Computers, Materials & Continua CMC, vol.65, no.2, pp.1625-1647, 2020

A Middleware for Polyglot Persistence and Data Portability of Big Data PaaS Cloud Applications

Kiranbir Kaur[1, *], Sandeep Sharma[1] and Karanjeet Singh Kahlon[2]

Abstract: Vendor lock-in can occur at any layer of the cloud stack-Infrastructure, Platform, and Software-as-a-service. This paper covers the vendor lock-in issue at Platform as a Service (PaaS) level where applications can be created, deployed, and managed without worrying about the underlying infrastructure. These applications and their persisted data on one PaaS provider are not easy to port to another provider. To overcome this issue, we propose a middleware to abstract and make the database services as cloud-agnostic. The middleware supports several SQL and NoSQL data stores that can be hosted and ported among disparate PaaS providers. It facilitates the developers with data portability and data migration among relational and NoSQL-based cloud databases. NoSQL databases are fundamental to endure Big Data applications as they support the handling of an enormous volume of highly variable data while assuring fault tolerance, availability, and scalability. The implementation of the middleware depicts that using it alleviates the efforts of rewriting the application code while changing the backend database system. A working protocol of a migration tool has been developed using this middleware to facilitate the migration of the database (move existing data from a database on one cloud to a new database even on a different cloud). Although the middleware adds some overhead compared to the native code for the cloud services being used, the experimental evaluation on Twitter (a Big Data application) data set, proves this overhead is negligible.

Keywords: Cloud computing, platform as a service, middleware, polyglot persistence, SQL, NoSQL, data migration tool, Twitter data set.

1 Introduction

Big Data and Cloud computing go hand-in-hand and the cloud services exist because of Big Data [InsideBigData (2019)]. Cloud computing has garnered a lot of attention as well as fostered competition in the industry during the last decade. It consists of three service models viz. Infrastructure as a Service (IaaS), Platform as a service (PaaS), and Software as a Service (SaaS). PaaS has a booming market. The PaaS model offers application

[1] Department of Computer Engineering and Technology, Guru Nanak Dev University, Amritsar, 143001, India.

[2] Department of Computer Science and Applications, Guru Nanak Dev University, Amritsar, 143001, India.

[*] Corresponding Author: Kiranbir Kaur. Email: kiran.dcse@gndu.ac.in.

Received: 14 May 2020; Accepted: 11 June 2020.

developers with hardware and software tools commonly needed for application development over the Internet. Moreover, it provides a plethora of technology resources with minimal configuration work resulting in the speedy development of applications [Yasrab and Gu (2016)]. Gartner [Gartner (2019)] says "As of 2019, the total PaaS market contains more than 360 vendors." and expects that "from 2018 to 2022, the market will double in size and that PaaS will be the prevailing platform delivery model moving forward." However, on the flip side, it's not easy for the PaaS users to migrate the services and the generated data to another competing PaaS. The applications on PaaS platforms store their data in various types of databases (such as relational and NoSQL) according to the characteristics of the data. NoSQL technology is the powerhouse to implement Big Data applications. Cloud computing provides the infrastructure to store this Big Data. Generally, a cloud environment offers limited data stores for the data of the deployed applications. But there may arise some situations where these specific data store models could not cater to the application's all requirements. We propose a middleware as a solution in this paper which can alleviate the technical intricacies dealing with the above scenarios. It supports multiple data stores and exposes a simple API to the user while all the complexities of the data portability and conversion are managed by the abstraction layer of the middleware.

Section 2 describes the background related to data portability and polyglot persistence. Section 3 discusses the related work to data portability. Section 4 presents the technical design of the proposed middleware. The evaluation and the experimental tests are presented in Section 5 before the conclusion is drawn in Section 6.

2 Background

2.1 Definition of "data portability across clouds"

Before defining the term Data portability, a definition of Cloud portability is required. Petcu [Petcu (2011)] stated that the portability of applications at the PaaS level along with a successful data migration should require a minimum amount of application rewriting. For cloud interoperability also, the application should be able to span multiple cloud providers, facilitating data exchange as well as data portability. Chetal et al. [Chetal, Peterson, Wallace et al. (2011)] also stated that data portability is a prerequisite for switching cloud providers. These statements infer the importance of Data portability in the context of Cloud portability.

Data Portability is regarded as data reuse which entails a quick and easy transfer of data among applications [Kostoska, Gusev and Ristov (2015)].

We define "Data Portability to include both the switching of the data store within the same cloud platform (providers) or among different cloud platforms as well as the migration of the data from the original data store to the destination data store." Also, the terms "data stores" and "databases (DBs)" have been used interchangeably in this paper.

2.2 Polyglot persistence-simultaneous use of relational and NoSQL DBs

Catering the needs of data storage of applications with relational requirements as well as non-relational features implies the usage of both kinds of DBs (SQL and NoSQL). This requires accessing and interacting with their disparate APIs. This scenario in which a cloud

application uses multiple data stores of relational and NoSQL types, is called Polyglot Persistence. This kind of persistence levies ponderous efforts by the application developer as he/she needs to be acquainted with these APIs while developing code for the application. However, Polyglot persistence, rather than only NoSQL DBs was perceived as the future of the data storage in the enterprise by Fowler et al. [Fowler and Sadalage (2012)]. Based on their data storing techniques NoSQL DBs can be categorized into 4 types:

- Key-Value stores-They store each item as a key-value pair. These are considered the simplest type of NoSQL. Examples include Redis, Voldemort, Riak.
- Document oriented-They store and manage data in the form of documents. Documents can store key-value pairs similar to the key-value stores. Examples are MongoDB, Apache CouchDB, and Cosmos DB. These DBs support JSON, XML, and YAML formats.
- Columnar-Similar to the relational DBs, Column-oriented DBs also support the concept of rows and columns but does not mandate to define the columns. Also, these allow storing data sets as columns in contrast to the relational DBs which store data sets as rows of a table. Examples are Apache Cassandra, HBase, and Apache Accumulo.
- Graph-When data to be stored can be represented as graphs or networks such as social networks, the graph DBs can be used. The nodes of the graph represent conceptual objects and are connected by the lines called edges. Examples are Neo4j, OrientDB, and AllegroGraph.

