Maximizing Bigdata Retrieval: Block as a Value for NoSQL over SQL

Abdulrauf Gidado
School of Computer Science
University of Windsor
Windsor, ON N9B 3P4
gidado@uwindsor.ca

C. I. Ezeife*
School of Computer Science
University of Windsor
Windsor, ON N9B 3P4
cezeife@uwindsor.ca

***Abstract*—This paper presents NoSQL Over SQL Block as a Value Database (NOSD), a system that speeds up data retrieval time and availability in very large relational databases. NOSD proposes a Block as a Value model (BaaV). Unlike a relational database model where a relation is $R(K, A_1, A_2, ...A_n)$, with a key attribute K and a set of attributes of the relation: $A_1, A_2, ...A_n$, BaaV represents a relation $R(K, r_1, r_2, ...r_n)$ with a key attribute K and a set of n relations called blocks. Each r contains a set of its own attributes denoted as $r(k, a_1, a_2, ...a_n)$ with a key attribute k and a set of n attributes. The relations $r_1, r_2, ...r_n$ in R are related through foreign key relationships to a super relation R with primary key K. The BaaV model is then denoted in a keyed block format $R\{K, B\}$, where K is a key to a block of values B of partial relations implemented on NoSQL databases and replicating existing large relational database systems. As opposed to conventional systems such as Zidian, Google's Spanner, SparkSQL and Simple Buttom-Up (SBU) which implement SQL over NoSQL and replicate data into different nodes, NOSD implements NoSQL over SQL and uses Lucene functionality on NoSQL to enhance data retrieval costs. Experimenting with our proposed model, we demonstrated the performance of NOSD under the following conditions to prove its novelty (a) scan free queries, and (b) bounded queries on NoSQL databases. We showed that NOSD (a) performs excellently than ordinary relational databases (b) guarantees no scans for no scan queries (c) allows parallelization in query execution, and (d) can be deployed into existing SQL databases with guaranteed horizontal scalability, data retention and accurate autonomous data replication. Using existing benchmark systems, we demonstrated that NOSD outperforms existing SQL databases, SQL over NoSQL systems and is novel in ensuring that existing large SQL database systems utilize the functionalities of NoSQL databases without data loss.**

***Index Terms*—NoSQL, BigData, Relational Database, Block as a Value**

I. Introduction

The amount of data produced by a large variety of web applications in today's information technology driven world is enormous and in different forms. This is evident taken Facebook social networking application as an instance; as at December, 2021, there were more than 100 billion daily messages on Facebook with more than 2 billion monthly active users, out of which more than 100 million are business accounts as stated by [1]. These messages are in different forms, including texts, images, videos, and other documents forms. Over the years, several systems have been proposed and implemented for saving these heterogeneous data, which include systems such as Dynamo [2], HBase [3], Memcache [4] and Cassandra [5]. With the skyrocket increase in web applications, databases are faced with new challenges, including the ability to store large data sets, the fast changing nature of data in terms of their forms/data-types and meanings, and the ability to effectively retrieve, analyse and discover patterns from these data. The inherent nature of these data referred to as bigdata makes it impossible to store and effectively analyse them on a single machine or even clusters [6]. The nature of this data also affects the time and accuracy cost on traditional relational databases. The need to store, retrieve and analyse these data effectively, leads to the innovation of new techniques of storing data known as NoSQL (i.e., Not Only SQL), which currently have more than 100 of its implementations [7] such as Cassandra [5], Elasticsearch [8], MongoDB [9] and many more. Majority of these NoSQL databases are based on the MapReduce data storage techniques proposed by Google [10]. MapReduce [10] is a NoSQL system that partitions and distributes data for processing between machine nodes, and Dynamo is another Keyed-Value database model developed by Amazon [2].

A Schema of Relational-Like Key-Value data representation

Problem 1: Given a relational database D, formulate a Key-Value representation of D.

Employees(**emp_id**, first_name, last_name, gender, hire_date, dept_no)
Departments(**dept_no**, dept_name)
Dept_Emp(**emp_id**, **dept_no**, from_date, to_date)
Salaries(**emp_id**, salary, from_date, to_date)

Given an instance of Employee table of the schema D below:

TABLE I Employee Table

emp_id	first_name	last_name	gender	hire_date	dept_no
1	John	Bull	Male	2021-09-12	4
2	Peter	Susan	Female	2000-03-12	2
3	Ali	Zang	Female	1989-12-03	3

*This research was supported by the Natural Science and Engineering Research Council (NSERC) of Canada under an Operating grant (OGP-0194134) and a University of Windsor grant.

IEEE/ACM ASONAM 2022, November 10-13, 2022
978-1-6654-5661-6/22/$31.00 © 2022 IEEE

Solution 1:BaaV Representation of Instance in NoSQL

The above example is a typical relational database schema. A simple Key-Value model of the above would be a straightforward mapping of the primary key column(s) to the respective attributes per tuple entry. This means that for relation Employee, the key for each tuple is the emp_id while the respective values are the corresponding attributes for the tuples of Employee (i.e., first_name, last_name, gender, hire_date, dept_no). This Key-Value representation of the above instance is similar to the instance representations in relational database provided in the table I above.

