International Journal of Scientific Research in Engineering and Management (IJSREM)
Volume: 07 Issue: 05 | May - 2023 Impact Factor: 8.176 ISSN: 2582-3930

AN UNSTRUCTURED DATA MANAGEMNET SYSTEM USING NO-SQL DATABASE

MD. MUZAKKIR HOSSEN KHAN[1], MD. MIZAN[2], RAAD SHAHAMAT CHOWDHURY[3]

[123]*Dept. of Computer Science and Engineering & Anna University, Chennai*

Abstract - Unstructured data is information that is not arranged according to any preset data model or schema, and therefore cannot be stored in a traditional relational database system because there's no certain criteria for query. Traditional SQL based systems use relational tables to store data. the problem with these traditional solutions is that it cannot work with unorganized, unstructured data like media files. Our framework introduces a new logical schema for MongoDB's document database system that uses no-SQL document database. From small businesses to professionals this system can provide data management to anyone who needs to keep digital records. This data system does not require any structure of data nor it has any relational tables. It is the future of data storage and retrieval because of the ability to receive and store data as it is generated. The proposed logical schema can analyze unstructured data to generate various user requested parameters. Its graphical user interface is created using React - A JavaScript library, which creates a highly dynamic and volatile user interface. The planned integration of artificial intelligence will unlock even more possibilities. We have decided to name the system MIN2.0 which is short for Minimizer 2.0.

***Key Words*:** Unstructured Data, Logical Schema, No-SQL Document Database

I. INTRODUCTION

According to the latest estimates, 328.77 million terabytes of data is created each day. which includes data that is newly generated, captured, copied, or consumed. In zettabytes, that equates to 120 zettabytes per year, 10 zettabytes per month, 2.31 zettabytes per week, or 0.33 zettabytes every day. This is referred to as big data in recent times. Big data is defined as the large, diverse sets of information that grow at ever-increasing rates. It encompasses the volume of information, the velocity or speed at which it is created and collected. Big data often comes from data mining and arrives in multiple unstructured formats. 80% to 90% of data generated and collected by organizations is unstructured, and its volumes are growing rapidly. Which traditional SQL based data systems cannot address.

To respond to this situation, we need a solid solution to store, organize and manage the huge amount of data that is generated every day. Our framework containerizes these types of data by storing key value pairs into documents and documents into collections. The system essentially focuses on four operations that manipulate the data stream which are Create, Read, Update and Delete. And creates a JSON object file of the input data which can be any type of data. It also generates a unique identifier of that particular data so, we can easily retrieve the recorded data from the database system with simple queries from the server. Our vision is to provide a strong client sided schema to the schema-less design of MongoDB database that helps in creating this highly scalable data management system.

II. Existing System

Thought:

Sql based RDMS data management systems have been used since people required databases to store data. But, due to the latest rise in bigdata it can no longer support the needs of data management.

Technique: Sql-based Relational Database

Disadvantage: Cannot handle unstructured data

III. Proposed System

Thought:

To create a highly volatile data system that stores data in collections instead of relational tables. And provide a software schema that can be scaled to as many participants as needed. This helps us tackle the surge in unorganized data in recent times create storage for the data, read the data from database, update the data and delete the data safely.

Technique: No-Sql database, JSON Document

Advantage: It can store any type of data as a document in the database.

IV. System Implementation

The eval() function in JavaScript is used to take any expression and return as a string. As a result, it can be used to convert any data into a string. This string is then divided into key value pairs to create a JSON object file. After an object is converted it is automatically assigned to an identifier which is usually a combination of 24 strings and numbers. This identifier can be used to perform various operations on the input data.

How is Unorganized Data Stored?

The main function of the implemented system is converting unorganized data into objects known as JSON. These JSON files are then directly stored into the data cluster.

Data can be converted into a collection through a process known as "data structuring". This involves organizing the data as documents in a specific way that makes it easier to store, access, and manipulate. Different types of data have different data structures which have their own unique characteristics and advantages, and the choice of which one to use is set by our data processing system.