3 Related work

After exploring the literature about the cloud portability, two prominent approaches come up as the solutions to tackle the challenge of application portability and data portability [Gonidis, Paraskakis and Kourtesis (2012)]:

Standardization. If all the cloud service providers embrace the standards, then the developers would be able to create the applications agnostic of the particular cloud environment or even the application developed for one environment would be effortlessly ported over another environment. Several organizations have taken an endeavor to institute users' trust for various cloud computing services by proposing standards (OVF, OCCI, UCI, CIMI, CDMI, TOSCA, CAMP etc.) associated with the operation of cloud services. Standardization of services among different cloud providers is least likely to happen; users must consider other approaches to facilitate interoperability and portability of applications among disparate clouds. Moreover, the focus of the most active cloud standards is on the IaaS instead of the PaaS level [Kaur, Sharma and Kahlon (2017)].

Intermediation. Besides standardization, another alternative to facilitate portability issues is intermediation which detaches the application's development from any platform's APIs and supported formats [Korte, Challita, Zalila et al. (2018); Gonidis, Simons, Paraskakis et al. (2013)]. Intermediation solutions further encompass three types after [Gonidis (2015)]:

- **Library based** (JClouds, LibClouds, Pkgclouds etc.) These offer an intermediate API to the developers which is provider agnostic and thus abstracts the

heterogeneous providers.

- **Middleware solutions.** These solutions obscure the differences in underlying computer architectures and operating systems to abstract the disparities of inherent resources on which the application is running. These also deal with the application's execution and communication with its components which could have been hosted on heterogeneous environments.
- **Model-Driven Engineering Based solutions.** These follow the technique of "model once, generate anywhere" [Rith, Lehmayr and Meyer-Wegener (2014)] and are based on the core concepts of abstraction and automation. Here, the application is described at a higher abstraction level which is different than what is exposed by the cloud providers [Munisso and Chis (2017)]. Then automation allows changing the level of abstraction automatically with the help of model transformation.

The researches done in Alomari et al. [Alomari, Barnawi and Sakr (2015); Shirazi, Kuan and Dolatabadi (2012); Bastião Silva, Costa and Oliveira (2013); Hill and Humphrey (2010); Beslic, Bendraou, Sopena et al. (2013); da Silva, Lucrédio, Moreira et al. (2015); Strauch, Andrikopoulos, Bachmann et al. (2013); Sellami, Bhiri and Defude (2016); Bansel, Gonzalez-Velez and Chis (2016)] deal with data storage in clouds. However, [Shirazi, Kuan and Dolatabadi (2012)] did not handle the data portability challenge at the application level as our proposed middleware handles. The work in [Bastião Silva, Costa and Oliveira (2013)] catered the columnar data only whereas our middleware targets relational, document-oriented, and Columnar DBs. CSAL (Cloud Storage Abstraction Layer) in Hill et al. [Hill and Humphrey (2010)] preserved metadata concerning each container level entity viz. blob containers, tables, and queues which results in the overhead for dealing with metadata intensive operations. da Silva et al. [da Silva, Lucrédio, Moreira et al. (2015)] targeted the Google App Engine and Microsoft Azure cloud storage but they have not mentioned that their technique supports leveraging heterogeneous DBs platform services of different clouds as our middleware handles. The work in Bansel et al. [Bansel, Gonzalez-Velez and Chis (2016)] targeted just NoSQL DBs for data migration whereas our focus is data portability as well as data migration among relational and NoSQL DBs of the supported clouds. In both works [Beslic, Bendraou, Sopena et al. (2013); Strauch, Andrikopoulos, Bachmann et al. (2013)], the migration of relational data to and from NoSQL data is not targeted as our middleware targets. ODBAPI (OPEN-PaaS-DataBase API) proposed in Sellami et al. [Sellami, Bhiri and Defude (2016)] provided REST API for the migration which would incur more latency than our middleware. But in our proposed middleware, the whole database-related services are bundled inside the user's application and hosted along with the user's application. Our proposed middleware is inspired by the CDPort framework in Alomari et al. [Alomari, Barnawi and Sakr (2015)] but we tried to improve over it in terms of supported clouds, supported data storage services, and technical flaws as CDPort framework is prone to SQL injection. After thoroughly examining the source code [CDPort github (2014)] of CDPort, we found that the INSERT query in this library is prone to SQL Injection since it is not using parameterized queries. The query is being generated by merely concatenating the values in the query text which not recommended in real-world applications. Other queries (SELECT, UPDATE, and DELETE) implemented in this file are also not parameterized and are prone to SQL

Injection. This library relies on the user to sanitize their data before sending to the query, rather than handling it in the library by sanitizing the values in the library's SQL Service and using parameterized queries.

The model-based approaches such as in Jia et al. [Jia, Zhao, Wang et al. (2016); Atzeni, Bugiotti and Rossi (2012); Beslic, Bendraou, Sopena et al. (2013); da Silva, Lucrédio, Moreira et al. (2015); Scavuzzo, Tamburri and Di Nitto (2016); Pulgatti (2017); Bansel, Gonzalez-Velez and Chis (2016)] generate some models which further generate code based on the models. These models and codes are used since the very beginning while developing an application (which consumes platform basic services) [Munisso and Chis (2017)]. This renders the applications based on these models to be portable among heterogeneous cloud platform providers. On the other hand, in our approach, the user provides the entity models. In the case of already developed applications, if the MVC (Model View Controller) approach had been followed for data persistence, the application can use the proposed middleware. But if they did not adapt their source code according to the MVC approach, they need to adapt their source code according to the MVC approach to use this middleware.