However, to represent the above schema in our proposed BaaV model, a super key is chosen for all the set of relations in the given schema. Using the example above, the emp_id occurs in most of the relations and/or could be linked to all other relations in the schema. In our proposed and implemented BaaV model, given a document-based NoSQL database N (such as Elasticsearch) and a relational database D, for each tuple in all the SQL relations of the schema D, emp_id is chosen as a super key to all the sets of relations (i.e., block in BaaV) in the schema. This super key emp_id serves as the document id at the NoSQL database N part of the NOSD system. A simple document-based NoSQL representation of a tuple in the given instance D is presented below in a document-based NoSQL JSON format. A detailed BaaV representation of a relational database and further details on our proposed data replication algorithm is provided in section V of this paper.

BaaV Representation of Instance in Problem 1:

{
”employee.createIndex()”:[
{ ”emp_d”:1 },
{ ”unique”:true }
],
”employee.insertOne()”:[
{ ”emp_id”:”1”, ”first_name”:”John”, ”last_name”:”Bull”, ”gender”:”Male”, ”hire_date”:”2021-09-12”, ”department”:[
{ ”dept_no”:”4”, ”dept_name”:”Finance” }]
”salary”:[{ ”salary per annum”:”150,000”, ”from_date”:”2021-09-12”, ”to_date”:”2022-01-30”
}] }] }

Subsequent to the development of Dynamo [2], there has been prevalent use and development of other databases based on the Key-Value database concept. Systems such as Bigtable developed by Google [11], F1 [12], Casandra produced by Apache [5], and HBase [3] used the Key-Value data storage concepts. In the Key-Value store concept, each key is linked to specific set of data, mapped in the form of primary key – data relation in a relational database table concept. Over the years, the Key-Value gained traction due to its flexibility of scaling. However, its earlier inability to fully support SQL queries necessitate the production of several SQL engines to support the Key-Value databases in handling SQL queries. The need to deploy NoSQL without eliminating existing SQL data in databases is evident, because over 60% of large companies still use relational databases as of 2021 [13]. Nevertheless, many of the NoSQL databases today now have capabilities for handling complex queries like relational databases (SQL databases). Research such as [14]–[16] have shown that as relational database table increases, the retrieval times tend to slow down. Although, indexing reduces the retrieval time but does not completely eradicate this increase in time. To mitigate this problem, in this paper, we present NOSD, which is a NoSQL over SQL framework. Unlike Zidian [17], in which BaaV model is deployed at SQL level, NOSD uses Block as a Value (BaaV) representation in the NoSQL module, which is built over the SQL module. The BaaV model represents relations in a keyed block formats k, B; where k is a key to a block of values B of partial tuples in a schema. Contrary to systems of SQL-over-NOSQL such as Fast Scans [18], Spanner from Google [19], Apache Hive [20] [21], SparkSQL [22] and Zidian [17], NOSD is a NoSQL-over SQL framework.

II. Contributions

A major limitation of existing relational databases such as MySQL [23], MSSQL [24] and DB2, is that as the number of tuples in a relation increases, the retrieval time increases. Also, their inability to scale horizontally and efficiently store data of different forms (such as videos, audio, and text) necessitated the use of NoSQL database. However, due to the prevalent use and ease of deployment of relational (i.e., SQL) databases, many large corporations still make use of relational databases for their day-to-day activities [13] due to their structuredness and ACID (Atomicity, Consistency, Isolation and Durability) properties. Thus, in this paper, contrary to SQL over NoSQL models such as Zidian [17], Spanner [19] and F1 [12], we propose a system called NoSQL over SQL Block as a Value Database (NOSD) to mitigate the effects of increase in response time as relational database grows in tuples, and to leverage on the powerful features of NoSQL in terms of data retrieval, nodes distribution, horizontal scaling, and ability to store heterogeneous data.

A. NOSD Features Contributions

The main features contribution of this study are:

1) Using the BaaV strategy to model NoSQL database over existing large relational database with the aim reducing data saving and retrieval time.
2) Extracting and replicating data from SQL to NoSQL database autonomously and at configurable intervals.
3) Enabling structural and automatic saving of videos and audio alongside textual data in relational database by autonomously converting videos and audios into base64 text encoding.

B. Procedural Contribution

Towards actualizing the specified features contribution, this paper proposes two algorithms, the NOSDRelpica Algorithm (i.e., Algorithm 1) that replicates data based on access rates and nodes availability on the NoSQL. NOSDReplica algorithm is a modified version of the SBU dynamic replication algorithm [25]. However, we designed and implemented

NOSDReplica to replicate data from a central database to different NoSQL datastores (i.e., nodes within a cluster on a single server or multiple server). This is different from SBU because SBU only considers data saved in a file and SBU is primarily designed to replicate data on servers in different locations. Furthermore, our second proposed algorithm called KeyedBlockDocument algorithm (i.e., Algorithm 2) handles how old data in the NoSQL nodes are replaced when a NoSQL node capacity is full. This is invoked in the event of failure at the NoSQL automatic nodes allocation for data replacement.

III. Problem Statement

Given a Key-block database defined as $\overrightarrow{R}<P,Q>$ pair relation (i.e., key block pair) mapped as (k,B) where k is a tuple over attributes P of $\overrightarrow{R}$, and B is a set of tuples over the attributes Q; also, given a relational database $\overrightarrow{D}$ of $\overrightarrow{R}<P,Q>$, (k,B) is a set of keyed blocks over $\overrightarrow{R}$ of $\overrightarrow{D}$. Hence, the task of NOSD is to effectively map $\overrightarrow{R}<P,Q>$ to (k,B) with automatic data replication and integration for efficient time saving, data saving and retrieval, using NoSQL over SQL databases.

