# A Comparative Study of Cloud Data Portability Frameworks for Analyzing Object to NoSQL Database Mapping from ONDM's Perspective

Salil Bharany[1*], Kiranbir Kaur[2], Safaa Eltayeb Mohamed Eltaher[3],
Ashraf Osman Ibrahim[4*], Sandeep Sharma[5], Mohammed Merghany Mohammed Abd Elsalam[6]
Department of Computer Science and Engg, Lovely Professional University, Phagwara , India[1]
Department of Computer Engg. And Technology, Guru Nanak Dev University, Punjab, India[2]
Prince Sattam Bin Abdulaziz University, College of Computer Engineering and Sciences, Department of software engineering[3]
Creative Advanced Machine Intelligence Research Centre, Faculty of Computing and Informatics, Universiti Malaysia Sabah, 88400 Kota Kinabalu, Sabah, Malaysia[4]
Department of Computer Engg. and Technology, Guru Nanak Dev University, Punjab, India[5]
Faculty of Computer Science and Information Technology, Alzaiem Alazhari University, Khartoum North 13311, Sudan[6]

***Abstract*—Cloud computing revolves around storing and retrieving data in a portable manner. However, practical data portability across multiple Database-as-a-service (DBaaS) cloud data stores is challenging. This becomes even more complicated when data needs to be migrated between different types of data storage, such as SQL and NoSQL databases. NoSQL databases have gained significant popularity among developers due to their ability to provide high availability, fault tolerance, and scalability, making them suitable for managing big data in large-scale infrastructures. However, the varied data models in NoSQL databases make it difficult to migrate or port data among data repositories. Object to NoSQL database mappers (ONDMs) solves this problem. However, only a few ONDMs are available for C#.NET development, and the ONDM market used in Java development could be more stable. To address this issue, we propose building a middleware solution using the .NET framework to support cloud data portability, leveraging the capabilities of ONDMs. In this study, we evaluate several frameworks and compare them to our suggested middleware solution through empirical research. Our middleware solution can perform open network data management (ONDM) and object-relational mapping (ORM).**

***Keywords*—NoSQl; Portability; Cloud; middleware; platform as a service; platform services**

## I. INTRODUCTION

Cloud Computing has become a pre-eminent paradigm for hosting modern software systems, and the database layer is the most valuable and extensive layer of a software system [1-2]. The heterogeneity of cloud service providers (CSPs), the data stores they offer, and the software systems pose substantial impedance while developing an approach for cloud data migration. However, the database-related requirements of modern applications call for polyglot persistence [3]. An application that leverages persistent polyglot databases is considerably more arduous to design and implement than an application using just one backend [7]. The overhead of configuration, deployment and maintenance keeps increasing with each DB used. This makes implementing polyglot persistence quite tricky without the detailed know-how of involved DBs.

### A. Data Models

The use of more than one data model within a single system has become a usual practice for modern application development [4]. The cloud computing paradigm supports both of the types of models for data storage:

*1) Relational (SQL) data models:* Relational data models are schema-based, store data in the form of tables, and maintain ACID (Atomicity, Consistency, Isolation, and Durability) properties. They prevailed since the 1970s when E.F Codd proposed they orchestrate the data into tables (or relations) consisting of rows (also known as records/tuples) and columns (also known as attributes). Each table has a unique key called the primary key which identifies each row and may have a foreign key that represents a primary key of some other table for cross-reference [5].

*2) NoSQL data model:* The acrostic for NoSQL means "Not Only SQL" rather than completely against the traditional relational databases (DBs), as is commonly misunderstood. Carl Strozzi1, in 1998 first time, used the term "NoSQL" to name his open-source relational DB "Strozzi NoSQL". This DB used APIs with several plugins and libraries instead of using SQL for accessing the data. NoSQL data stores are distinguished in the following four categories based on data and query models, and persistence design [6]:

*a)* Key-Value DBs represents a model based on keys-values and are easy to implement. These are suitable to store session information, user profiles, or storing shopping cart data. Examples include Redis, Voldemort, Riak.

*b)* Document DBs in which semi-structured documents are stored in JSON format (XML and YAML formats are also supported), are suited for big data storage and better query performance. Examples are MongoDB, Apache CouchDB, and Cosmos DB.

*c)* Column family DBs represent a model for storing and processing huge amounts of data, which is distributed over various machines without rigid consistency. Examples are Apache Cassandra, HBase, and Apache Accumulo.

*d)* Graph DBs which are suitable for storing relationships between entities. Examples are Neo4j, OrientDB, and AllegroGraph.

Each mired data store possesses its specific benefits. Relational DBs are favored if the data to be stored concerns financial transactions, as these DBs abide by transactional properties. On the other hand, the evolution of the Internet, social networking sites, and Cloud Computing has disputed the domination of relational DBs as the only selection of DBMS. Various considerations like prices, the volume of data, and the speed at which the data is being generated as well as consumed, dictate how and where the storage and management of the data.

## II. Object to NoSQL Database Mappers (ONDMS)

A single application may need heterogeneous DBs for the various types of requirements, for example [7]:

- For User Sessions: Redis is best suited for quick access for reads and writes without having to be durable.
- For financial data and reporting: RDBMS (Relational database management system) is required as this kind of data needs transactional updates. Moreover, data would better fit in a tabular structure.
- Product catalog: MongoDB is best as it supports a lot of reads and infrequent writes.
- Analytics and user activity logs: Cassandra can better handle a high volume of writes on multiple nodes.

And there may be many more types of requirements for application data, leading to the selection of appropriate DBs. Therefore, the application may require the simultaneous use of different DBs (relational as well as NoSQL, called Polyglot persistence) on different cloud platforms and also a data migration from one kind of data store to another, of similar type (SQL to another SQL) or dissimilar type (SQL to/from NoSQL). As there is a looming dearth of standardized query languages, it poses an adverse technical lock-in while building applications against the native interfaces of NoSQL data stores [8]. The solution to evade vendor lock-in caused due to selecting a particular database technology is to leverage Object-NoSQL Database Mappers (ONDMs). ONDMs offer a uniform abstraction interface for heterogeneous NoSQLs. ONDM frameworks decouple applications from database specifics and provide data portability [14].

- Handling the conversion of objects to the relational data model and vice versa.
- Managing persistence to the destination DB.
- Providing software developers with a uniform data access interface to store and query objects programmatically.