The abstraction based solutions such as in Roijackers et al. [Roijackers and Fletcher (2013); Rith, Lehmayr and Meyer-Wegener (2014)] mostly implement only that functionality that is available in every supported cloud platform whereas our solution also supports the services which are not available in every supported cloud. User has the option of using the unavailable service from a different supported cloud platform since the middleware is interoperable among the clouds.

4 Proposed cloud data portability middleware

Presently, no standards are available for NoSQL query language. When implementing the polyglot persistence involving SQL and NoSQL DBs, separating the data over secerned DBs involve manual efforts to handle multiple data sources. Here, the problem arises when the developer needs to access these different DB systems. The middleware, if used while developing the application, provides an abstraction layer over these disparate kinds of DBs to mitigate the implementation intricacies of each supported DB.

The following challenges are listed by Alomari et al. [Alomari, Barnawi and Sakr (2015); Gonidis, Simons, Paraskakis et al. (2013)], while porting an application's data among various PaaS platforms:

- Different data models: Not only porting data from relational SQL to NoSQL data stores poses issues, but even portability among different types of NoSQLs is also not trivial (e.g., porting data from column-oriented store to document-oriented). As our solution middleware uses POCO (Plain old CLR objects), all the operations of the application are done on objects only. To persist the objects, the objects are passed to the abstraction layer which determines the type of the object and the base class from which it is inherited. If it is inherited from the SQL class, the query is converted into SQL. Otherwise, if it is inherited from the NoSQL class, the query is converted into a NoSQL query. For different relational data stores, the syntax of queries also varies. For example, in SQL Server, the query is

SELECT * FROM table WHERE column1=@parameter1

And the same query in MySQL is

SELECT * FROM table WHERE column1=: parameter1

- Various data access and query interfaces: All the databases specify their APIs or data access mechanisms which make the data access an issue for even in the scenario in which data is ported among the same category of the data store (e.g., MongoDB to CouchDB even though both are document-oriented DBs or SQL server to MySQL as both come under relational category). So, if the user needs to switch the DB from SQL Server and MySQL, the framework requires minimal changes in the configuration file and no changes in the source code. If there is a requirement to change the data store from SQL Server to MongoDB, the user needs to change the data model associated with SQL to MongoDB in the source code and the associated connection string in the configuration file. These are the minimal changes that are required for SQL to NoSQL migration. Also, if the user needs to migrate the existing data from SQL Server to MongoDB, he/she can use the migration tool [Migration Tool (2020)].
- Incompatible data typing mechanisms: Each data store may support different data types, for instance, one data store stores data as strings only while another supports other data types as well. This poses a complex challenge which currently is out of the scope of the proposed middleware. Nevertheless, the data stores supported by the proposed middleware do not pose this issue.

Fig. 1 shows the architecture of the proposed middleware with the supported clouds and databases.

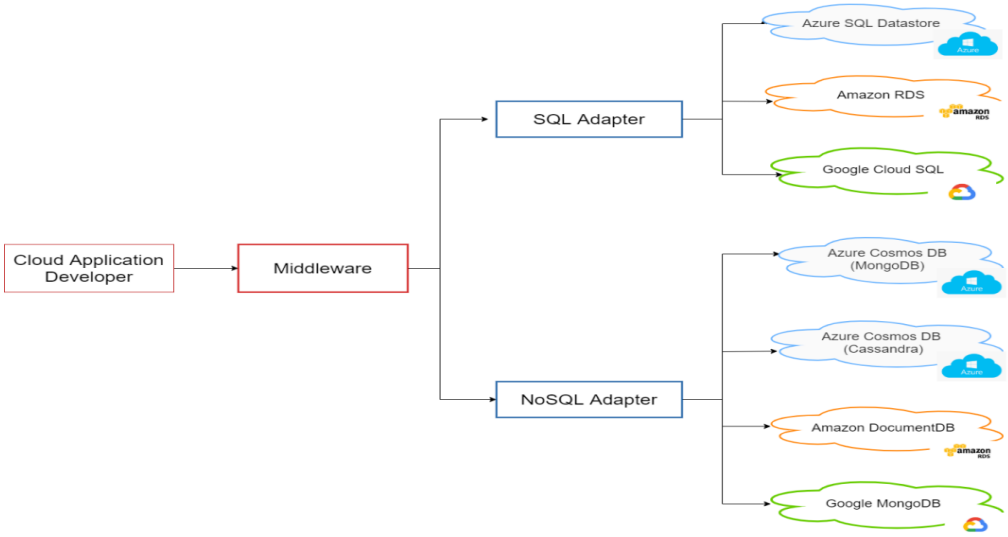


Figure 1: The architecture of the proposed middleware

4.1 Scenarios for the implementation

Some of the possible scenarios considered (for simplicity, we show scenarios involving

only two clouds, although we implemented total three clouds) in this paper for switching the data store as well as migrating the data of the application are as follows:

i) Cross-Cloud Homogeneous relational DB Migration (relational data store of one cloud to a similar relational data store of another cloud). The data model conversion in this scenario is easy since both are SQL databases. e.g., Azure SQL database to Amazon RDS (SQL Server) and vice versa.

ii) Cross-Cloud Homogeneous NoSQL DB migration (NoSQL data store of one cloud to same category NoSQL data store of another cloud). The data model conversion is complex in this scenario because even though for instance, Cosmos DB and Document DB support Mongo API, they have different data models. e.g., Azure Cosmos DB to Amazon DocumentDB and vice versa.