For instance, for an SQL database-query Q, NOSD detects the super key, k from Q and in parallel, checks the distributed nodes of key-block instance $\overrightarrow{D}$, preserving the properties of $\overrightarrow{R}$ and accurately retrieving Q on time less than what would have been done by $\overrightarrow{R}$ without instance $\overrightarrow{D}$. This is in addition to NOSD providing an application/network-based data pulling layer (API) for executing such a query.

This paper is structured as follows: We discussed related works in section IV. In Section V, the proposed NOSD is explored, while implementation and experimentation are discussed in section VI. Section VII features conclusion and future work.

IV. Related Work

Over the years, several systems have been proposed based on SQL-over-NoSQL [2], [4], [12], [17], [26]–[29]. These systems aim at using the SQL layer, which is capable of handling SQL queries on top of NoSQL databases. The SQL-over-NoSQL framework is used to tap the distributive features of NoSQL without losing the SQL functionalities. The Key Value architecture was primarily used for this purpose [17]. The Spanner system [19] leveraged on the key value model and supported the distributed transaction processing feature of NoSQL. Spanner is built on a full-scale Key Value architecture in which relations are represented as a table containing key and the corresponding values. This is based on Google's Bigtable system [11]. In this architecture, a table is a relation of key-value (KV) pairs, otherwise known as Table as a Value (TaaV). Successive research works after the introduction of BigTable [11], and Spanner as a SQL over NoSQL layer [19], include many open-source solutions. These include systems such as MyRocks [30] produced by Facebook [31]. MyRocks [30] is aimed at optimizing the space and write operation by using SQL over NoSQL paradigm. The SQL part of the implementation of MyRocks is based on MySQL database and InnoDB [31]. This is because Facebook uses MySQL to manage many Petabytes of users' data including comments, messages, and pictures, as this is the initial primary database management system used by Facebook [31]. Another system developed with the same KV architecture is the CockroachDB [29], which has the distributive property of performing several concurrent transactions by partitioning tasks between nodes. It has an SQL layer to support working with relational data and handling complex SQL queries [29]. An in-dept evaluation of these Key-Value systems shows that they perform efficiently with Online Transaction Processing (OLTP) tasks, however, they cannot efficiently scale through most Online Analytical Processing (OLAP) [18]. This is due to the simplicity that comes with the design of the Key-Value systems. Although, OLAP requires high locality and compact data representation, which most Key Value systems architecture lacks [32]. In contrast to the above points, our system, NOSD is a NoSQL over SQL model and majorly aims to improve data retrieval time, data replication efficiency and ability to handle unstructured data without data loss.

V. Proposed NOSQL over SQL Block a Value Database System (NOSD)

The proposed NOSD system is an integration of both SQL and NoSQL technologies. It builds NoSQL layer on top of existing SQL database. Leveraging on the NoSQL ability, we were able to configure Block as a Value model, allow efficient data replication for existing SQL database system migration to NoSQL, and improve retrieval time. Our proposed data replication and migration algorithm NoSDReplica is presented and explained below after presenting NOSD system architecture and Keyed Value block representations (See system architecture in fig 1).

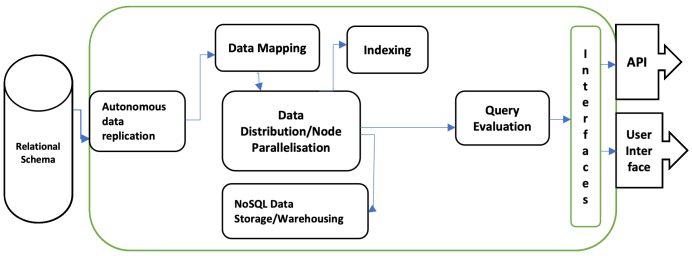

Fig. 1. NOSD System Architecture

A. Keyed Block Representation

In NOSD, at the NoSQL end, the Keyed Block k,B is achieved by denoting the block B, consisting of n tuples as a single value k. This is similar to the key value store (KV) with the exception that more than one tuple may be mapped to a key in our Keyed block approach and several relations could make up a block.

Problem 2: BaaV Representation of Schema Instance
Given a Schema C of customer. Containing relations: (a) Customer_Details, (b) Billing_Address and (c) Address (d) Product (e) Customer_Orders (f) Order below, identify the

super key k and represent an instance of the schema in BaaV : Customer_Details(**customer_id**,customer_name,customer_email, customer_phone)

Billing_Address(**customer_id**,**address_id**, **dateTime_from**, dateTime_to)

Address(**address_id**, address_line1, city, province, country, post_code)

Product(**product_id**,product_type_code, product_price)

Customer_Orders(**order_id**,customer_id,customer_pay ment_method_id, order_status_code, date_order_placed, date_order_paid)

Order(**order_id**, **product_id**, quantity, comments)

Solution 2: An instance of Key-Block representation of C
Iterating through the primary keys in the given schema C, the super key k that leads to all other relations in the schema is the primary key: $customer\_id$. The $BaaV$ representation of C is provided in the figure 2 below. Also, an example entry of an instance of C in a document-based NoSQL format and a tabular representation of an instance of C, both using BaaV model are provided in solution 2 and the table in figure 3 below.