Our middleware's architectural design is a Repository pattern. Repositories are classes that contain the logic necessary to access data sources. They consolidate common data access functions, improving maintainability and separating database access from the domain model layer. Because of strong typing, the code that must be implemented to use our middleware is simplified. This allows us to concentrate on the business logic rather than the data access plumbing. ONDMs are developer-centric and let the developers carry the application abstractions without having to be cognizant of the database and use these databases without expecting a level of expertise in those [5]. The benefits of using ONDMs include simplifying porting of an application to other NoSQL data stores and database interoperability as well as polyglot persistence [9]. There are ONDMs called Multi Data Store Mappers supporting multiple NoSQL data stores and ONDMs called Single Data Store Mappers supporting only a particular system [10].

- Kundera2: It is a capable JPA-based object-datastore mapping library that greatly cuts down the programming efforts needed to perform CRUD operations on NoSQL data stores.
- Spring Data3: It is an umbrella project that alleviates the use of data access technologies, namely relational and NoSQL, Map Reduce frameworks, as well as cloud-based data services. It provides a Spring-based data access programming model that preserves the special features of the underlying data stores [18].
- DataNucleus4: We also tried another industry-ready ONDM framework, 'Data Nucleus' for the implementation but faced the following difficulties [19].
- Mongo - The library exposed by DataNucleus and JavaMongo lib had clashing classes in the same classpath. This created issues while building the application [20].
- MySQL - Framework was enhancing model classes after running the maven enhancement step, as mentioned in the documentation. Still, it was not able to detect them at the time of attempting to persist the object [23].

So, we dropped DataNucleus for the comparison with our proposed middleware. Our selection of databases to be implemented tried to find the databases that are not only quite prevalent and ripe but also have great applicability in specific fields [22] [24]. MongoDB and Cassandra are the most prominent NoSQL databases in the market. MongoDB produces high throughput, and Cassandra supports horizontal scalability [25]. MongoDB is supported by almost all ONDMs, followed by Cassandra. Our selection of databases tried to find the databases that are not only quite prevalent and ripe but also have great applicability in specific fields [26].

## III. Implementation Technology - .Net Core

We have created a custom data model based on Twitter data set to benchmark the proposed middleware and the other frameworks. Some of the key features of .NET Core platform are highlighted below.

- The. NET Core is a new version of Microsoft. NET Framework is a free, open-source, general-purpose programming platform. It is a cross-platform framework that works on Windows, macOS, and Linux [28].
- The .NET Core Framework may be used to create a variety of applications such as mobile, desktop, online, cloud, IoT, machine learning, microservices, and so on.
- The .NET Core is developed from the bottom up to be a modular, lightweight, fast, and cross-platform Framework [30]. It offers the essential capabilities necessary to run a basic.NET Core app. Other functionalities are available as NuGet packages, which you may add to your application as needed [30]. As a result, the.NET Core program performs faster, has a smaller memory footprint, and is easier to maintain.

It is a new platform that is gaining traction in the industry, but there are no ONDM frameworks available for it. This is one of the main reasons to opt .NET Core framework for our middleware implementation [31]. Although Microsoft provides its ORM for .NET Core named ENTITY FRAMEWORK, it is strictly an ORM (that means it is only for RDBMS mapping to objects and not for NoSQLs). Table I presents that most of the ONDMs available are for Java language [32] [34]. Although individual NoSQL database drivers [14] or wrappers are available for the C# language, there are no mature ONDM frameworks for C#.

TABLE I. ONDM FRAMEWORKS SUPPORTED BY DIFFERENT OBJECT-ORIENTED PROGRAMMING LANGUAGES

| OOPL | ONDM Frameworks | Inactive Frameworks |
|---|---|---|
| Java | Apache Gora, Kundera, Data Nucleus, EclipseLink, Eclipse JNoSQL, Spring Data, Hibernate OGM, GORM | Java |
| Python | KEV, pyDAL | NA |
| JavaScript | JS Data | Resourceful |
| Node.JS | Thinodium, Bass, Waterline, JS Data | JugglingDB, Cleverstack, Node Docu- |
| PHP | Lithium, Yii framework, Doctrine | KO3-NoSQL, Vork |
| C#.NET | | Slazure, Charisma |
| Scala | Lift | Activate Framework |

Definitions

- Poly DB: The frameworks support multiple types of database systems (i.e., relational and NoSQL).
- Wrapper: The library is a wrapper around a database system; this means it might not be an object-mapper (e.g. driver). It just interfaces with the application, but it may not have the capability of object mapping [35].
- ODNM: The framework has objected to NoSQL database mapping capabilities.
- Strict OR/NDM: The framework strictly has either ONDM or ORM mapping functionality [36].

OUR proposed middleware [1] is all POLYDB (as it is supporting multiple DBslike SQL Server, MongoDB, and Cassandra) as well as ORM and ONDM.

- Mongo - The library exposed by DataNucleus and JavaMongo lib had clashing classes in the same classpath. This was creating issues while building the application.
- MySQL - Framework was enhancing model classes after running the maven enhancement step, as mentioned in the documentation. Still, it was not able to detect them at the time of attempting to persists the object.
- SpringData - The Challenge was to integrate with the DBs only. Enough documentation is available to make the application ready.
- OBDApi - There was code in the application that was creating issues while building the application. We needed to remove the unnecessary pieces to make it work.
- Kundera SQL - No major challenge apart from integrating with the DB and adding code for our use case. Analyzed the code to identify how it will work.

In the paper [11], the author introduces and defines the term "ONDM (Object-NoSQL Datastore Mapper) is a framework to facilitate the storage and retrieval of persistent objects in NoSQL datastore systems". In [6] it has been studied state-of-the-art ORMs and dedicated ONDMs that are capable of handling disparate NoSQL data stores. This work studies the performance of the abstraction layers for NoSQL data stores with an emphasis on the runtime performance impact. In the paper [9] also, the authors provide a performance evaluation of various ONDM frameworks [9]. The main difference to our work is that we perform a more comprehensive performance evaluation and contemplate with academic frameworks [12] and [13]). Table IV reveals that most of the ONDMs are available for Java language. [14] Although individual NoSQL database drivers or wrappers are available for C# language, there are no mature ONDM frameworks for C#. We compare analytically our proposed middleware with the academic frameworks CDPort and ODBAPI as well as industry-ready ONDMs viz. Kundera and Spring Data. The proposed middleware relieves the user of these saddles of dealing with the specific APIs. All he needs to do is change the connection string in the application configuration file (appsettings.json).

### A. Our Contribution

We offered a middleware solution created in.NET to allow cloud data portability, which corresponded to the capability of ONDMs in terms of performance and functionality. In this study, we compare our suggested middleware solution with the other frameworks and conduct an empirical evaluation of each of the frameworks. This paper demonstrates that our middleware can serve as both an ORM and an ONDM. Some of the core contributions have been mentioned below.

