cdot bd$$

Where: d = depth of traversal, b = average branching factor

e) Cost of result construction (aggregation, sorting, returning) as CRS is

$$CRS = O(n \log n)$$ (if sorted) or O(n) otherwise

Hence, putting all components together the final functional formula is

$$Q(S,B,P,G) \Rightarrow R \text{ with cost } CQ = \sum_{s \in S} scan(s) + \sum_{p \in P} eval(p) + traverse(G) + return(R)$$

B) pseudo code for ArangoDB database connection with frontend (Python, Django, JAVA, etc)

Step 1: Backend Initialization

Step 2: Define Backend API Endpoint for Query Execution

Step 3: Frontend Request Handler

Step 4: Query Execution and Response Cycle

C) Flowchart for ArangoDB database connection with frontend (Python, Django, JAVA, etc)

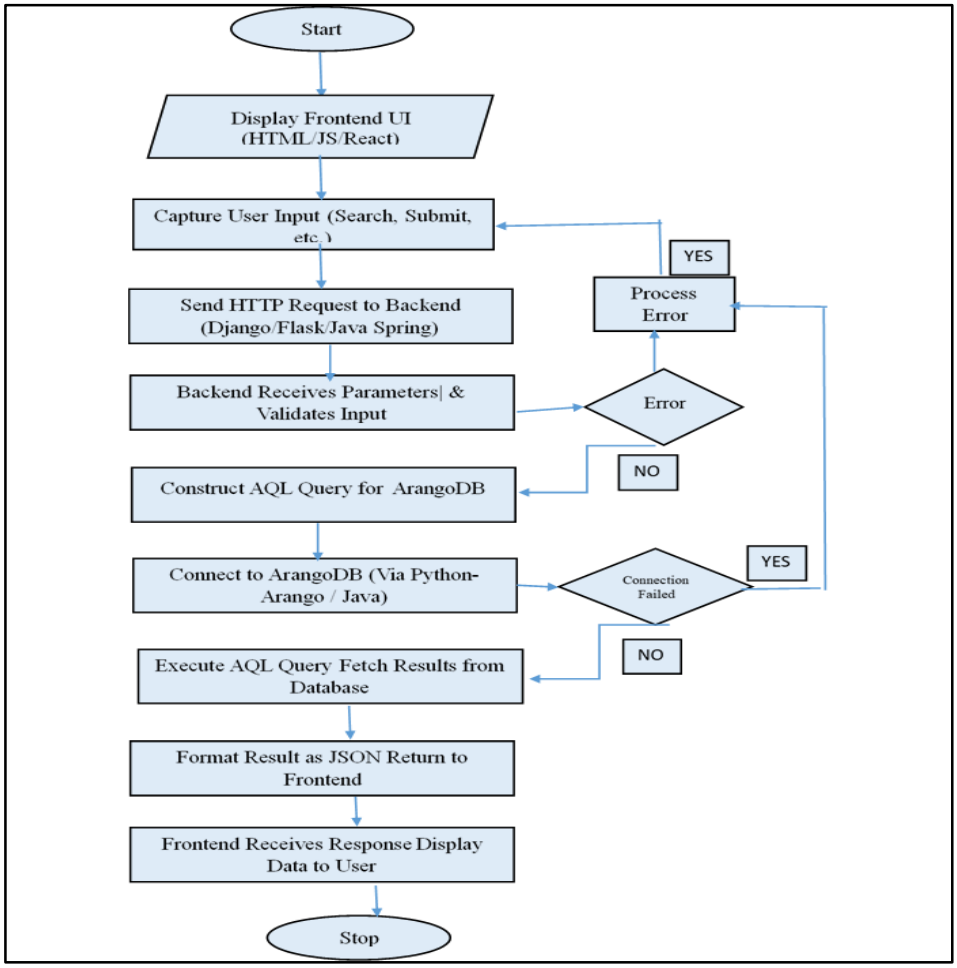

Fig 2- Flowchart for ArangoDB database connection with frontend

D) Complexity evaluation for ArangoDB database connection with frontend (Python, Django, JAVA, etc.)