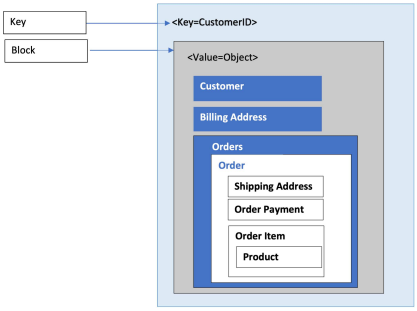

Fig. 2. BaaV (Keyed-Block) Representation of Customer Schema

Solution 2:
An example of an order entry into the above NoSQL document store for a single customer is presented below.
The index creation is mapped to the customer_id for uniqueness to avoid data redundancy associated with NoSQL databases, and all corresponding foreign keys' relations in the customer relation are stored along with the customer. (i.e., for this instance, the key is the customer_id and the block consist of all other attributes of the customer relation, foreign keys related relations/tables: Billing_Address, Orders, Order, Shipping_Address, Order_Payment, Order_Item and Product).

a) : {
“customers.createIndex()”:[
{ “customer_id”:1 },
{ “unique”:true }
],
“customers.insertOne()”:[
{ “customer_id”:“customer_id”,
“customer_name”:“testing name”,
“customer_email”:“a@m.com”,
“billing_address”:[
{
“address_id”:“*****”,
“dateTime_from”:“2021/08/01 10:56:23”,
“dateTime_to”:“2022/07/29 23:59:99”,
“Address”:[
{
“address_line1”:“Sunset Avenue”,
“city”:“Windsor”,
“province”:“Ontario”,
“country”:“Canada”,
“post_code”:“N9B 3P4” }] }],
“Orders”:[
{
“Order”:[
{
“order_id”:“jduuie83”,
“product_id”:“9jjeu38”,
“customer_payment_method_id”:“ek3ij”,
“order_status_code”:“P”,
“date_order_placed”:“2021/08/01 10:56:23”,
“date_order_paid”:“2021/08/01 10:59:23”,
“quantity”:5,
“comments”:“This is just a sample order entry” }] }, {
”Order”:[
{
“order_id”:“testorder2”,
“product_id”:“testproduct2”,
“customer_payment_method_id”:“visa”,
“order_status_code”:“P”,
“date_order_placed”:“2021/07/01 10:50:23”,
“date_order_paid”:“2021/08/20 10:59:23”,
“quantity”:5,
“comments”:“This is just a second sample order entry” }] }
] }] }

A tabular representation of the above document-based NoSQL Key-Block instance entry using BaaV is given in Figure 3 below:

customers.createIndex()

customer_id	unique
1	
	true

customers.insertOne()

customer_id	customer_name	customer_email	billing_address			Orders			
						Order			
			address_id	dateTime_from	dateTime_to	order_id	product_id	customer_payment_method_id	order_status_code
customer_id	testing name	a@m.com	*****	2021/08/01 10:56:23	2022/07/29 23:59:99	jduuie83	9jjeu38	ek3ij	P
						order_id	product_id	customer_paymentmethod_id	order_status_code
						testorder2	testproduct2	visa	P

Fig. 3. Keyed-Block View of Instance in Schema C of Problem 2

B. Data Replication Algorithm

For the autonomous data replication, we proposed and implemented a dynamic data replication algorithm called NOS-

DReplica. The fundamental principle behind this algorithm is to create a replica data from the relational database to the NoSQL database node (note: a node is a single point of storage on a NoSQL cluster, while a cluster consists of one or more nodes) with a high rate exceeding a predefined threshold. The threshold is used to distinguish popular relations within the SQL database, based on their access history. The input data to the NOSDReplica algorithm is a SQL database relations' access history, and the threshold is used to distinguish popular schema tables.

Algorithm 1 NOSDReplica(AH,threshold)

Input :access history of database tables (AH), threshold (T)

Output :set of Top-K relations replicated on the NoSQL database

Intermediate :A list containing information of popular relations for the individual databases sorted in Descending order

```
1: t ⇐ GET_TIME()
2: L ⇐ SORT_DEC(AH, threshold)
3: for all relation r ∈ L do
4:     f ⇐ r.relationID
5:     p ⇐ PARENT(r.nodeID)
6:     while p ≠ Root do
7:         if EXIST_IN(f, p) then
8:             UPDATE_CTIME(f, p, t)
9:             break
10:        end if
11:        if AVAILABLE_SPACE(p, t) ≥ SIZE(f) then
12:            REPLICATE(f, p, t)
13:            break
14:        end if
15:        p ⇐ PARENT(p)
16:    end while
17: end for
```

C. Steps in the Proposed NOSDReplica Algorithm

The inputs of the algorithm are the tables containing records of access to the different relations in the database and the threshold that is used to distinguish popular relations.

1) To track the creation time of the replicated data, the current time is recorded in t at the beginning of the algorithm execution.
2) The method $SORT - DEC$ is invoked to scan the database' relations access history (AH) and select the tables which number of access values are greater or equal to the threshold.
3) L contains the information of the popular relations; these relations are to be replicated on suitable nodes on the NoSQL database.
4) For each relation r in L, NOSDReplica gets the associated $relationID$ of the relation in the SQL database and the parent node p of the NoSQL database in steps 4 and 5.
5) For each relation r in L, NOSDReplica gets the parent node p of the NoSQL database.
6) The while loop of lines $6-16$ of NOSDReplica decides if the replica data should be created and where to take care of available space management of nodes in the NoSQL database.
7) If a replica already exists in the NoSQL database store, then there is no need to replicate this relation.
8) The creation time of the replica in 7 is updated to the current replication time by the function $UPDATE\_CTIME(f, p, t)$.
9) After updating C_TIME, NOSD breaks the while loop and process the next relation r in L.
10) If the replica of f does not exist in p in step 7 above, and the available space in the NOSQL database node is large enough for the relation coming from the SQL module, function $REPLICATE(f, p, t)$ is invoked to create new replica of f in p, and the creation time of the new replica will be set to t (Step in the line 12)