*1)* We discussed various data models used for storing data.

*2)* ONDMs (Object to NoSQL data mappers (Academic and Industrial)) have been also discussed.

*3)* Available ONDMs are mostly developed for Java developers and to the best of us knowledge, no ONDM is available for .NET developers. We developed .NET ONDM in previous paper and validated it by comparing it with other ONDMs (two academic and two industrial).

*4)* The results we got after experimentation proved our middleware have comparable performance with respect to the above said ONDMs.

## IV. Related Work

Data portability has been taken up by researchers in the literature, where it is considered a mechanism that enables the migration of data as well as enhances interoperability across multiple heterogeneous cloud platforms [15]. While working towards data portability among clouds, the requirement of converting one type of database into another rises owing to the numerous types (SQL and NoSQL) and data models (key-value, columnar, document-oriented, and graph) of the databases offered by the providers. One solution is the mapping of objects to NoSQLs which essentially corresponds to the functionality of ODNMs. We have also proposed a solution to support cloud data portability in [1], which maps the objects into cloud NoSQLs (MongoDB and Cassandra). Other solutions include [16]:

- SQL fication of NoSQL databases with SQL-like wrappers which generally provide various features corresponding to those of classical relational database query language while retaining a grammar identical to that of SQL
- Meta-model approaches which abstract from the data models by identifying the common concepts in different NoSQL solutions' data models.

The rivet of the middleware is that the application, the database, and the platform basic services (such as message queues, email, and SMS service) are so loosely coupled that each of these can be ported to any of the clouds (supported by the middleware) without having to rewrite much code in the application. Although a plethora of research efforts has been done towards data portability, our proposed work relates to [12, 13]. To the best of our knowledge, data migration among clouds (where data previously stored in one cloud is shifted/copied to another cloud) is not much covered in the literature. Some notable research works towards data portability are discussed in Table II.

TABLE II. Analysis of the Related Work

| Ref. | Solution approach | Work Done |
|---|---|---|
| [17] | Design patterns | This paper proposed an effective design pattern method for shifting data from a columnar DB (HBase) to a graph DB (Neo4J) and vice versa. However, this work appears to be only a suggestion, as no implementation work is provided in this or any subsequent publications published by the author (to the best of our knowledge). |
| [18] | Service Delivery Cloud Platform (middleware) and common API | This paper proposed a cloud middleware infrastructure called SDCP and offered a common API to deliver three cloud services viz. Storage, DB, and Notification service. Using JPA (Java Persistence API) methods, they provided abstraction for DB access. |
| [19] | CSAL (Cloud Storage Abstraction Layer) | An abstraction layer is also provided here in order to give common storage abstraction to diverse cloud providers. The layer also creates a namespace that programmers may utilize to support blobs, tables, and queues. |
| [20] | Abstraction layer | This paper presented a mediation-based approach to integrate SQL and NoSQL DBs to retrieve data from either of them. Moreover, their proposed extended SQL can execute join queries as well. |
| [21] | GUI tool, point to point the translator | This thesis work implemented a Graphical User Interface tool that alleviates the data migration from relational DB to NoSQL document data stores. |
| [22] | NoSQLayer | This paper focused on the automatic translation of SQL queries to NoSQL by proposing a framework called "NoSQLayer". The focus here is in query execution rather than data migration. |
| [23] | Middleware, Common interface | This paper described a subset of SQL commands for accessing NoSQL DBs with the help of proposed middleware which uses C# and ANTLR for parsing SQL. |
| [24] | Heuristic-based | This paper presented a 2-phase transformation mechanism from relational DB to HBase. The first phase transformed relational schema to HBase schema, and the second phase expressed the relationships of two schemas as a set of nested schema mappings. |
| [12] | Common data model | This paper focused on the challenge of data portability and proposed a framework called "CDPort" which is equipped with tools for conversion, transformation, and data exchange among disparate data storage models. |
| [25] | Model Transformation | The authors developed a tool called ERWin HAWK for model transformation and accomplishing data migration. Their work reckoned the query characteristics of relational DB, prepared a model transformation algorithm that extracts the ER model and description tags from relational DBs, and based on these model transformations, migrated the data into MongoDB. |
| [26] | Metamodelling approach | This paper proposed SOS (Save Our Systems) tool which provides a uniform Application Programming Interface based on meta-modeling to support heterogeneous NoSQL data stores. |
| [27] | Model-Driven Engineering | This article addressed the issue of data portability and offered a system called "CDPort" that includes tools for data conversion, transformation, and interchange across heterogeneous data storage types. |
| [28] | Model-Driven Engineering | This paper leveraged MDE to harmonize the differences among the storage models of two prominent PaaS namely GAE and Azure. The authors created a DSL (Domain Specific Language) to support portable applications. They also addressed the issue of data portability of the applications. |
| [29] | Data Adapter | This paper proposed a "Data Adapter" system to provide data synchronization which uses both relational and NoSQL DBs at the same time. Their mechanism offered three modes for query in DB: Blocking Transformation mode (BT), Blocking Dump mode (BD mode), Direct access mode (DA mode). |
| [30] | Metamodel (Hegira4Cloud) | This paper proposed an architecture called "Hegira4Cloud" which provides an intermediate metamodel for Columnar DBs especially. The |

| | | |
|---|---|---|
| | | authors also focused on the fault tolerance feature of the NoSQL portability of Big Data applications. |
| [31] | Metamodel | This dissertation work proposed a metamodel which is used to convert data to different formats via an intermediate state (especially JSON). |
| [32] | MetaModel | This article extracted system knowledge using an ontology called KDM (Knowledge Discovery meta-model) and utilized many pre-defined patterns to help users through the application migration from one cloud platform to another. |
| [33] | Map Reduce framework | This paper proposed a framework called "JackHare" based on Hadoop and HBase which includes an SQL query compiler, JDBC driver as well as MapReduce method to process the unstructured data of NoSQL DB. The data from relational DB as a source is stored on Hadoop and HBase and is processed with SQL queries. |
| [34] | Unified REST API | This paper presented a unified REST API called Open-PaaS-DataBase (ODBAPI) to interact with the different data stores uniformly. |
| [35] | Cloud data patterns | This paper presented pattern-based application refactoring to accomplish the various migration scenarios of data migration and data portability. |

The author in [14] also proposed the common programming interface but the system does not comply with cloud data store specifications as our proposed system does. The reason is that it leverages the XML in conjunction with SQL for modeling the system [15]. CDPort provided a common data model to handle different cloud storage services through a common API whereas, in our middleware, each datastore has its data model which enables it to detect the associated datastore of the user-defined model. While it may seem that a unified data model is better than using different data models for each datastore but when implemented both the approaches are fine and yield similar results. By using different data models, our middleware detects and converts the objects to their associated data store supported queries/models with more precision. We ought to improve on it in terms of implemented clouds and implemented data storage services. Moreover, a thorough examination of the source code depicts that it is prone to SQL injection as it is not using parameterized queries. We are manually implementing the adapters for each database and if there is any change in the API of the database, we must update the adapter manually. But the user using our middleware in his/her application does not need to change the source code to accommodate this update. He/she just needs to update the middleware package in his/her application.