iii) Same Cloud Heterogeneous DB migration (Relational to/from NoSQL data store within the same cloud). The complexity of conversion is high because data is being ported from RDBMS to NoSQL document-oriented data storage service. Data type casting has to be taken care of. e.g., Azure Cosmos DB to Azure SQL database and vice versa.

iv) Cross-Cloud Heterogeneous DB migration (Relational to/from NoSQL data store within the different clouds). The complexity of conversion is high because data is being ported from RDBMS to NoSQL document-oriented data storage service. Data type casting has to be taken care of. Also, the cloud service compatibility is to be taken into account. e.g., Azure SQL Database to Amazon DocumentDB and vice versa.

v) Cross-Cloud Heterogeneous NoSQL DB migration (One category of NoSQL data store to/from another category of NoSQL data store of another cloud). The complexity of conversion is high because data is to be ported from one NoSQL (Document oriented) to another NoSQL (Columnar) data storage service. Data type casting has to be taken care of. e.g., Amazon DocumentDB to Azure Cosmos DB (Cassandra) and vice versa.

Data migration needs to be done in all cases if the user has a requirement to have access to the previous data residing in the source DB. Nonetheless, the above scenarios are merely examples; the proposed middleware supports the combinations of data stores services and clouds given in Tab. 1.

Table 1: Supported clouds and data stores

Supported Clouds→ **Supported Data stores**	**Azure**	**Amazon Web Services**	**Google Cloud Platform**
SQL	Azure SQL Database	Amazon RDS (SQL SERVER)	Google Cloud SQL
MongoDB	Azure Cosmos DB (With MongoDB API)	Amazon DocumentDB	Google MongoDB
Cassandra	Azure Cosmos DB (With Cassandra API)	-	-

4.2 Architecture of the proposed middleware

4.2.1 Switching the data stores

All the entities of the user data models are kept in the form of objects, so the objects' type decides the data stores where they persist. A user-defined model is a class that is inherited from a specific parent class (i.e., middleware meta model class). So, instead of the unified data model as in [Alomari, Barnawi and Sakr (2015)], we provide a different middleware metamodel class for each of the data stores supported. e.g., TSqlModel for SQL type data store, and TMongoModel for Mongo data store.

```
public class TSqlModel
  {
    [Key]
    public Guid Id { get; set; }
  }
```

Each of these middleware meta model classes has a key/property called ID which acts as a primary key. The value of the Primary key for the database is generated by the middleware only, although the user can also define a unique key. After defining the properties of the user-defined model/class, the user-defined model is inherited from the respective middleware meta model class and is passed to the abstraction layer provided by the proposed middleware.

```
public class UserDefinedModel: TSqlModel
  {
    public string Prop1 { get; set; }
    public string Prop2 { get; set; }
  }
```

DatabaseContext class (Fig. 2) determines the data store where the object is persisted, by detecting the middleware meta model class from which the passed model object is inherited. The only change required for switching among the data stores is to replace the middleware meta model class with the new data store model class. For instance, to change the data store from SQL to MongoDB, 'TSqlModel' in the above code snippet needs to be replaced with 'TMongoModel' as in the following code snippet:

```
public class UserDefinedModel: TMongoModel
  {
    public string Prop1 { get; set; }
    public string Prop2 { get; set; }
  }
```

Then the query is generated according to the user-defined model class that is passed to the data store specific context class. This is done with the help of Reflection (a concept of Object-Oriented programming) which extracts the properties and their values from the passed object in the data store specific context class as follows:

```
if (type.IsClass)
```

```
 {
  string query = $"INSERT INTO {type.Name}";
  query += " (";
  foreach (var propertyInfo in type.GetProperties())
  {
        query += propertyInfo.Name + ",";
  }
  query = query.TrimEnd(',');
  query += ") ";
  query += "VALUES";
  query += " (";
  foreach (var propertyInfo in type.GetProperties())
  {
       query += $"@{propertyInfo.Name},";
  }
  query = query.TrimEnd(',');
  query += ")";
  RunCommand(query, obj);
}
```

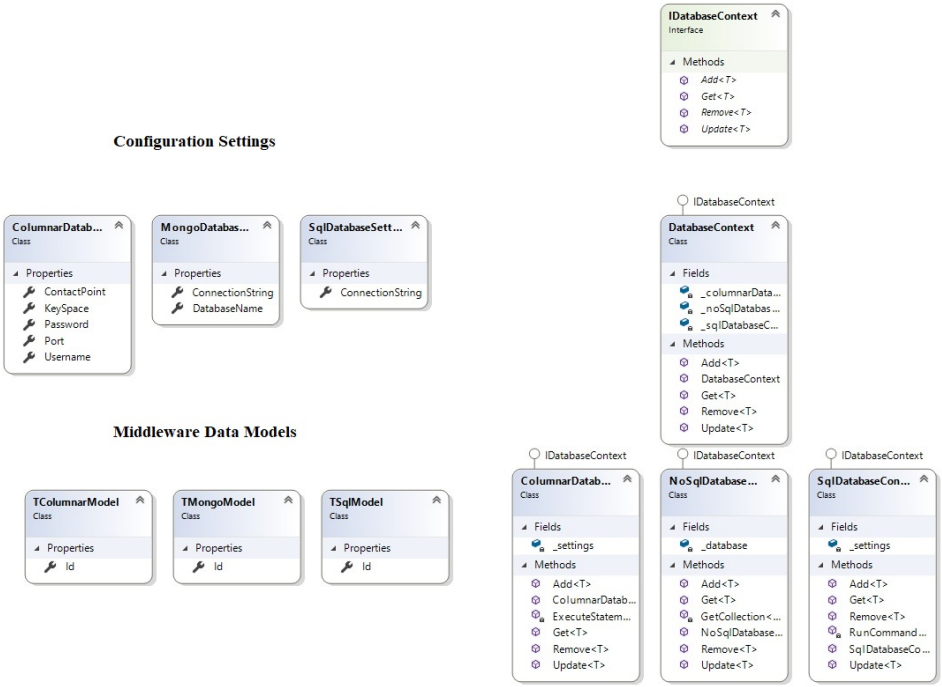


Figure 2: Class Diagram of the proposed middleware

4.2.2 Migrating the data

Lewis [Lewis (2012)] defined data migration as "Data that resides in one cloud provider can be moved to another cloud provider." Data migration is a tedious task that involves