Data cluster used by the system can be represented as followed:

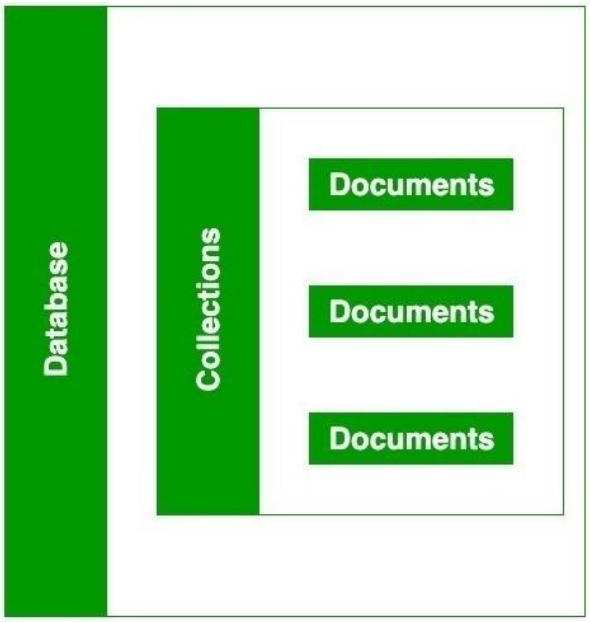


Fig – Collection Based Database

- **Unstructured Data Indexing**

Indexes support the efficient execution of queries of our system. Without indexes, the system must perform a collection scan, i.e. scan every document in a collection, to select those documents that match the query statement. If an appropriate index exists for a query, it uses the index to limit the number of documents it must inspect.

Indexes are special data structures that store a small portion of the collection's data set in an easy to traverse form. The index stores the value of a specific field or set of fields, ordered by the value of the field. The ordering of the index entries supports efficient equality matches and range-based query operations. In addition, our system can return sorted results by using the ordering in the index.
The following diagram illustrates a query that selects and orders the matching documents using an index:

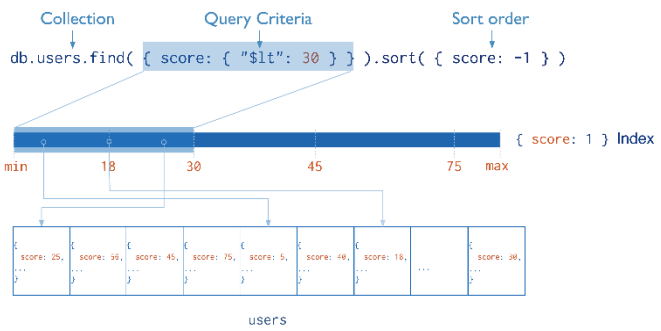


Fig – Data Indexing

Indexing Algorithms Supported by the System:

- **Single Field**

In addition to the MongoDB-defined _id index, MongoDB supports the creation of user-defined ascending/descending indexes on a single field of a document.

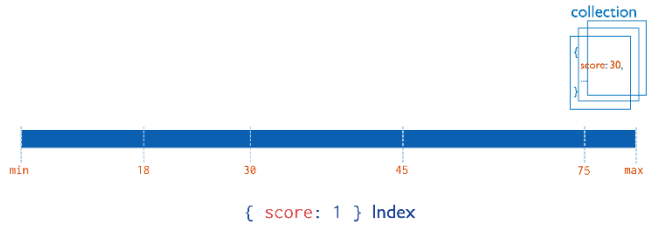


Fig – Single Field Data Indexing

For a single-field index and sort operations, the sort order (i.e. ascending or descending) of the index key does not matter because the system can traverse the index in either direction.

- **Compound Index**

It also supports user-defined indexes on multiple fields, i.e. compound indexes.

The order of fields listed in a compound index has significance. For instance, if a compound index consists of { userid: 1, score: -1 }, the index sorts first by userid and then, within each userid value, sorts by score.

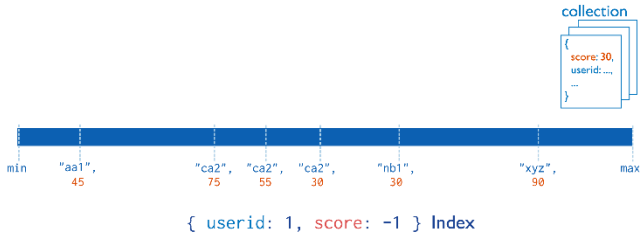


Fig – Compound Data Indexing

For compound indexes and sort operations, the sort order (i.e. ascending or descending) of the index keys can determine whether the index can support a sort operation

- **Multikey Index**

Our framework uses multikey indexes to index the content stored in arrays. If you index a field that holds an array value, it creates separate index entries for every

element of the array. These multikey indexes allow queries to select documents that contain arrays by matching on element or elements of the arrays. MIN2.0 automatically determines whether to create a multikey index if the indexed field contains an array value; user does not need to explicitly specify the multikey type.