Then, the algorithm quits the while loop, otherwise, p is updated to point to the parent node and the while loop is repeated.

a) Remarks 1: Each node on the NoSQL server cluster has a storage limitation and which could be fully occupied by the replicated data. If the free storage of the NoSQL replica node is less than the new incoming replica, most NoSQL implementations have handled automatic node allocation for incoming data in such scenarios. However, in the failure of such, we defined an alternative method for removable replicas techniques similar to [25]. In our implementation, removable replicas are defined as replica created before the current replication time, and the number of access count is less than current threshold and not currently in use/pined by any NoSQL node taken the parent node p into consideration. This is defined in remarks 2 below:

b) Remarks 2: Let D_p be the set of all replica nodes in the NoSQL database with parent node p and the set of removable replicas be $\hat{D}_p$

$$\hat{D}_p = \left\{ d \| d \in D_p, \text{d is created before the current session time and d is currently unpinned} \right\} \quad (1)$$

The above condition is to avoid the deletion of recently created replica and avoid interruption of ongoing data replication process in the case of failure of the NoSQL automatic node space reallocation.

c) Remarks 3: At a time t, the available space of a replica node is the maximum storage that the particular NoSQL node can provide for new replicas. In the case of the automatic replica node allocation failure, the space must be checked before creating new replicas. For a particular node,

the available space is the addition of its free space and the space occupied by the removable replicas $\hat{D}_p$. Denoting the available space on a NoSQL parent node p as $AS(p)$ and the free space at time t as $FS(p)$.

$$AS(p) = FS(p) + \sum_{d \in \hat{D}_p} size(d) \quad (2)$$

Using the equation 2 above, the function $AVAILABLE\_SPACE(p,t)$ in NOSDReplica obtains the available space of parent node p at the current replication time t to determine the removable replicas. This implies that the function $REPLICATE(f,p,t)$ will only be invoked if available storage on the destination replica node is greater or equal to the size of the incoming relation from the SQL database.

Algorithm 2 KeyedBlockDocument Algorithm

NOSDDocRep (f,t)

Input :SQL relations from step 4 of NOSDReplica (Algorithm 1), time t

Output : Document store on the NoSQL database

1: $t \Leftarrow GET\_TIME()$
2: **for** all relation $r \in f$ **do**
3: $PK \Leftarrow f.relationID$
4: $FK \Leftarrow f(f.foreignKeys)$
5: **while** $numberOf(FK) \neq 0$ **do**
6: **if** $EXIST\_IN(DK, FK)$ **then**
7: break
8: **end if**
9: $DK \Leftarrow FK$
10: $WRITE(FK, DocK)$
11: **end while**
12: **end for**

D. Steps in the Proposed NOSDDocRep Algorithm

The input data is the set SQL relations passed from line 12 of the NOSDReplica algorithm.

1) The current data replication time is tracked with time t.
2) For all the relations identified to be replicated in Algorithm 1, line 3 gets the primary key (PK) of the relation and line 4 gets all the foreign keys (FK) that exists in the relation r in f.
3) DK is a sub document key used to identify a relation within a block within the NoSQL document store.
4) The while loop checks through all the foreign key in a particular relation r in f, then creates the relations for those FK in the same document. As represented in the example shown in figure 2 above.
5) Lines 6 and 7 check if the FK relations have already been written in the document and exit the while loop for that FK, if true.
6) Line 9 keeps track of the foreign keys that have been written inside DK and line 10 writes all the FK relations inside the document to keep the block intact.
7) Steps 2 to 6 are repeated for all the relations in f.

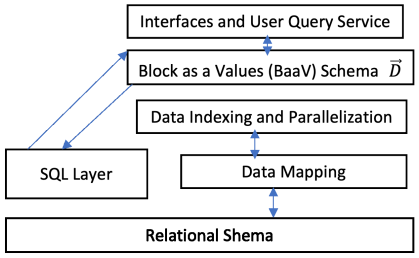

Fig. 4. NoSQL over SQL Layers Representation

VI. EXPERIMENTAL DESIGN

We implemented our proposed NoSQL Over SQL Database model ($NOSD$), with two other existing Benchmark NOSQL data replication and keyed block systems of Zidian [17] and Aggregate Bottom Up (ABU) [25] algorithm. This was used to evaluate the performance of our proposed NOSD system, in comparison with others using the same system configuration to test the three systems. Firstly, we deployed the NOSD system on real life dataset; the US air carrier's dataset from the US Department of Transportation, Bureau of Transportation Statistics [33], which consists of flight on-time performance data, with records from 1995 to 2021, and the UK Department for Transport anonymized MOT dataset [34]. We used the same dataset as Zidian [17]. Likewise, we also used simulated data with the same configuration of ABU [25] and test our system in terms of retrieval, update and create time, including the data replication time for ABU [25] and NOSDDoCRep (Algorithm 2)

A. Dataset and Implementation

Using the same dataset as Zidian to perform the experiment, we first transform the datasets i.e., UK MOT [34] and the US Air carrier data [33] and import it into the relational databases. We implemented NOSD using Java programming language and the SQL module of the experiment with two different SQL database management systems (MySQL and MSSQL). Furthermore, we implemented the NOSQL module using Elasticsearch [8] and Apache Cassandra [5], as discussed in section 3, remarks-1 of this paper. Elasticsearch automatically handles the storage node distribution and parallelization. However, to allow the experimentation and proof of concept that our proposed NOSDDocRep algorithm (i.e., Algorithm 1 above) works in the event of failure of this, or system malfunctioning, we specify the number of nodes inside the Elasticsearch configuration file (elasticsearch.yml file). The indexing module of the NOSD is handled by the Elasticsearch inbuilt indexing engine for shards and its enriched Lucene tailored query technology.