If we consider,

n = size of the data fetched from ArangoDB (number of documents/records).

q = size/complexity of the AQL query.

c = constant time for network connections.

v = number of input validations or fields.

r = size of the final formatted response (in JSON).

Then,

a) Time Complexity can be calculated as follows:-

1. Frontend UI Interaction = O(1)
2. Capture User Input = O(v)
3. Send HTTP Request = O(c)
4. Backend Validates Input = O(v)
5. Construct AQL Query = O(q)
6. Connect to ArangoDB = O(1) to O(c)
7. Execute AQL & Fetch Results = O(q + n)
8. Format Results (JSON) = O(n)
9. Return to Frontend = O(c)
10. Display Data on Frontend = O(n)

Total Time Complexity required from ArangoDB database connection with frontend and getting result = O (v + q + n) where,

Input validation = O (v)

AQL query construction = O (q)

Result fetching & formatting = O (n)

In most practical scenarios, v and q are small constants. So this simplifies to:
Time Complexity ≈ O (n) (linear with result size)

b) Space Complexity can be calculated as follows:-

1. Frontend UI Interaction = O(1)
2. Capture User Input = O(v)
3. Send HTTP Request = O(v)
4. Backend Validates Input = O(v)
5. Construct AQL Query = O(q)
6. Connect to ArangoDB = O(1)
7. Execute AQL & Fetch Results = O(n)
8. Format Results (JSON) O(r) ≈ O(n)
9. Return to Frontend = O(r)
10. Display Data on Frontend = O(n)

Total Space Complexity = O (v + q + n) where,
Input buffer: O (v)
AQL string: O (q)
Response data: O (n)
In most practical scenarios, v and q are small constants. So this simplifies to:
Space Complexity ≈ O (n)

E) Simulation setup and parameters

a) simulation environment

The proposed library information system was developed and tested on a system with the following configuration: an Intel Core i5 10th Generation processor, 8 GB of RAM, 1 TB storage, and a high-speed internet connection. The database consists of approximately 20 million records, amounting to a total size of around 200 GB. The backend of the system was implemented using Python (version 3.12.0) and the Django web framework (version 5.2), enabling rapid development and secure data handling. The frontend was created using HTML5 and CSS3, ensuring a responsive and user-friendly interface. Data storage and querying were handled using ArangoDB (version 3.11.8 for Windows 64-bit), a multi-model NoSQL database supporting document, key-value, and graph data models. Development was carried out on a Windows operating system using Visual Studio Code (VS Code) as the primary Integrated Development Environment (IDE). The Python code was executed using the CPython interpreter, the default compiler for Python, ensuring efficient runtime performance and compatibility with Django and ArangoDB integrations.

b) simulation parameters

To insert data in the proposed system, we can insert one by one through the GUI or bulk data can be inserted using csv or json file. A script is written in python to import csv file. In the proposed system, we have a requirement of large volume of data i.e. big data. So, considering this fact we have inserted records more than 1,000 in each of the collection. We have collected the dataset from online sources and modified it according to our requirement.

c) Metrics used for query Evaluation

1. Performance (Speed and Efficiency)
Total Execution Time: Time taken from python script start to finish (it includes connection, query, and processing).
Query-Specific Time: Time taken by each database query to establish connection, fetch query to database and complete query execution
Scalability: How execution time is needed for changes with increasing data size (e.g.,
100 vs. 10,000 records).
2. Resource Usage (Efficiency) can be defined as the addition of CPU Usage and Memory Usage where, CPU Usage is the CPU consumed during query execution.
Memory Usage is how much RAM is consumed.
3. Network Evaluation Metrics
Network Latency: Time taken for a request to travel from the client (Python script) to the database server and back (round-trip time, RTT).

d) Simulation steps

To develop a web-based application using Django (Python) with ArangoDB as a multimodel backend, the setup involves multiple stages. Initially, essential tools must be installed including Python, Django, and the ArangoDB server. Django can be installed via pip, and ArangoDB can be installed either locally or through Docker for containerized environments.