i) Extracting data from its original DB

ii) Transforming it to a format which is compatible with the target DB

iii) Uploading the formatted data into the target DB

For migration, we developed a graphical user interface tool (Fig. 3) using the middleware. This tool allows the user to choose the source as well as destination DB types. The user has to provide the source and destination DB connection string along with the database name.

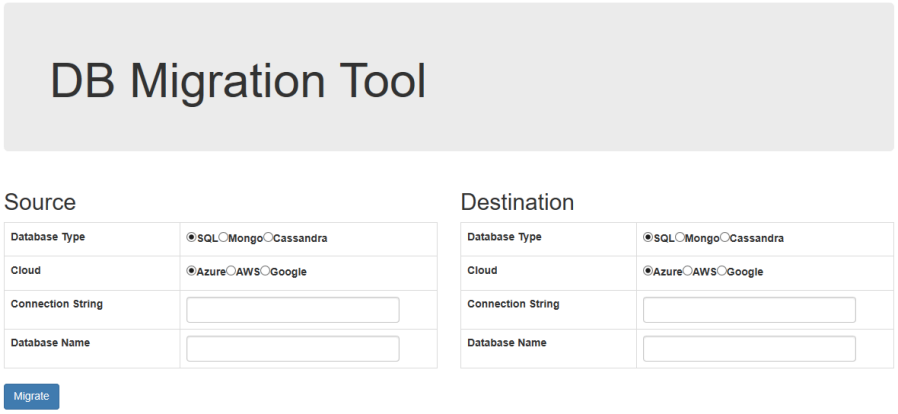


Figure 3: Graphical User Interface for the proposed Data Migration Tool

The data transformation process of the proposed middleware and the algorithm to perform the data migration is as follows:

```
// get source and destination data store credentials
source_db=connect(source_database_conn_string);
// connect to the source data store
destination_db=connect(destination_database_conn_string);
// retrieve the entities to migrate
entities=get_all_entities(source_db);
// convert the entities according to middleware data models
foreach(entity in entities)
{
tuples=get_tuples(entity);
// connect to destination data store
converted_tuples=convert_to_destination_tuples(tuples);
foreach(converted_tuple in converted_tuples)
{
```

```
// store the entities in destination data store
save_in_destination_db(converted_tuple);
}
}
```

For this period, the application might be down for the migration. After the migration, all the following operations will start being performed in the new data store.

4.2.3 Data models

As discussed above, the proposed middleware facilitates the portability of data (which means changing the data store and migrating the data as well) between different data stores (i.e., SQL and NoSQL) across different cloud data storage services. The middleware accomplishes this by converting the data into POCO (Plain Old CLR Objects) and then converting them into their SQL and NoSQL counterparts and vice versa. It supports various data stores across different cloud platforms listed below:

- **Amazon RDS:** Amazon RDS is the relational database service provided by Amazon. Marketed as scalable and easy to set up, Amazon RDS service provides support for PostgreSQL, MySQL, MariaDB, Oracle Database and Microsoft SQL Server. Our proposed middleware supports MySQL and Microsoft SQL Server. While Amazon RDS supports SQL Server, it does not fully support all the features of SQL Server, for e.g., several server-level roles and server-level permissions are not supported by Microsoft SQL Server on Amazon [Amazon RDS (2020)].
- **Azure SQL Database:** Azure SQL Database is the scalable cloud database service by Microsoft Azure that is based on SQL Server. It has machine learning-based monitoring and tuning of SQL Server instance. Azure SQL Database provides high-level security via various layers of network security, access management, threat protection, and information protection [Azure SQL Database (2020)].
- **Azure Cosmos DB:** Azure Cosmos DB is a fully managed multi-model database service that supports elasticity and unlimited scalability. It supports Cassandra, MongoDB, and SQL [Azure Cosmos DB (2020)]. Although Cassandra and MongoDB are provided as DB services, the API is provided in such a way that it works as a cloud-native database. Our proposed middleware supports both Cassandra and MongoDB API of Azure Cosmos DB.
- **Amazon DocumentDB:** Amazon DocumentDB is a fully-managed database service that supports MongoDB workloads. In Amazon DocumentDB, the storage and compute are decoupled. They can be scaled independently [Amazon DocumentDB (2020)].
- **Google Cloud SQL:** Google Cloud SQL is a SQL database service provided by the Google Cloud Platform. It provides the MySQL, PostgresSQL, and SQL Server database engines. It has built-in automation for high availability, backups, and security updates [Google Cloud SQL (2020)].
- **Google MongoDB:** MongoDB is provided on Google Cloud Platform as MongoDB as a Service. It is a partner solution that is a result of collaboration between the MongoDB and Google. It is MongoDB Atlas which is hosted on MongoDB cloud

servers and managed through Google Cloud Platform [Google MongoDB (2020)].