B. NoSQL Node Parallelization

In our implementation, each index is split into smaller elements (called shards). These shards are distributed across multiple nodes of our Elasticsearch datastore (NoSQL database).

The system is configured to withstand failures by specifying a number of primary shards with the respective number of replicas in the Elasticsearch configuration file. This ensures data integrity when the primary shard fail and increases performances of the NoSQL. This is because, replica shards handle search requests on Elasticsearch databases.

C. Autonomous Data Replication

In the implementation of our project, we designed a module that automatically discovers database structure/schema and creates this schema using Java Persistent API (JPA) with Java Spring Framework. Also, we developed a crone job using the Java programming language (i.e., using SpringBoot Scheduler class). The crone job has a configuration file to specify the time interval of data replication/pulling of the data in the database. This can be specified to any desired time interval for replication.

D. Interfaces and API

We implemented an application programming interface (API) for other applications to have the ability to query the NoSQL over SQL system. The developed framework provides an easy-to-use layer for any enterprise applications to easily integrate and post their query. The implemented Elasticsearch is built on Apache Lucene, which is a powerful library that provides indexing and search technology with advanced tokenization technology [35]. Furthermore, our system provides a user-friendly interface for data visualization.

E. EXPERIMENTAL RESULTS

Figures 5, 6, 7, and 8 below present the results of our experiments. These graphs show the results and comparison for retrieval, update and create times, accuracy for get request, and data replication time respectively, using the metrics proposed by [36], [37].

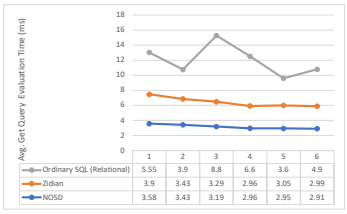

Fig. 5. Avg. Get Query Evaluation Time (ms)

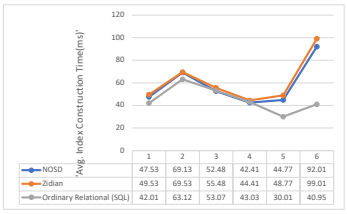

Fig. 6. Avg. Index creation time (ms)

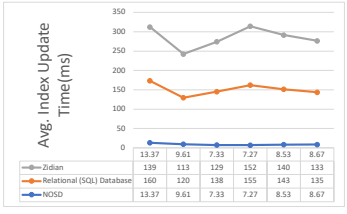

Fig. 7. Average index update time (ms)

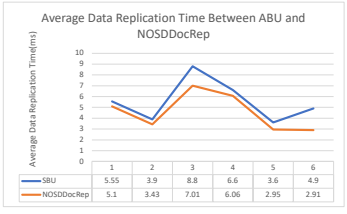

Fig. 8. Average Data Replication Time(ms) Between ABU and NOSDDocRep

VII. Conclusion and Future Work

NoSQL databases have now become one of the most used database systems in most bigdata scenarios. This is due to their ability to store unstructured or semi-structured and heterogeneous data, lesser data retrieval time and ability to be scaled horizontally with no experience of down time. This leads to a huge adoption in the industry and research environment involving big data. In this paper, we presented our NOSD model, a middleware to improve data retrieval and saving techniques in large datasets scenario in which the time cost of data saving, and retrieval increases as relational databases grows. Unlike existing systems which proposed completely migrating databases to NoSQL, after which an SQL layer is built on top of the NoSQL to handle queries, our system proposes sustaining the existing relational databases with its data. However, building an autonomous and integrate-able data replication system over most of the existing legacy large relational databases mitigates possibility of systems shutdown for existing solutions. Also, it reduces saving and retrieval time of relational databases. Most importantly, the proposed framework also provides a mechanism of using the merits associated with NoSQL databases. This is without the need to shutdown existing relational databases. Finally, our experimental result shows that using NoSQL-over-SQL highly outperforms ordinary SQL model and SQL-over-NoSQL models. We aim to perform future research in this direction, particularly since the most notable downfalls of NoSQL database is the inability to handle atomic transactions. Likewise, we aim to direct further studies on developing and deploying pattern mining algorithms on the NoSQL-over-SQL models/framework.

References

[1] Metaverse, "Facebook company info," 10 2021. [Online]. Available: https://about.fb.com/company-info/.

[2] D. Giuseppe, D. Hastorun, M. Jampani, G. Kakulapati, A. Lakshman, A. Pilchin, S. Sivasubramanian, P. Vosshall, and W. Vogels, "Dynamo: Amazon's highly available key-value store," *SIGOPS Oper. Syst. Rev.*, vol. 41, no. 6, p. 205–220, 2007. [Online]. Available: https://doi.org/10.1145/1323293.1294281

[3] HBase, "Apache hbase," Dec. 2021, available Online, accessed 2021-09-23. [Online]. Available: http://hbase.apache.org/.

[4] R. Nishtala, F. Hans, G. Steven, K. Marc, L. Herman, L. Harry, M. Ryan, P. Michael, P. Daniel, S. Paul, S. David, T. Tony, and V. Venkat, "Scaling memcache at facebook," in *10th USENIX Symposium on Networked Systems Design and Implementation*, ser. NSDI '13, 2013. [Online]. Available: https://research.facebook.com/publications/scaling-memcache-at-facebook/

[5] A. Cassandra. (2020) Apache cassandra, open source nosql database, manage massive amounts of data, fast, without losing sleep. Available online, accessed 2020-05-10. [Online]. Available: https://cassandra.apache.org/index.html.