In the paper [13], it includes more latency than our proposed middleware because the REST API server processes the request as follows:

*1)* The user's request goes to the REST API server.

*2)* REST API server processes the request and sends it to the cloud server.

*3)* The cloud server sends the response to the REST API server.

*4)* REST API server sends a response to the user's applicationHowever, in our proposed middleware [1], all the database related services are packaged within the user's application and hosted together with the user's application.

As NoSQLs are further of various types, it is not practical to develop a single query language. So, the proposed solution to this problem is to leverage the middleware to mitigate the requirement of accessing, storing, and migrating the data from and within the implemented DBs. If the proposed middleware is used while developing the cloud application, it extenuates the implementation details of all the supported DBs. The middleware supports homogenous SQL migration between different clouds, homogeneous NoSQL migration between different clouds, heterogeneous SQL to NoSQL migration in the same cloud, heterogeneous SQL to NoSQL migration between different clouds, heterogeneous NoSQL to NoSQL migration in the same cloud, and heterogeneous category NoSQL to NoSQL migration between the different clouds. The factors to be considered for switching the data store and for migrating the data include the heterogeneous categories of the source and target DBs (SQL and NoSQL). Even within the same category, there are different products available e.g., for SQL, there are MySQL and SQL Server and for document DB, there are MongoDB and Cassandra. NoSQLs further have another level of categorization as briefly discussed in previous section "Object to NoSQL Database Mappers (ONDMs)".

## V. The Proposed Middleware

Cloud portability is defined by [36] as "the ability of data and application components to be easily moved and reused regardless of the choice of cloud provider, operating system, storage format or APIs." Out of the categorized scenarios for cloud portability suggested in [37], only the third and fourth categories have been considered by the proposed middleware and this paper describes the benchmarking of the fourth category particularly.

- Virtual machine portability across cloud providers.
- Portability of virtual machines across cloud providers.
- Portability of applications in the context of Infrastructure as a Service (IaaS).
- Portability of PaaS apps.
- Data portability between cloud providers.

All the entities of the user models are stored as objects. To persist these objects in the appropriate data-store, the object's type is determined with the help of reflection (feature of C# language). A user-defined model is a class that inherits from a particular middleware meta-model base class (as we implemented a separate middleware meta-model corresponding to each type of the supported data store). Fig. 1 shows the decision making about the data store to be used by checking the middleware's meta-model class:

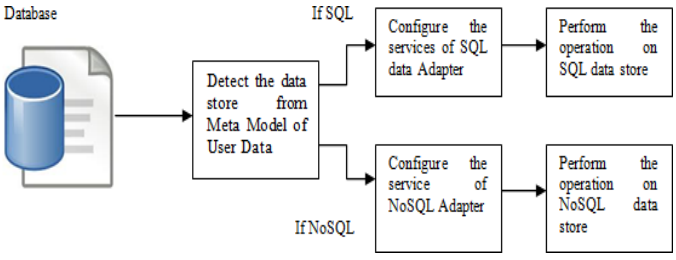


Fig. 1. Decision making procedure to select a data store for persistence.

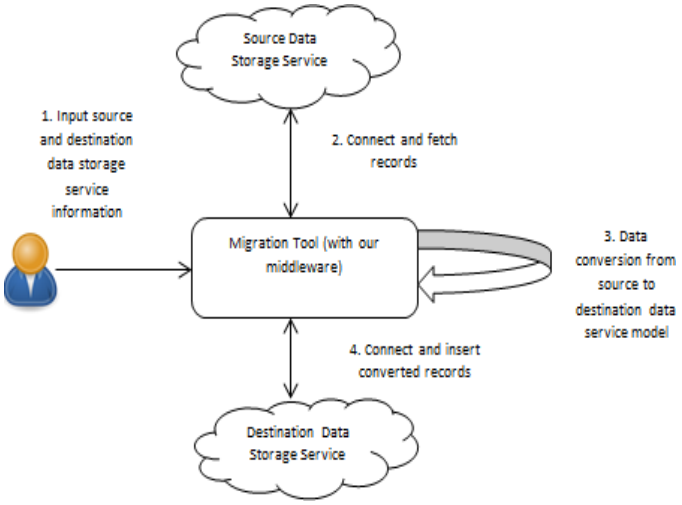


Fig. 2. Data transformation process of the middleware.

The middleware is designed to be extended to support other data stores also. Fig. 2 shows the data transformation process of the proposed middleware where the migration tool takes source and destination data storage service information as an input [37]. Then the connection with the source storage service is established to fetch the records and the tool converts the data from the source to the destination data model [38]. This converted data is inserted into the destination data storage service. A detailed description of the middleware implementation is given in [39]. It was observed that leveraging the middleware makes it quite easy for the user to achieve the data portability decreasing user's efforts greatly [40]. We have experimentally evaluated the industry ready ONDMs namely KUNDERA and Spring Data as well as academic ODNMs namely CDPort and ODBAPI against SQL and NoSQL (MongoDB and Cassandr data-stores [40]. The four candidate methodologies have been evaluated using Twitter dataset and implementing different migration scenarios. An experiment was carried out in the next section to determine the effectiveness of the migration [39]. During the assessment, three cloud platforms were employed (Google Cloud Platform, Microsoft Azure, and Amazon Web Services) [41] [42]. It is also claimed that the suggested middleware is interoperable with various PaaS providers.

## VI. Results and Evaluation

We compare our proposed middleware analytically with the academic frameworks viz. CDPort and ODBAPI as well as industry ready ONDMs viz. Kundera and Spring Data. We created a custom data model based on the Twitter data set to benchmark the proposed middleware and the other frameworks. We created two similar applications, one in Java language (to evaluate ODBAPI, Kundera, CDPort, and Spring Data) and another in C#.NET (to evaluate our middleware). Both these applications have minimal functionality to perform just the CRUD operations on the Twitter data set. The time taken to perform these operations using the applications is noted, and these values of readings are compared to know the efficiency of each of them. The experiments were executed on a system with configurations - 2 core machines with 4 GB RAM. The data in Table III was captured for three different workloads of 1000, 5000, and 10000 no. of tweets/records. In this paper we are using three types of scenarios mentioned below.