- **Proposed Data Model:** The proposed data model is different than that given by [Alomari, Barnawi and Sakr (2015)]. Whenever a new data store is added, (the type of the database being used viz. SQL or NoSQL), a new meta-model class for the respective data store is created. Currently, we have three meta model classes: TSqlModel, TColumnarModel, and TMongoModel in the middleware. For every data store supported by the middleware, there will be a meta-model class. These classes define, that when a user creates a user-defined entity, he/she has to inherit that entity from these metamodel classes. This will make the proposed middleware's abstraction layer to detect the data store entity to which this user-defined entity belongs. When an object is passed to the abstraction layer, it will look for the meta-model class it belongs to, and from that it can determine the data store where this object needs to be stored.

Tab. 2 presents a sample of data type conversions that are carried out automatically by the proposed middleware.

Table 2: Sample of data types conversion

Middleware's Data Types	Azure SQL Database	Azure Cosmos DB (MongoDB)	Azure Cosmos DB (Cassandra)	Amazon RDS	Amazon DocumentDB	Google SQL	Google MongoDB
Guid	UniqueIdentifier	ObjectId	UUID	UniqueIdentifier	ObjectId	UniqueIdentifier	ObjectId
Int	Int	Integer	Int	Float	Integer	Int	Int
Double	Float	Long	Float	Float	Float	Float	Float
DateTime	Datetime	Date	Datetime	timestamp	Datetime	Datetime	Date
String	Varchar	String	Text	Varchar	String	Varchar	String

4.2.4 Middleware implementation

The middleware API hides the complex programming of the implementation of each data store. It provides a single programming interface that automatically connects to all the data stores configured in the configuration file of the application. The middleware makes use of the built-in dependency injection feature of the .NET core framework to instantiate the credentials and services of all the data stores being used in the application. The middleware detects the data stores being used by going through the configuration file and instantiates only those services for which the credentials are provided. Then the user can instantiate the main API with the following code:

```
class SomeModel: TSqlModel
{
  public string SomeProperty {get; set; }
```

```
}
class SomeController
{
  private DatabaseContext _dbContext;
  public SomeController(DatabaseContext dbContext)
  {
    _dbContext = dbContext;
  }
  public IActionResult SomeMethod()
  {
   SomeModel obj = new SomeModel() { SomeProperty = "Some Valid Property"};
    _dbContext.Add(obj);
  }
}
```

Here, SomeModel is a user-defined data model and TSqlModel is the middleware meta-model. SomeModel is inherited by TSqlModel which signifies that it belongs to a SQL data store. DatabaseContext is the class that provides the abstraction to the user from the intricacies of the middleware. The Add() method of DatabaseContext class determines the data store by checking its base class (middleware meta-model) and then internally calls the SqlDatabaseContext which creates the INSERT query to save the object as a row in the SQL data store.

Fig. 2 in Section 4.2.1 presents the class diagram of the middleware, with the services and middleware data models. The middleware API offers the following CRUD and JOIN operations that are needed to manipulate the data persisted in the supported data stores:

(i) **Add:** This method takes an object of user-defined models and stores it in the relevant data store. If the object already exists, it updates the existing record.

(ii) **Update:** This method takes an object of user-defined models and updates it in the relevant data store. If the object does not exist, it creates a new record in the data store.

(iii) **Remove:** It removes the record from the data store of the passed object if the ID property of the object is found in the data store. If no record is found, nothing is deleted.

(iv) **Get:** Get method is used to get the ResultSet back to the user for the model's class type passed in the function.

(v) **Join:** Join is used to perform JOIN operations between the result sets of two different user-defined entities. It works similar to the JOIN operations in SQL. The join method of our middleware also works on NoSQL entities.

Normally, in case of inserting a record in SQL, the following query is executed:

```
INSERT INTO table (column1, column2) VALUES ('value1', 'value2')
```

And in case of inserting a record in MongoDB, following code is required,

```
var obj = new SomeClass() { someProperty="Some Property" };
```

```
var client = new MongoClient("connection_string");
var database = client.GetDatabase("database_name");
var collection = database.GetCollection("collection_name");
collection.InsertOne(obj);
```

However, the proposed middleware eases out this insertion statement execution by using

```
_dbContext.Add(obj);
```

Using the above statement, if the user-defined data model is inherited from TSqlModel, it will generate an INSERT query. If the user-defined data model is inherited from TMongoModel, it will generate a BSON object, retrieve a collection based on the name of the data model class and insert the object in the collection. The design patterns leveraged in the middleware are repository pattern, adapter pattern and dependency injection.

5 Experimentation and evaluation

For the evaluation, we conducted an experiment where we created a sample web application using .NET Core since the proposed middleware currently is coded in C# using .NET core framework. The application is hosted using the App Services Linux instance of SKU Free F1 (1 GB RAM, 60 minutes/day compute time). Azure SQL Database used is of Basic Tier consisting of 5 DTUs and a size of 2 GB max. Azure Cosmos DB is dynamically prices based on pay as you go method. Amazon RDS instance is of SQL Server Express engine of db.t2.micro size which has 1 vCPU and 1 GB RAM. Amazon DocumentDB used is of db.r5.large type which has 2 vCPUs and 16 GB RAM with up to 3500 Mbps bandwidth. Google SQL is an instance of the SQL Server Express engine which has 1 vCPU, 3.75 GB RAM, and 20 GB SSD storage. Google MongoDB is an instance hosted on MongoDB cloud which has shared vCPU and RAM with 512 MB storage.