6. Results and Analysis

ArangoDB is a native multi-model database that supports document (NoSQL), graph, and key-value models under a single database engine. This allows seamless execution of complex queries involving books, users, borrowing records, and recommendations using AQL (Arango Query Language). For example, borrowing transactions can be stored as documents, while relationships such as "User borrowed Book" can be stored in graph format for efficient recommendations. This eliminates the need for separate databases and reduces query overhead, making it ideal for fast book searches, borrowing history tracking, and personalized book suggestions. Additionally, ArangoDB's scalability and distributed architecture make it a strong choice for large-scale library systems. However, its learning curve for AQL and higher RAM usage compared to traditional RDBMS might be challenging for some implementations.

Library management systems poses multiple factors for evaluating multimodel databases include query performance, data consistency, scalability, ease of maintenance, and integration complexity. We can assess performance scalability by analyzing execution times at different scales, query optimization needs, database efficiency, and real-world applicability. We tested the queries by executing them directly on particular databases as well using python script to execute these queries and find the execution time for insert, update, search and delete operations for 100, 1000 and 10000 records each.

Database	Operation	100 Records	1000 Records	10000 Records
PostgreSQL	Insert	0.039	0.268	0.616
	Update	0.157	0.101	0.267
	Select	0.101	0.162	0.157
	Delete	0.068	0.469	3.363
CouchDB	Insert	5.5253	27.7669	299.0078
	Update	2.8783	31.9452	320.4156
	Select	0.010	0.020	0.015
	Delete	2.7309	33.1802	301.3301
Neo4j	Insert	0.281	0.163	0.189
	Update	0.015	0.016	0.019
	Select	0.007	0.007	0.030
	Delete	0.114	0.061	0.115
ArangoDB	Insert	0.28294	0.094	0.189
	Update	0.211	0.500	3.639
	Select	0.09539	0.02636	0.04883
	Delete	0.09741	0.215	1.635

Table 1 – Execution time in seconds required to execute query on database

Database	Operation	100 Records	1000 Records	10000 Records
PostgreSQL	Insert	0.2051	1.8267	18.8337
	Update	0.3153	0.0771	0.5867
	Select	0.0000553	0.0000295	0.0000309
	Delete	0.6086	0.0702	0.5699
CouchDB	Insert	7.2692	59.0753	397.6720
	Update	4.4149	4.9176	8.5523
	Select	2.5568	3.4850	5.1231
	Delete	4.1609	4.8040	8.7344
Neo4j	Insert	2.2081	2.2567	2.8779
	Update	2.7206	2.4391	2.4189
	Select	2.1727	2.2232	4.1135
	Delete	2.3267	2.0592	2.0466
ArangoDB	Insert	0.0170	0.0870	0.8700

	Update	0.8430	6.5320	45.3200
	Select	0.2970	9.1920	45.7800
	Delete	0.8330	11.5930	65.5600

Table 2 – Execution time in seconds required to send query from python to database, execute query on database

7. Discussion and Interpretation

When query from python to database is given, for the Insert operation, ArangoDB demonstrated the fastest and most optimized performance, showing minimal overhead even at large scales. Neo4j also performed consistently well, with only a slight increase in execution time as data volume grew. PostgreSQL scaled steadily and predictably, though not as fast as ArangoDB or Neo4j. In contrast, CouchDB showed the slowest insert times, with performance significantly degrading as the number of records increased due to its document-based architecture and overhead.

In the case of Select operations, both PostgreSQL and Neo4j maintained excellent performance, with PostgreSQL remaining particularly stable even at 10,000 records. Neo4j handled large data volumes efficiently, maintaining fast and consistent response times. CouchDB's performance slightly declined as the dataset grew, though it remained usable. However, ArangoDB experienced a sharp drop in select performance with larger datasets, indicating challenges in scaling read-heavy workloads efficiently.