[6] N. Ruflin, H. Burkhart, and S. Rizzotti, "Social-data storage-systems," in *Databases and Social Networks*, ser. DBSocial '11. New York, NY, USA: Association for Computing Machinery, 2011, p. 7–12. [Online]. Available: https://doi.org/10.1145/1996413.1996415

[7] H. Moditha, A. Alberto, V. Jovan, and Z. Esteban, "A cost model for random access queries in document stores," *The VLDB Journal*, vol. 30, no. 4, pp. 559–578, 2021.

[8] B. Elasticsearch. (2020) Elasticsearch: The heart of the free and open elastic stack. Available Online, accessed 2022-01-20. [Online]. Available: https://www.elastic.co/elasticsearch/.

[9] MongoDB-Inc. (2021) Mongodb; build faster build smarter. Available Online. [Online]. Available: https://www.mongodb.com/

[10] J. Dean and S. Ghemawat, "Mapreduce: Simplified data processing on large clusters," *Commun. ACM*, vol. 51, no. 1, p. 107–113, 2008. [Online]. Available: https://doi.org/10.1145/1327452.1327492

[11] 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, 2008. [Online]. Available: https://doi.org/10.1145/1365815.1365816

[12] J. Shute, R. Vingralek, B. Samwel, B. Handy, C. Whipkey, E. Rollins, M. Oancea, K. Littlefield, D. Menestrina, S. Ellner, J. Cieslewicz, I. Rae, T. Stancescu, and H. Apte, "F1: A distributed sql database that scales," *Proc. VLDB Endow.*, vol. 6, no. 11, p. 1068–1079, 2013. [Online]. Available: https://doi.org/10.14778/2536222.2536232

[13] ScaleGrid. (2021) 2019-database trends – sql vs. nosql, top databases, single vs. multiple database use. [Online]. Available: https://scalegrid.io/blog/2019-database-trends-sql-vs-nosql-top-databases-single-vs-multiple-database-use/

[14] A. Tomasic and H. Garcia-Molina, "Caching and database scaling in distributed shared-nothing information retrieval systems," *SIGMOD Rec.*, vol. 22, no. 2, p. 129–138, 1993. [Online]. Available: https://doi.org/10.1145/170036.170063

[15] S. E. Robertson and E. Kanoulas, "On real-time ad-hoc retrieval evaluation," in *Proceedings of the 35th International ACM SIGIR Conference on Research and Development in Information Retrieval*, ser. SIGIR '12. New York, NY, USA: Association for Computing Machinery, 2012, p. 1119–1120. [Online]. Available: https://doi.org/10.1145/2348283.2348498

[16] O. Chaparro and A. Marcus, "On the reduction of verbose queries in text retrieval based software maintenance," in *Proceedings of the 38th International Conference on Software Engineering Companion*, ser. ICSE '16. New York, NY, USA: Association for Computing Machinery, 2016, p. 716–718. [Online]. Available: https://doi.org/10.1145/2889160.2892647

[17] Y. Cao, W. Fan, and T. Yuan, "Block as a value for sql over nosql," *Proc. VLDB Endow.*, vol. 12, no. 10, p. 1153–1166, 2019. [Online]. Available: https://doi.org/10.14778/3339490.3339498

[18] M. Pilman, K. Bocksrocker, L. Braun, R. Marroquín, and D. Kossmann, "Fast scans on key-value stores," *Proc. VLDB Endow.*, vol. 10, no. 11, p. 1526–1537, 2017. [Online]. Available: https://doi.org/10.14778/3137628.3137659

[19] J. C. Corbett, J. Dean, M. Epstein, A. Fikes, C. Frost, J. J. Furman, S. Ghemawat, A. Gubarev, C. Heiser, P. Hochschild, W. Hsieh, S. Kanthak, E. Kogan, H. Li, A. Lloyd, S. Melnik, D. Mwaura, D. Nagle, S. Quinlan, R. Rao, L. Rolig, Y. Saito, M. Szymaniak, C. Taylor, R. Wang, and D. Woodford, "Spanner: Google's globally distributed database," *ACM Trans. Comput. Syst.*, vol. 31, no. 3, 2013. [Online]. Available: https://doi.org/10.1145/2491245

[20] J. Camacho-Rodríguez, A. Chauhan, A. Gates, E. Koifman, O. O'Malley, V. Garg, Z. Haindrich, S. Shelukhin, P. Jayachandran, S. Seth, D. Jaiswal, S. Bouguerra, N. Bangarwa, S. Hariappan, A. Agarwal, J. Dere, D. Dai, T. Nair, N. Dembla, G. Vijayaraghavan, and G. Hagleitner, "Apache hive: From mapreduce to enterprise-grade big data warehousing," in *Proceedings of the 2019 International Conference on Management of Data*, ser. SIGMOD '19. New York, NY, USA: Association for Computing Machinery, 2019, p. 1773–1786. [Online]. Available: https://doi.org/10.1145/3299869.3314045

[21] Y. Huai, A. Chauhan, A. Gates, G. Hagleitner, E. N. Hanson, O. O'Malley, J. Pandey, Y. Yuan, R. Lee, and X. Zhang, "Major technical advancements in apache hive," in *Proceedings of the 2014 ACM SIGMOD International Conference on Management of Data*, ser. SIGMOD '14. New York, NY, USA: Association for Computing Machinery, 2014, p. 1235–1246. [Online]. Available: https://doi.org/10.1145/2588555.2595630