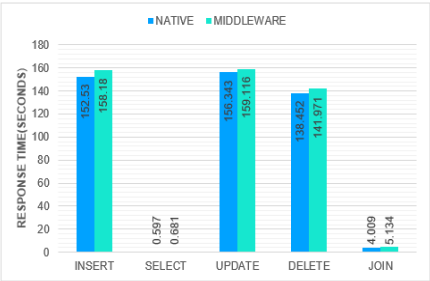
Microsoft Azure, Google, and Amazon Web Services are the currently supported cloud platform providers for the proposed middleware and Amazon RDS, Azure SQL Database, Azure Cosmos DB (Mongo and Cassandra), Amazon DocumentDB, Google Cloud SQL, and Google MongoDB are the supported data stores. For our experimentation, we have used the same Twitter dataset by [Singh, Sawhney and Kahlon (2017)]. Tweets of this Twitter data set were used for evaluating the benchmark results for the middleware. The schema has Guid, string, int, date, and float datatypes.

5.1 Evaluating the performance of the proposed middleware

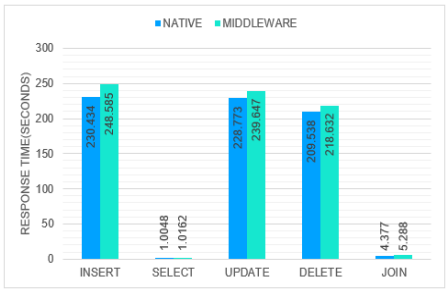
We evaluated the performance of the middleware for each data store on each cloud service by performing CRUD and JOIN operations using one thousand tweets. These one thousand tweets were selected randomly from one hundred thousand tweets that are archived in the form of an excel file. A toy web application was hosted on Azure App Service which is using Azure SQL Database. Fig. 4 shows the schema of the data set used to benchmark our middleware.

	Name	Data Type	Allow Nulls
🔑	Id	uniqueidentifier	☐
	Candidate	nvarchar(MAX)	☑
	tweet	nvarchar(MAX)	☑
	Area	nvarchar(MAX)	☑
	Date	nvarchar(MAX)	☑
	Send_via	nvarchar(MAX)	☑
	Sender	nvarchar(MAX)	☑
	Place	nvarchar(MAX)	☑
	Flag1	int	☑
	Flag2	int	☑

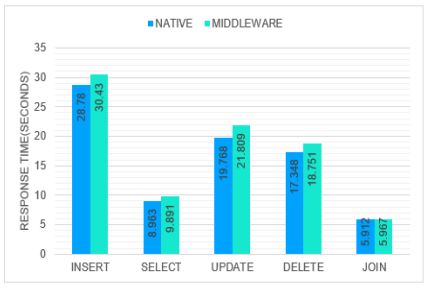
Figure 4: Schema of the data set



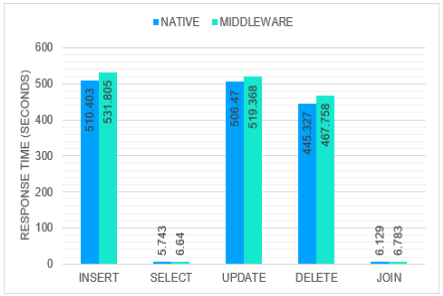
(a) Amazon Document DB



(b) Amazon RDS

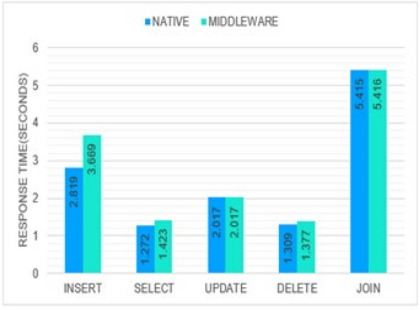


(c) Azure Cosmos DB (MongoDB)

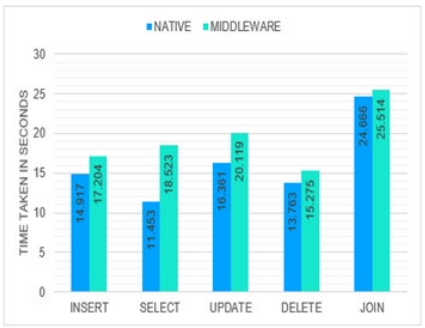


(d) Azure SQL Database

(e) Google Cloud SQL



(f) Google MongoDB



(g) Azure Cosmos DB (Cassandra)

Figure 5: Response Times for CRUD and JOIN operations in various supported clouds

We compared the time of executing the calls by the underlying data storage services using our middleware's abstracted API with the time of executing the same call using the .Net framework API provided by each of the services. In particular, we used the difference of Start time and End time of the System.DateTime objects to calculate the time taken by the call. Since, it is the native date-time API of the .Net framework; it does not add to any overhead and provides an accurate time. We repeated all execution use cases five times each and then calculated the mean value to avoid the effect of any faulty measurements that could be caused by any external factor such as network fluctuations. We measured the performance of each of the main operations: Create, Read, Update, and Delete. Also, we measured the performance of more complex calls involving JOIN operation. The abstraction layer of the proposed middleware hides all the variations between different data stores. The scenarios described in Section 4.1 were implemented during the experimentation for benchmarking the response times for performing CRUD and JOIN operations using the proposed middleware for different data stores against the same for native API. Fig. 5(a) illustrates the comparison between Amazon Document DB native API and our middleware API. It shows that the performance overhead is very negligible for all the operations. Similarly, the graphs in Figs. 5(b)-5(g) show the efficiency of the middleware against native API of Amazon RDS using SQL Server, Azure Cosmos DB (MongoDB), Azure SQL Database, Google Cloud SQL, Google MongoDB and Azure Cosmos DB (Cassandra) respectively. The graphs and readings support our claim that using the proposed middleware will not pose much latency differences in the user's application. Extensive testing was done to validate the efficiency and accuracy of the middleware.