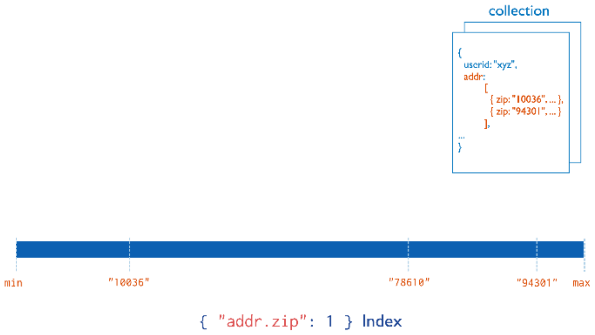


Fig – Multikey Index

- **Geospatial Index**

To support efficient queries of geospatial coordinate data, our system provides two special indexes: 2d indexes that uses planar geometry when returning results and 2dsphere indexes that use spherical geometry to return results.

This index is a sparse index. If you want to use a sparse index to create a compound index, first we review the special considerations of using sparse compound indexes. Then the query is stored.

- **Text Search Indexes**

For data hosted on MongoDB Atlas, the system can support full-text search with Atlas Search indexes. It creates an Atlas Search Index for full text search.

For self-managed (non-Atlas) deployments, MongoDB provides a text index type that supports searching for string content in a collection called text Indexes.
This index is a sparse index. If user wants to use a sparse index to create a compound index, first the system reviews the special considerations of using sparse compound indexes.

- **Hashed Indexes**

To support hash based sharding, our framework provides a hashed index type, which indexes the hash of the value of a field. These indexes have a more random distribution of values along their range, but only support equality matches and cannot support range-based queries.

- **Clustered Indexes**

Starting in MongoDB 5.3, you can create a collection with a clustered index. Collections created with a clustered index are called clustered collections.

With this clustering containerization system our framework gains its most important functionality.

Dataflow Diagram

A data flow diagram (DFD) shows the "stream" of data moving through an information system graphically. It differs from a flowchart because it displays the program's data stream rather than its control stream. Similarly, data handling may be represented using a data stream diagram. The DFD is intended to demonstrate how a structure is divided into less obvious parts.

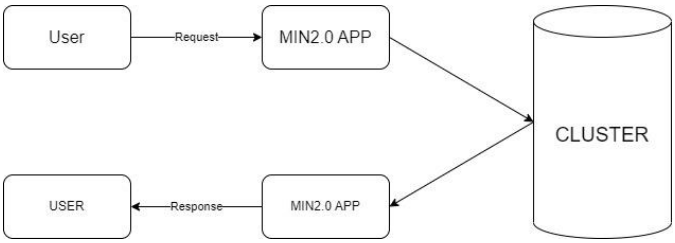


Fig – Dataflow Diagram

V. Methodology

The whole project is built upon the combination of various dependencies to demonstrate the working of the system,

The program server is initially created with the NodeJS which is an open source server environment, so first we have created a server that communicates directly with our MongoDB M0 data cluster. The server uses **Nodemon, Cors** etc. dependencies.

After, creation of the initial server we created a REST api for the purpose of request and response from and to database from the client. This concludes the backend system implementation of our Application. It stores data based on sharding.

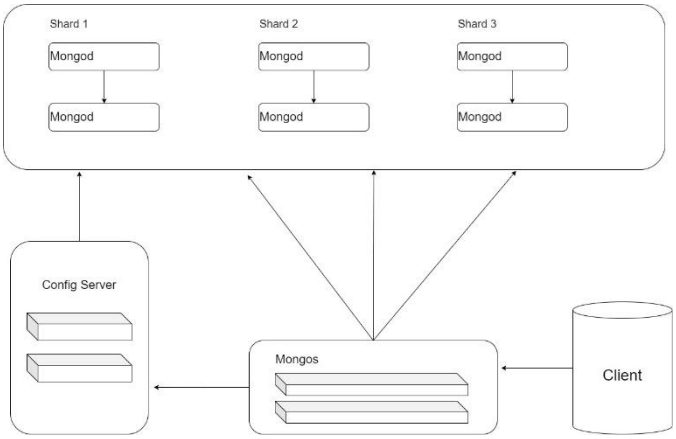
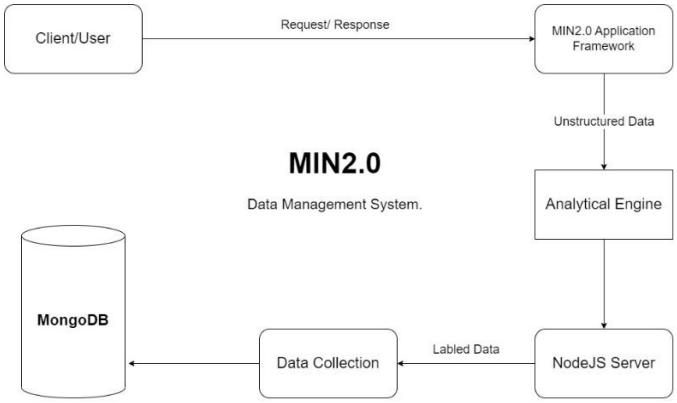