Three scenarios are:

*1)* SQL to/from NoSQL
*2)* One category of NoSQL to another category NoSQL
*3)* Even among different SQL data-stores or data stores of the same category NoSQL

In each experiment scenario, the following operations were performed on SpringData, Kundera, ODBAPI and CDPort:

*1)* Add records (tweets/records)
*2)* Get all records (tweets/records)
*3)* Update records (tweets/records)
*4)* Delete records (tweets/records)

- SpringData It is an enterprise-level ORM with solid developer support and easy integration. SpringData removes all DAO (Data Access Object) implementations. Only the DAO's interface must be defined explicitly. By extending the interface, we obtain all the normal DAO CRUD functions. This informs Spring Data to look for this interface and generate an implementation for it. The problem with Spring Data was merely integrating with DBs [43].
- Kundera - is a "Polyglot Object Mapper"(Single Application Using Multiple Data Storage Technologies) with a JPA interfaces [44]. It serves as a JPA Compliant mapping solution for NoSQL Datastores. After running our scenario, we observed that "Get All" for a lower number of records took more time as compared to "Get All" for a larger number of record [45] (we ran this scenario multiple times to conclude this).
- ODBAPI - This ORM is a unified REST-based API. This API enables to execute CRUD operations on relational and NoSQL data stores. There is no support for Cassandra, so we ran our scenario for MySQL and mongo [46]. There was code in the application that was creating issues while building the application, removed the conflicting code to make it work. For SpringData, Kundera, and OBDApi, we ran our scenario by connecting our test application with local DB instances. This helped us by realistically compare the framework performance by not considering network latencies (as compared to if integrated with Cloud DB).
- CDPort - The CDPort's API has been designed to hide the programmatic difference between the different SQL and NoSQL database systems [47]. It enables software developers to easily change their backend cloud- based data storage without the need to change the software code. They expose adapters for each cloud DB and thus client applications need to integrate with these adapters [48]. Thus, providing a clean way to integrate. It provides support for cloud DB. Thus, we have executed our scenario for Amazon RDS and MongoDB (Amazon Document DB) [49].

TABLE III. COMPARISON OF TIME TAKEN TO PERFORM DATABASE OPERATIONS BY THE MIDDLEWARE VS. OTHER FRAMEWORKS

| Databases & Operations Performed | Proposed Middleware (milliseconds) | | | KUNDERA (milliseconds) | | | ODBAPI (milliseconds) | | | Spring Data (milliseconds) | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **SQL** | **1000** | **5000** | **10000** | **1000** | **5000** | **10000** | **1000** | **5000** | **10000** | **1000** | **5000** | **10000** |
| **INSERT** | 2442 | 11208 | 20056 | 1310 | 4666 | 8458 | 392 | 1539 | 3536 | 3805 | 53203 | 189860 |
| **SELECT** | 34 | 175 | 281 | 413 | 72 | 86 | 7 | 29 | 37 | 10 | 177 | 76 |
| **UP- DATE** | 3060 | 16529 | 29047 | 2594 | 8081 | 14867 | 437 | 2087 | 4724 | 4794 | 88932 | 334545 |
| **DELETE** | 941 | 4630 | 9581 | 1315 | 4762 | 9359 | 533 | 1666 | 3334 | 3183 | 52341 | 178737 |
| **MONGO** | | | | | | | | | | | | |
| **INSERT** | 28 | 320 | 317 | 891 | 1238 | 2299 | 547 | 2000 | 3089 | 701 | 1995 | 4880 |
| **SELECT** | 33 | 74 | 132 | 487 | 36 | 71 | 19 | 83 | 237 | 82 | 131 | 229 |
| **UP- DATE** | 70 | 324 | 649 | 2709 | 7204 | 1397 | 821 | 13927 | 51926 | 525 | 1861 | 4116 |
| **DELETE** | 35 | 172 | 357 | 789 | 2966 | 5820 | 809 | 11624 | 42935 | 384 | 1551 | 3166 |
| **Cassandra** | | | | | | | | | | | | |
| **INSERT** | 149 | 647 | 1069 | 1220 | 2899 | 5835 | - | - | - | 1221 | 3640 | 5853 |
| **SELECT** | 2 | 3 | 4 | 332 | 143 | 192 | - | - | - | 413 | 1105 | 1981 |
| **UPDATE** | 1537 | 1813 | 2369 | 807 | 2021 | 3298 | - | - | - | 1405 | 2893 | 6423 |
| **DELETE** | 1241 | 1433 | 1963 | 672 | 1880 | 3499 | - | - | – | 797 | 2004 | 3931 |

For SQL databases, ODBAPI performed the best of all the frameworks which can be seen in readings of Table III and graph of Fig. 3. For the Cassandra database, INSERT and SELECT operations took the least time with our middleware [1], and UPDATE and DELETE operations took comparable time which can be seen in Fig. 4. The middleware proposed in this work can be considered as comparable to the two-industry ready ONDMs (Kundera and Spring Data) and academic framework (ODBAPI) [50]. For the Mongo database, our proposed framework performed exceptionally well as seen in Fig. 5. Comparison between different middleware can be seen in Table III. The data in Table IV was captured for three different workloads of 1000, 5000, and 10000 no. of tweets/records.

As the proposed middleware supports cloud data portability, another comparison is done with the CDPort framework which also supports cloud data portability. Except for the SQL INSERT operation, all other operations took lesser time with our middleware.

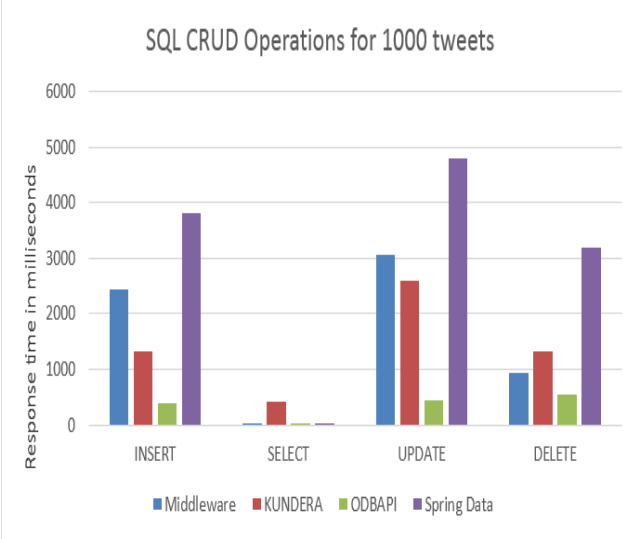


Fig. 3. Comparison of performance of frameworks for SQL CRUD operations on 1000 tweets.