[22] M. Armbrust, R. S. Xin, C. Lian, Y. Huai, D. Liu, J. K. Bradley, X. Meng, T. Kaftan, M. J. Franklin, A. Ghodsi, and M. Zaharia, "Spark sql: Relational data processing in spark," in *Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data*, ser. SIGMOD '15. New York, NY, USA: Association for Computing Machinery, 2015, p. 1383–1394. [Online]. Available: https://doi.org/10.1145/2723372.2742797

[23] Oracle-Corporation. (2021) Mysql database service. Available Online. [Online]. Available: https://www.mysql.com/

[24] Microsoft. (2021) Sql server on-premises or in the cloud. Available Online. [Online]. Available: https://www.microsoft.com/en-ca/sql-server/sql-server-2019

[25] M. Tang, B.-S. Lee, C.-K. Yeo, and X. Tang, "Dynamic replication algorithms for the multi-tier data grid," *Future Gener. Comput. Syst.*, vol. 21, no. 5, p. 775–790, 2005.

[26] D. F. Bacon, N. Bales, N. Bruno, B. F. Cooper, A. Dickinson, A. Fikes, C. Fraser, A. Gubarev, M. Joshi, E. Kogan, A. Lloyd, S. Melnik, R. Rao, D. Shue, C. Taylor, M. van der Holst, and D. Woodford, "Spanner: Becoming a sql system," in *Proceedings of the 2017 ACM International Conference on Management of Data*, ser. SIGMOD '17. New York, NY, USA: Association for Computing Machinery, 2017, p. 331–343. [Online]. Available: https://doi.org/10.1145/3035918.3056103

[27] A. Pavlo and M. Aslett, "What's really new with newsql?" *SIGMOD Rec.*, vol. 45, no. 2, p. 45–55, 2016. [Online]. Available: https://doi.org/10.1145/3003665.3003674

[28] J. Tatemura, O. Po, W.-P. Hsiung, and H. Hacigümüş, "Partiqle: An elastic sql engine over key-value stores," in *Proceedings of the 2012 ACM SIGMOD International Conference on Management of Data*, ser. SIGMOD '12. New York, NY, USA: Association for Computing Machinery, 2012, p. 629–632. [Online]. Available: https://doi.org/10.1145/2213836.2213917

[29] R. Taft, I. Sharif, A. Matei, N. VanBenschoten, J. Lewis, T. Grieger, K. Niemi, A. Woods, A. Birzin, R. Poss, P. Bardea, A. Ranade, B. Darnell, B. Gruneir, J. Jaffray, L. Zhang, and P. Mattis, "Cockroachdb: The resilient geo-distributed sql database," in *Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data*, ser. SIGMOD '20. New York, NY, USA: Association for Computing Machinery, 2020, p. 1493–1509. [Online]. Available: https://doi.org/10.1145/3318464.3386134

[30] S. Dong, A. Kryczka, Y. Jin, and M. Stumm, "Rocksdb: Evolution of development priorities in a key-value store serving large-scale applications," *ACM Trans. Storage*, vol. 17, no. 4, 2021. [Online]. Available: https://doi.org/10.1145/3483840

[31] Y. Matsunobu, S. Dong, and H. Lee, "Myrocks: Lsm-tree database storage engine serving facebook's social graph," *Proc. VLDB Endow.*, vol. 13, no. 12, pp. 3217–3230, 2020. [Online]. Available: https://doi.org/10.14778/3415478.3415546

[32] L. Braun, T. Etter, G. Gasparis, M. Kaufmann, D. Kossmann, D. Widmer, A. Avitzur, A. Iliopoulos, E. Levy, and N. Liang, "Analytics in motion: High performance event-processing and real-time analytics in the same database," in *Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data*, ser. SIGMOD '15. New York, NY, USA: Association for Computing Machinery, 2015, p. 251–264. [Online]. Available: https://doi.org/10.1145/2723372.2742783

[33] B. o. T. S. United States Department of Transportation, Bureau of Transportation Statistics. (2021) Summary statistics origin and destination airport. Available Online. [Online]. Available: https://www.transtats.bts.gov/ONTIME/OriginDestination.aspx

[34] United-Kingdom-Transport-Department. (2021) Department for transport anonymised mot tests and results. Available Online. [Online]. Available: https://ckan.publishing.service.gov.uk/dataset/anonymised_mot_test

[35] R. Kuc and R. Marek, *Elasticsearch Server Create a Fast, Scalable, and Flexible Search Solution with the Emerging Open Source Search Server*, 1st ed. Birmingham: Packt Pub., 2013.

[36] A. S. Butt, A. Haller, and L. Xie, "A taxonomy of semantic web data retrieval techniques," in *Proceedings of the 8th International Conference on Knowledge Capture*, ser. K-CAP 2015. New York, NY, USA: Association for Computing Machinery, 2015. [Online]. Available: https://doi.org/10.1145/2815833.2815846

[37] E. Oren, R. Delbru, M. Catasta, R. Cyganiak, H. Stenzhorn, and G. Tummarello, "Sindice.com: A document-oriented lookup index for open linked data," *Int. J. Metadata Semant. Ontologies*, vol. 3, no. 1, p. 37–52, 2008. [Online]. Available: https://doi.org/10.1504/IJMSO.2008.021204