Fig – Data Sharding

Coming to the client side we again made a client side NodeJS system. But it was created with Express framework this time. Express provides the skeletal design of the frontend system of application. The express framework has routes created that connects to the backend data models of our system.

Finally, MIN2.0 app gains its truest look and feel with the integration of REACT making it a highly volatile and responsive application. The React framework provides the

graphical user interface of the software. It's unparalleled optimization of code and transitive communication with the rest of the system results in a seamless user-friendly experience. The data management system that provides an one clicks direct access to any of the data manipulation requirement of the user.



VI. Objectives

- Creating an unstructured data storage system.
- Which is a highly volatile no-sql data management system.
- It is complicated to handle so many different kinds of data, but our system tries to overcome most of it.
- Scale as much as required by the user while maintaining the same exact schema for the management system.
- Convert any type of data into JSON or BSON object format and store it into the cluster.
- Making a system with little to no downtime, so it can write data as they are generated.

VII. Results

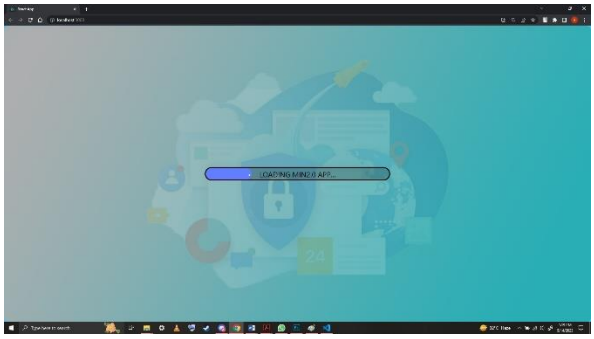


Fig – Initializing Engine

This screen is displayed while the system loads.

Fig – The Engine

This is an instance of the empty engine

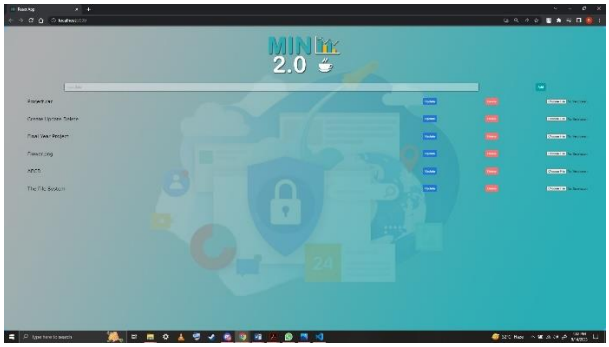


Fig – MIN2.0 Engine with Various Entries

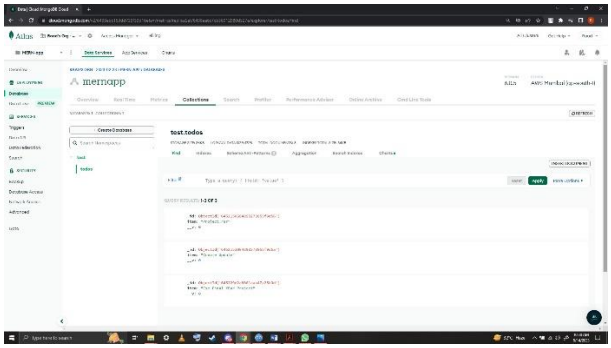
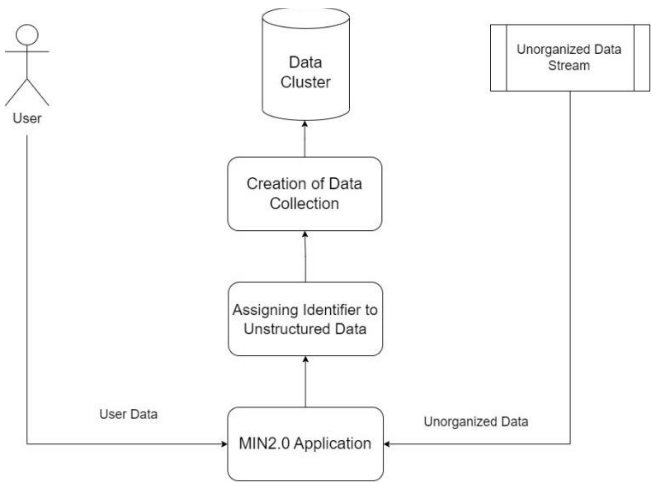