TABLE IV. COMPARISON OF TIME TAKENUSSING DIFFERENT WORKLOADS

| AWS cloud | Proposed Middleware | | |
|---|---|---|---|
| SQL | 1000 | 5000 | 10000 |
| INSERT | 84577 | 452959 | 957680 |
| SELECT | 309 | 1180 | 2999 |
| UPDATE | 111278 | 499048 | 913818 |
| DELETE | 80317 | 440182 | 718930 |
| INSERT | 1361 | 6840 | 14290 |
| SELECT | 1300 | 4767 | 11549 |
| UPDATE | 1648 | 11002 | 13954 |
| DELETE | 680 | 3611 | 7982 |

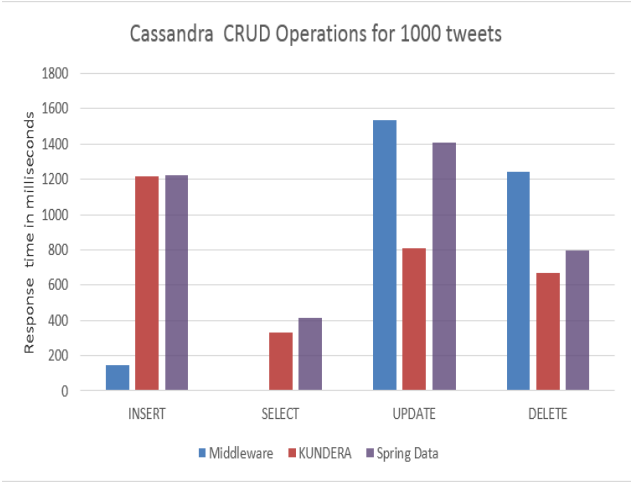


Fig. 4. Comparison of performance of frameworks for Cassandra CRUD operations on 1000 tweets.

The evaluation included performing CRUD operations on the Twitter data set with different workloads viz. 1000, 5000, and 10000 tweet/records through the proposed middleware, KUNDERA, Spring Data, ODBAPI, and CDPort frameworks as seen in reading of Table V. The total time taken to perform these operations were compared which depicted that middleware performs at par to all these frameworks [40] as can be seen in Fig. 6 and Fig. 7. We created two similar

applications, one in Java language (to evaluate ODBAPI, KUNDERA, CDPort, and Spring Data) and another in C#.NET (to evaluate our middleware). Both these applications have minimal functionality to perform just the CRUD operations on the Twitter data set. The time taken to perform these operations using the application is noted, and these values of readings are compared to know the efficiency of each of them [45-46]. We evaluated the impact of the ODNMs based on application runtime performance as response time is very crucial for the users' experience in the interactive modern applications. Also, different ONDMs have different runtime performance. Although adding ONDMs adds to the performance overhead [1], these provide the benefit of easy portability across disparate NoSQLs [38-43]. For Document DB, our middleware performed much better than CDPort which can be seen in Table IV.

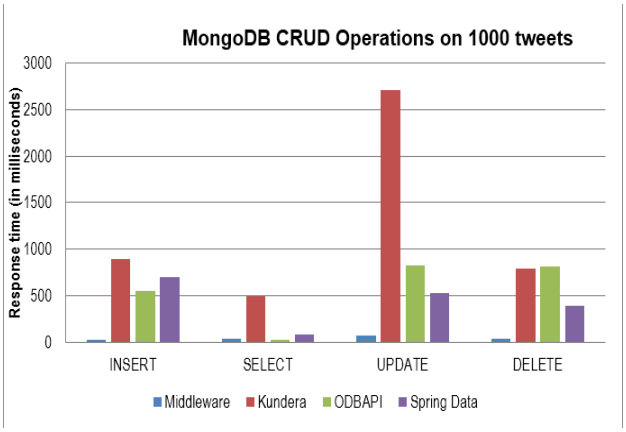


Fig. 5. Comparison of performance of frameworks for MongoDB CRUD operations on 1000 tweets.

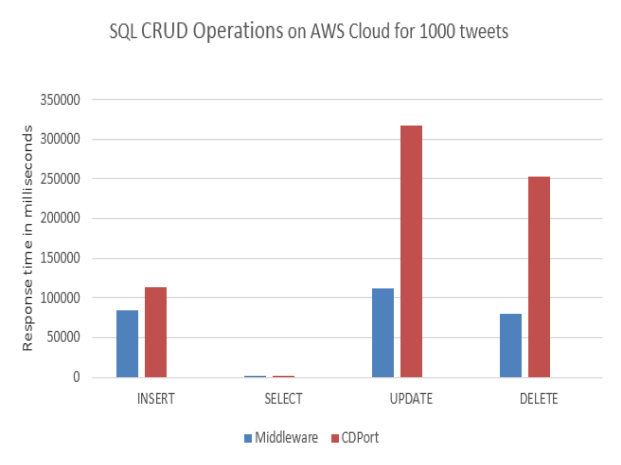


Fig. 6. SQL CRUD Operations on AWS.

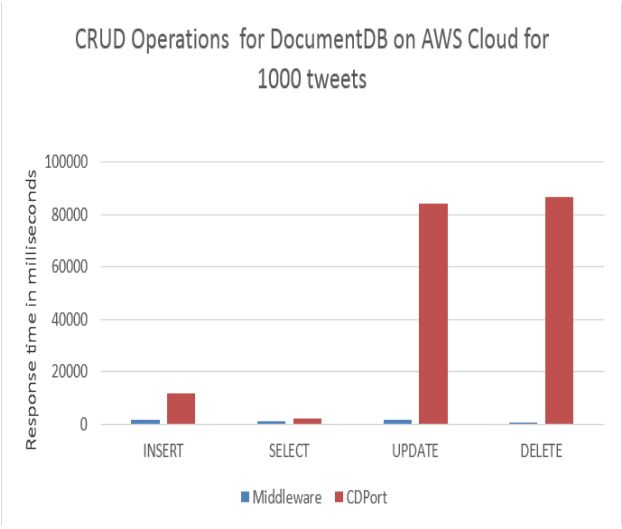


Fig. 7. CRUD operations for document DB.