When evaluating Update operations, PostgreSQL and Neo4j again showed high efficiency, delivering minimal execution time growth even with large datasets. They proved to be reliable choices for update-heavy applications. CouchDB, on the other hand, exhibited the most performance degradation, with update times increasing drastically as record counts grew. ArangoDB started well with small datasets but struggled with updates at scale, indicating it's more suited for mixed workloads or scenarios with moderate update frequency.

Finally, for Delete operations, PostgreSQL and Neo4j once again led with efficient and stable deletion performance, even as record counts scaled up. CouchDB performed the worst, showing significant delays in deletion as the dataset size increased, likely due to the overhead of document revisioning. ArangoDB also experienced a substantial increase in delete times with larger datasets, making it more suitable for applications where deletion frequency is lower or batched.

8. Key Findings

Based on the performance observations across insert, select, update, and delete operations at increasing data volumes, each database shows strengths in specific areas. For Insert operations, ArangoDB is equal to Neo4j, but 69% faster than PostgreSQL as well as 99.94% faster than CouchDB. Same if we compare the select operations than ArangoDB is 69% faster than PostgreSQL, 225% slower than CouchDB and 63% slower than Neo4j. For Update operations 1100% slower than PostgreSQL, 18000% slower than Neo4j, Still 98.86% faster than CouchDB. For delete operations 51% faster than PostgreSQL, 99.46% faster than CouchDB and 1200% slower than Neo4j. For a system requiring balanced and consistent performance across all CRUD operations, PostgreSQL and Neo4j emerge as the most reliable choices. PostgreSQL offers excellent scalability, ACID compliance, and efficient handling of large structured datasets, making it ideal for traditional library management systems with complex relational data and transaction requirements.

Neo4j, on the other hand, is well-suited for applications that rely heavily on relationships and recommendations, such as user-book interaction graphs, borrowing patterns, and semantic search. Its performance remains stable even with large volumes, especially for select and update operations.

ArangoDB stands out for its high-speed insertions and flexible multi-model architecture, making it a strong candidate for systems with a mix of document, key-value, and graph data. However, its performance drops significantly in select and delete operations at large scale, which may limit its use for read- or delete-intensive workloads unless further optimized.

CouchDB, while useful for document-based storage, demonstrated the least efficiency under scale, particularly in update and delete operations. Its performance limitations at higher volumes suggest it is more suitable for smaller-scale or less write-intensive systems unless paired with aggressive optimization.

9. Future Enhancement

We can further develop a hybrid model using PostgreSQL for transactional data (e.g., user accounts, borrowing history), Neo4j for recommendation and relationship graphs, and ArangoDB or MongoDB for flexible metadata storage. Combining the strengths of multiple database systems can help to create a highly responsive and scalable hybrid library information system.

10. Conclusion

ArangoDB had the best insert performance, while CouchDB was the slowest due to document storage overhead. PostgreSQL and Neo4j had consistently fast select times, while ArangoDB's select time increased at 10,000 records. PostgreSQL and Neo4j updated data the fastest, while CouchDB had the longest update times due to its distributed nature. PostgreSQL and Neo4j handled deletions efficiently, while CouchDB had the highest delete times.

The ArangoDB-based approach is simpler to maintain and ensures efficient multi-model queries in a single system, reducing integration complexity. However, hybrid multi-model databases combining CouchDB, PostgreSQL, and Neo4j provide specialized optimization for each data model, making it a powerful choice for high-traffic library systems with distinct data storage needs. The decision between these approaches depends on the system's scale, required query performance, and administrative complexity. If a unified, scalable solution with multi-model support is preferred, ArangoDB is the better option.