Fig – Collection Based Cluster

VIII. Workflow



IX. CONCLUSIONS

Min2.0, Although being an experimental project, has a lot of potential. The use of DOM or Document Object Modeling enables database entries to be stored as an object. The need for data to be defined in any of the predefined classes of the prior database engines ends with this project. Any data or any information that resembles text, numbers or symbols can be stored in our application as a JSON object entry. It also enables the Users to create their own types and classes of data where the user can themselves store the information.

React, a Java Script based framework helps the system to make changes to the database and the code of the main program itself without the need of refreshing the server. It has potential to become the future of data management systems due to the strong logical schema and the core of the system. The use of Min2.0 will redefine the way we view data.

X. Future Scope

- We will be regularly updating to support more and more data types.
- Currently we are on a 500MB M0 cluster for development and test purpose. we will scale the system to a much higher tier as the user base grows.
- Our developer build already has an artificial intelligence engine named TypeWise AI integrated that can analyze user information to summarize and predict in the criteria they desire.
- The software is deployed to an online server. But there will be an offline build of MIN2.0 which will also be able to help manage user data locally. The user's hardware-based storage will be converted to a data cluster that can help manage local files.
- MIN2.0 is unique in the sense that it is a hybrid between a database management system and a file system
- The application will be converted into a full-fledged Application Programming Interface(API) in future so, it can be used as a part of various software systems handling their data operations

REFERENCES

[1] J. Gray and A. Reuter, *Transaction Processing*. Burlington, MA, USA:Morgan Kaufmann, 1993.

[2] S. Gilbert and N. Lynch, ``Brewer's conjecture and the feasibility of consistent, available, partition-tolerant Web services," *ACM SIGACT News*, vol. 33, no. 2, pp. 51_59, Jun. 2002.

[3] H. Garcia-Molina, J. D. Ullman, and J. Widom, *Database Systems: The Complete Book*, 2nd ed. Upper Saddle River, NJ, USA: Prentice-Hall Press, 2008.

[4] J. N. Gray, *Notes Data Base Operating System*. Berlin, Germany: Springer,1978.

[5] K. P. Eswaran, J. N. Gray, R. A. Lorie, and I. L. Traiger, ``The notions of consistency and predicate locks in a database system," *Commun. ACM*, vol. 19, no. 11, pp. 624_633, Nov. 1976.

[6] F. Chang, J. Dean, S. Ghemawat,W. C. Hsieh, D. A.Wallach, M. Burrows, T. Chandra, A. Fikes, and R. E. Gruber, ``Bigtable: A distributed storage system for structured data," *ACM Trans. Comput. Syst.*, vol. 26, no. 2, pp. 1_26, Jun. 2008.

[7] J. Gantz and D. Reinsel, *The Digital Universe Decade_Are You Ready*. Needham, MA, USA: IDC, 2010. Accessed: Jan. 26, 2014. [Online]. Available: http://www.emc.com/collateral/analyst-reports/idc-digital-universeare-you-ready.pdf

[8] E. A. Brewer, ``Towards robust distributed systems," in *Proc. PODC*, 2000, p. 7.

BIOGRAPHIES

Md. Muzakkir Hossen Khan is a motivated and dedicated student pursuing a Bachelor's degree in Computer Science and Engineering from Anna University, Chennai. He has a solid foundation in computer science

Principles, programming languages, and problem-solving, and is eager to specialize in web development through his career.

Md. Mizan is currently studying Bachelor's in Computer Science and Engineering from Anna University, Chennai. He's an extremely talented data analyst and strategist, has numerous projects and achievements under his belt.

He plans to establish initiatives that affect an organization's IT and business strategy. The project is his brain child

Raad Shahamat Chowdhury is an engineering student currently pursuing bachelor's in Computer Science and Engineering from Anna University. He has great skills and knowledge about the current trends in technology and a aspires

to be a software developer. The tech-stack and system implementation of the project was his initiative.