TABLE V. COMPARISON WITH CDPORT

| AWS cloud | | CD Port | |
|---|---|---|---|
| SQL | 1000 | 5000 | 10000 |
| INSERT | 13939 | 139678 | 183158 |
| SELECT | 568 | 2886 | 4356 |
| UPDATE | 316903 | 1602408 | 2312081 |
| DELETE | 253408 | 1298309 | 1772904 |
| INSERT | 11648 | 12242 | 113997 |
| SELECT | 2032 | 9426 | 19406 |
| UPDATE | 83975 | 409502 | 972376 |
| DELETE | 86491 | 638225 | 982814 |

## VII. CONCLUSION AND FUTURE SCOPE

Compared to an application that only uses a SQL database, a combination of the graph, document, and column-based data stores will have a data access layer that is far more complicated and will require additional work. The developer's knowledge and experience are the primary factors that should be considered when selecting an application's database management system (DBMS). The vast majority of cloud service providers make available various services and application programming interfaces (APIs) that may be used to access and manage the services they offer. A problem with interoperability arises due to the variety of cloud services. Utilizing an intermediary abstraction layer or adhering to pre-existing standards is the recommended action for resolving this problem. This article focuses solely on the database migration process, utilizing a method known as "Object to NoSQL database mappers. Organizations may require transferring the system (software and database layer) among different providers. However, this article only discusses database migration (ONDMs). Evaluation of the effect of the proposed solution's implementation on response time and throughput is used to validate the solution's performance. In addition, the performance is evaluated in relation to other methods described in the published research and commercial solutions currently on the market. According to the findings, our strategy performs noticeably better than the other strategies. The work that will be done in the future will

involve adding support for more clouds and additional data stores that fall into other categories, such as graph and key-value stores.

## REFERENCES


[1] Kaur, S. Sharma, and K.S. Kahlon, "A middleware for polyglot persistence and data portability of big data paas cloud applications", *Computers, Materials & Continua*, vol. 65, no. 2, pp. 1625-1647, 2020

[2] M.H. Ellison, "Evaluating Cloud Migration Options for Relational Databases", 2017

[3] M. Schaarschmidt, F. Gessert, N. Ritter, "Towards automated polyglot persistence", in *Proc. Lect. Notes LNI), Proc. - Ser. Gesellschaft Fur Inform*, 241 ,73–82, 2015.

[4] J. Castrejón, G. Vargas-Solar, C. Collet and R. Lozano, "ExSchema: Discovering and maintaining schemas from polyglot persistence applications", in *Proc. IEEE Int. Conf. Softw. Maintenance*,Eindhoven, Netherlands, pp. 496–499, 2013.

[5] V. Abramova, and J. Bernardino, "NoSQL Databases: MongoDB vs Cassandra", In: Proc. *Int. Conf. Comput. Sci. Softw. Eng. ACM*, Porto, Portugal, pp. 14–22, 2013.

[6] U. Störl, M. Klettke, T. Hauf, S. Scherzinger, and Schemaless "NoSQL data stores - Object-NoSQL mappers to the rescue?", *in Proc. Lect. Notes Informatics (LNI), Proc. - Ser. Gesellschaft Fur Inform*, Bonn Gesellschaft für Informatik ,vol. 241, pp. 579–599, 2015.

[7] M. Fowler, and P. Sadalage, "The future is : NoSQL Databases", 2012.

[8] S. Bharany, S. Sharma, N. Alsharabi, E. Tag Eldin, and N. A. Ghamry, "Energy-efficient clustering protocol for underwater wireless sensor networks using optimized glowworm swarm optimization," *Frontiers in Marine Science, Frontiers Media SA, vol. 10,* 2023.

[9] V. Reniers, A. Rafique, D. Van Landuyt, and W. Joosen, "Object-NoSQL Database Mappers : a benchmark study on the performance overhead", *J. Internet Serv. Appl.* vol. 8, pp. 1–16, 2017.

[10] J. R. Lourenço, B. Cabral, P. Carreiro, M. Vieira, and J. Bernardino, "Choosing the right NoSQL database for the job: a quality attribute evaluation", *J. Big Data*., vol. 2, pp. 1–26, 2015.

[11] Reniers, V., Rafique, A., Van Landuyt, D. et al. Object-NoSQL Database Mappers: a benchmark study on the performance overhead. J Internet Serv Appl 8, 1 (2017). https://doi.org/10.1186/s13174-016-0052-x

[12] E. Alomari, A. Barnawi, and S. Sakr, "CDPort: A Portability Framework for NoSQL Datastores", *Arab. J. Sci. Eng*., vol. 40, pp. 2531–2553, 2015.

[13] R. Sellami, S. Bhiri, and B. Defude, "ODBAPI: A unified REST API for relational and NoSQL data stores", *in Proc. - 2014 IEEE Int. Congr. Big Data, BigData Congr*, Anchorage, AK, USA ,pp. 653–660, 2014.

[14] V. Reniers, D. Van Landuyt, A. Rafique, and W. Joosen, "Object to NoSQL Database Mappers ( ONDM ): A systematic survey and comparison of frameworks", *Inf. Syst*, vol. 85, pp. 1–20, 2015.

[15] A. Bansel, "Cloud based NoSQL Data Migration Framework to achieve data portability", *National College of Ireland, Dublin, Ireland,* 2015.

[16] F. Arcidiacono, "Avoiding CRUD operations lock-in in NoSQL databases: extension of the CPIM library", *Politecnico di Milano Computer*, 2015.

[17] M. N. Shirazi, H.C. Kuan, H. Dolatabadi, "Design patterns to enable data portability between clouds' databases", *in Proc ICCSA* , Salvador, Brazil ,pp.117–120 ,2012.

[18] L.A. Bastião Silva, C. Costa, J.L. Oliveira, A common API for delivering services over multi-vendor cloud resources, J. Syst. Softw. 86 ,2309–2317,2013.

[19] Z. Hill, M. Humphrey, CSAL: A cloud storage abstraction layer to enable portable cloud applications, *in Proc IEEE Int. Conf. Cloud Comput. Technol. Sci. CloudCom* , Indianapolis, IN, USA ,pp. 504–511. 2010.

[20] S. Bharany et al., "Energy efficient fault tolerance techniques in green cloud computing: A systematic survey and taxonomy," Sustainable Energy Technologies and Assessments, vol. 53. Elsevier BV, p. 102613, Oct. 2022. doi: 10.1016/j.seta.2022.102613.

[21] M. Mughees, Data Migration From Standard SQL to NoSQL, 2013.

[22] L. Rocha, F. Vale, E. Cirilo, D. Barbosa, F. Mourão, A framework for migrating relational datasets to NoSQL, *Procedia Comput. Sci*. 51, 2593–2602, 2015 .