References

1. Cameron H,"What is Big Data and Why is it Important?", https://www.techtarget.com/searchdatamanagement/definition/big-data, Accessed on 22 Jun 2024
2. "What is Heterogeneous Data? – Dremio", https://www.dremio.com/wiki/heterogeneous-data/#:~:text=Heterogeneous%20data%20refers%20to%20a,%2Dstructured%2C%20and%20unstructured%20data., Accessed on 22 Jun 2024
3. Wang, C., Xu, S., Chen, L., & Chen, X. (2016). Exposing library data with big data technology: A review. In 2016 IEEE/ACIS 15th International Conference on Computer and Information Science (ICIS)(pp. 1-6). In IEEE, June (pp. 26-29).
4. Khine, P. P., & Wang, Z. (2019). A review of polyglot persistence in the big data world. Information, 10(4), 141.
5. Oliveira, F. R., & del Val Cura, L. (2016, July). Performance evaluation of NoSQL multi-model data stores in polyglot persistence applications. In Proceedings of the 20th International Database Engineering & Applications Symposium (pp. 230-235).
6. Storey, V. C., & Song, I. Y. (2017). Big data technologies and management: What conceptual modeling can do. Data & Knowledge Engineering, 108, 50-67.
7. Harrington, M. D., & Christman, D. B. (2019). (Un) Structuring for the Next Generation: New Possibilities for Library Data with NoSQL.
8. Ruldeviyani, Y., & Aji, R. F. (2016, October). Enhancing query performance of library information systems using NoSQL DBMS: Case study on library information systems of Universitas Indonesia. In 2016 International Workshop on Big Data and Information Security (IWBIS) (pp. 41-46). IEEE.
9. Kaladhar, A., Naick, B. D., & Rao, K. S. (2018). Application of big data technology to library data: a review. International Journal of Library and Information Studies, 8(2), 25-30.
10. Teets, M., & Goldner, M. (2013). Libraries' role in curating and exposing big data. Future Internet, 5(3), 429-438.
11. Xi, Y. (2018, December). Research on the construction of library data integration system in big data era. In 2018 International Conference on Transportation & Logistics, Information & Communication, Smart City (TLICSC 2018) (pp. 112-116). Atlantis Press.
12. Ran Tan; Rada Chirkova; Vijay Gadepally; Timothy G. Mattson, Enabling query processing across heterogeneous data models: A survey, 2017 IEEE International Conference on Big Data (Big Data), 11-14 Dec. 2017, DOI: 10.1109/BigData.2017.8258302, Boston, MA, USA
13. Mehmet Zahid Ercan, Michael Lane, Evaluation of NoSQL databases for EHR systems, 25th Australasian Conference on Information Systems 8th -10th Dec 2014, Auckland, New Zealand
14. Marin Fotache, Dragos Cogean, NoSQL and SQL Databases for Mobile Applications. Case Study: MongoDB versus PostgreSQL, Informatica Economică vol. 17, no. 2/2013 DOI: 10.12948/issn14531305/17.2.2013.04
15. Jong Sung Hwang, Sangwon Lee, Yeonwoo Lee, and Sungbum Park, A Selection Method of Database System in Bigdata Environment: A Case Study From Smart Education Service in Korea, Int. J. Advance Soft Compu. Appl, Vol. 7, No. 1, March 2015 ISSN 2074-8523
16. Srikrishna Prasad, Nunifar Sha M.S, NextGen Data Persistence Pattern in Healthcare: Polyglot Persistence, 4th ICCCNT - 2013 July 4 - 6, 2013, Tiruchengode, India
17. Srikrishna Prasad, Avinash S B, Application of Polyglot Persistence to Enhance Performance of the Energy Data Management Systems, 2014 International Conference on Advances in Electronics, Computers and Communications (ICAECC)
18. André Magno Costa de Araújo1, Valéria Cesário Times1, Marcus Urbano da Silva1, PolyEHR: A Framework for Polyglot Persistence of the Electronic Health Record, Int'l Conf. Internet Computing and Internet of Things | ICOMP'16