[23] J. Rith, P.S. Lehmayr, K. Meyer-Wegener, Speaking in tongues: SQL access to NoSQL systems, *in Proc. ACM Symp. Appl. Comput.* , *New York, NY, USA* , pp. 855–857,2014.

[24] C. Li, Transforming relational database into HBase: A case study, *in Proc. 2010 IEEE Int. Conf. Softw. Eng. Serv. Sci. ICSESS*, Beijing, China, 683–687, 2010.

[25] T. Jia, X. Zhao, Z. Wang, D. Gong, G. Ding, Model transformation and data migration from relational database to MongoDB, in Proc. IEEE Int. Congr. Big Data, BigData Congr , San Francisco, CA, USA ,60–67 ,2016.

[26] P. Atzeni, F. Bugiotti, L. Rossi, Uniform access to non-relational database systems: in Proc. The SOS platform, Lect. Notes Comput. Sci , Berlin, Heidelberg.160–174 , 2012.

[27] A. Beslic, R. Bendraou, J. Sopena, J.Y. Rigolet, Towards a solution avoiding vendor lock-in to enable migration between cloud platforms, in Proc CEUR Workshop , MIAMI, FLORIDA, USA 1118 , 5–14, 2013.

[28] E.A.N. Da Silva, D. Lucrédio, A. Moreira, R. Fortes, Supporting multiple persistence models for PaaS applications using MDE: Issues on cloud portability, in *Proc. CLOSER* , Lisbon, Portugal, 331–342 ,2015 ,

[29] S. Bharany, S. Sharma, N. Alsharabi, E. Tag Eldin, and N. A. Ghamry, "Energy-efficient clustering protocol for underwater wireless sensor networks using optimized glowworm swarm optimization," Frontiers in Marine Science, vol. 10. Frontiers Media SA, Feb. 02, 2023. doi: 10.3389/fmars.2023.1117787.

[30] M. Scavuzzo, D.A. Tamburri, E. Di Nitto, Providing big data applications with fault-tolerant data migration across heterogeneous NoSQL databases, in *Proc.BIGDSE* , Austin, TX, USA ,26–32 ,2016.

[31] Curitiba, Data Migration between different data models of NoSql Databases, 2017.

[32] A. Bansel, H. Gonzalez-Velez, A.E. Chis, Cloud-Based NoSQL Data Migration, in *Proc.* Euromicro Int. Conf. Parallel, *Distrib. Network-Based Process* , , Heraklion, Greece ,224–231, 2016.

[33] W. C. Chung, H.P. Lin, S.C. Chen, M.F. Jiang, Y.C. Chung, JackHare: a framework for SQL to NoSQL translation using MapReduce, *Autom. Softw. Eng.* ,489–508, 2014 .

[34] R. Sellami, S. Bhiri, B. Defude, Supporting Multi Data Stores Applications in Cloud Environments, *IEEE Trans. Serv. Comput* ,59–71,2016.

[35] S. Strauch, V. Andrikopoulos, T. Bachmann, F. Leymann, Migrating application data to the Cloud using Cloud data patterns, in *Proc.* CLOSER , Aachen, Germany 36–46 ,2013.

[36] Z. Zhang, C. Wu, and D. W. L. Cheung, "A survey on cloud interoperability," ACM SIGMETRICS Performance Evaluation Review, *Association for Computing Machinery* , vol. 40, no. 4. pp. 13–22, 2013.

[37] G. C. Silva, L. M. Rose, and R. Calinescu, "Towards a Model-Driven Solution to the Vendor Lock-In Problem in Cloud Computing," in *Proc.International Conference on Cloud Computing Technology and Science.* IEEE, 2013. Bristol, UK, pp. 711-716, 2013.

[38] S. Bjeladinovic, "A fresh approach for hybrid SQL/NoSQL database design based on data structuredness," *Enterprise Information Systems*, vol. 12, no. 8–9 ,pp. 1202–1220, 2018.

[39] Ilin, D., & Nikulchev, E.V. Performance Analysis of Software with a Variant NoSQL Data Schemes. In *Proc. MLSD*, 1-5,2020. Moscow, Russia, 2020, pp. 1-5,

[40] A. AGGOUNE and M. S. NAMOUNE, "A Method for Transforming Object-relational to Document-oriented Databases," in *Proc. ICMIT. IEEE*, 2020. Adrar, Algeria, 2020, pp. 154-158,

[41] M. Li, J. Xu, and L. Han, "Multi-dimensional Analysis of Industrial Big Data Based JSON Document," in *Proc.IEEE Intl Conf on Parallel and Distributed Processing with Applications.IEEE*, Exeter, United Kingdom, ,1066-1073, 2020/

[44] Bharany, K. Kaur, S. Badotra, S. Rani, Kavita, M. Wozniak, J. Shafi, and M. F. Ijaz, "Efficient Middleware for the Portability of PaaS Services

Consuming Applications among Heterogeneous Clouds," *Sensors*, vol. 22, no. 13. MDPI AG, p. 5013, 2022.

[45] Nurhadi, R. B. A. Kadir, and E. S. B. M. Surin, "Evaluation of NoSQL Databases Features and Capabilities for Smart City Data Lake Management," in *Proc. Lecture Notes in Electrical Engineering. Springer Singapore*, South korea, pp. 383–392, 2021

[46] K. Kaur, S. Sharma, and K. S. Kahlon, "Towards a Model-Driven Framework for Data and Application Portability in PaaS Clouds," in *Proc. First International Conference on Sustainable Technologies for Computational Intelligence. Springer Singapore*, pp. 91–105, 2019.

[47] K. Kaur, S. Bharany, S. Badotra, K. Aggarwal, A. Nayyar, and S. Sharma, "Energy-efficient polyglot persistence database live migration among heterogeneous clouds," *The Journal of Supercomputing*, vol. 79, no. 1. Springer Science and Business Media LLC, pp. 265–294, 2022.

[48] Y.T. Liao, J. Zhou, C.H. Lu, S.C. Chen, C.H. Hsu, W. Chen, M.F. Jiang, Y.C. Chung, Data adapter for querying and transformation between SQL and NoSQL database, Futur. Gener. Comput. Syst. 65, 111–121,2016.

[49] E. M. Onyema et al., "A Security Policy Protocol for Detection and Prevention of Internet Control Message Protocol Attacks in Software Defined Networks," Sustainability, vol. 14, no. 19. MDPI AG, p. 11950, Sep. 22, 2022

[50] J. Roijackers, H.L.G. Fletcher, XML Query Processing: On Bridging Relational and Document-Centric Data Stores, *in Proc*. *LNCS* , Berlin, Heidelberg, pp. 135–148, 2013.