5.2 Evaluating the performance of the data migration tool

For evaluating the effectiveness of the developed data migration tool, we ported the data among the supported cloud-based data storage services. We tried all the scenarios mentioned in Section 4.1 for moving the data. For migrating the data manually, the user is required to be acquainted with the data model of the source and destination data storage service. The user can leverage the available import or export tools provided by the data storage service. But for porting the data across different data storage services

having different data models, it is tedious to find the existing tools that support different scenarios. Therefore we developed the transformation code that could perform the data migration by retrieving the data from source data storage service, convert retrieved data to the target service's data model and eventually copy the converted data to the target data storage service. On the other hand, to carry out data migration using our migration tool, the user just needs to enter the connection information (type of data storage type, connection string, instance name etc.) of source and target data storage service. The data transformation is performed automatically by the developed migration tool.

Fig. 6 shows the exemplar identical results retrieved for performing the data migration scenarios mentioned in Section 4.1.

Id	Candidate	tweet	Area	Date	Send_via	Sender	Place
78f9746e-...	Donald Trump	@Mojoman4Real...	USA	9/1/2016	Iphone	WhereEagles...	Not Available
6ca69f83-...	Donald Trump	VIDEO : #Latinos...	USA	9/1/2016	Twitter Website	Bennie E	Not Available
64eaca8c-...	Donald Trump	@CookingConqu...	USA	9/1/2016	Twitter Website	Lifecoach	Not Available
8d752dba...	Donald Trump	#CrookedHillary'...	USA	9/1/2016	Android Phones	Karen P. Cox	Not Available
5ba7e689...	Donald Trump	I just needed to r...	USA	9/1/2016	Twitter Website	Edward Goldb...	Not Available
cc349daa-...	Donald Trump	Joe Biden said #...	USA	9/1/2016	Twitter Website	Stacie Walker	Not Available
d852b268...	Donald Trump	Thanks @tponew...	USA	9/1/2016	Iphone	?Women4Tru...	Not Available
2ccf5283-...	Donald Trump	Trump Website C...	USA	9/1/2016	Twitter Website	Sister Patriot	Not Available
a6c16f61-...	Donald Trump	Finally a Backlash...	USA	9/1/2016	Twitter Website	William J.	Not Available
cd4ad23e...	Donald Trump	#1A #2A #MAGA...	USA	9/1/2016	Iphone	World Zig Zag	Not Available
15585f99-...	Donald Trump	Exclusive: #Trum...	USA	9/1/2016	Iphone	Leon Report	Not Available

(a) Azure Cosmos DB (Cassandra API)

Id	Candidate	tweet	Area	Date	Send_via	Sender	Place
cbe003b9-c722...	Donald Trump	#Trump what does that unconsti...	USA	9/1/2016	Iphone	Sid Johnson	Not Available
b6b4e12f-110f-...	Donald Trump	#Coulter: #Hillary 'looks so hagg...	USA	9/1/2016	Twitter Website	Rodney L. Bruce	Not Available
24a03a4e-5934-...	Donald Trump	NOVEMBER 8th is the day!TRUM...	USA	9/1/2016	Iphone	TRUMP 2016	Not Available
0e291280-ec90-...	Donald Trump	#Trump's #Immigration Speech ...	USA	9/1/2016	Other Sources	Doc Hamrick	Not Available
de2fe826-2c67-...	Donald Trump	Trump's immigration pitch falls f...	USA	9/1/2016	Ipad	Speak out!	Not Available
f0a7ea35-2fac-...	Donald Trump	President of Mexico tweets they ...	USA	9/1/2016	Iphone	Stefanie, M.S.	Not Available
11ff3f7e-1acc-4...	Donald Trump	#VoteTrump, only an absolute cl...	USA	9/1/2016	Iphone	Tim Alexander	Not Available
9b3bb720-90bc...	Donald Trump	Gotta love #NeverTrumpers... Ev...	USA	9/1/2016	Twitter Website	Robert Suppen...	Not Available
ed664618-905b...	Donald Trump	You're as ridiculous as your cand...	USA	9/1/2016	Twitter Website	Cherry	Not Available
b023f165-1b1e-...	Donald Trump	I assume that some of my Hispa...	USA	9/1/2016	Ipad	Donald J. Trump	Not Available
6a0cd85b-5c71...	Donald Trump	.@Follow_Trump The Republican...	USA	9/1/2016	Iphone	Rods and Guitars	Not Available
d919810f-265a-...	Donald Trump	The New York Post published ph...	USA	9/1/2016	Other Sources	DailyNewsSpot	Not Available
74e15501-dae0-...	Donald Trump	#DonaldTrump Toilet Paper, is m...	USA	9/1/2016	Iphone	Twitting like nu...	Not Available
8bd767cb-4402...	Donald Trump	Lt. Gen. Michael Flynn, the kkk ...	USA	9/1/2016	Iphone	Sid Johnson	Not Available
2bd3b7e2-92b5...	Donald Trump	@tony_tonyt @naniiiiitas @Early...	USA	9/1/2016	Twitter Website	25° North	Not Available
7921f6a6-1484-...	Donald Trump	.@AriMelber to Jacob Monty: "W...	USA	9/1/2016	Ipad	Ginger	Not Available
de927b01-c7cb...	Donald Trump	Gotta love #NeverTrumpers.. Eve...	USA	9/1/2016	Twitter Website	Robert Suppen...	Not Available
0b726d85-6202...	Donald Trump	@FoxNews @TheJuanWilliams t...	USA	9/1/2016	Iphone	AssAndElephant	Not Available
c85890ff-82b2-...	Donald Trump	#Clinton #campaign slams #Do...	USA	9/1/2016	Other Sources	Renita Saint	Not Available
7ae7ed12-f1b7-...	Donald Trump	Donald Trump's fill in the blank (...	USA	9/1/2016	Android Phones	Matthew Mulder	Not Available

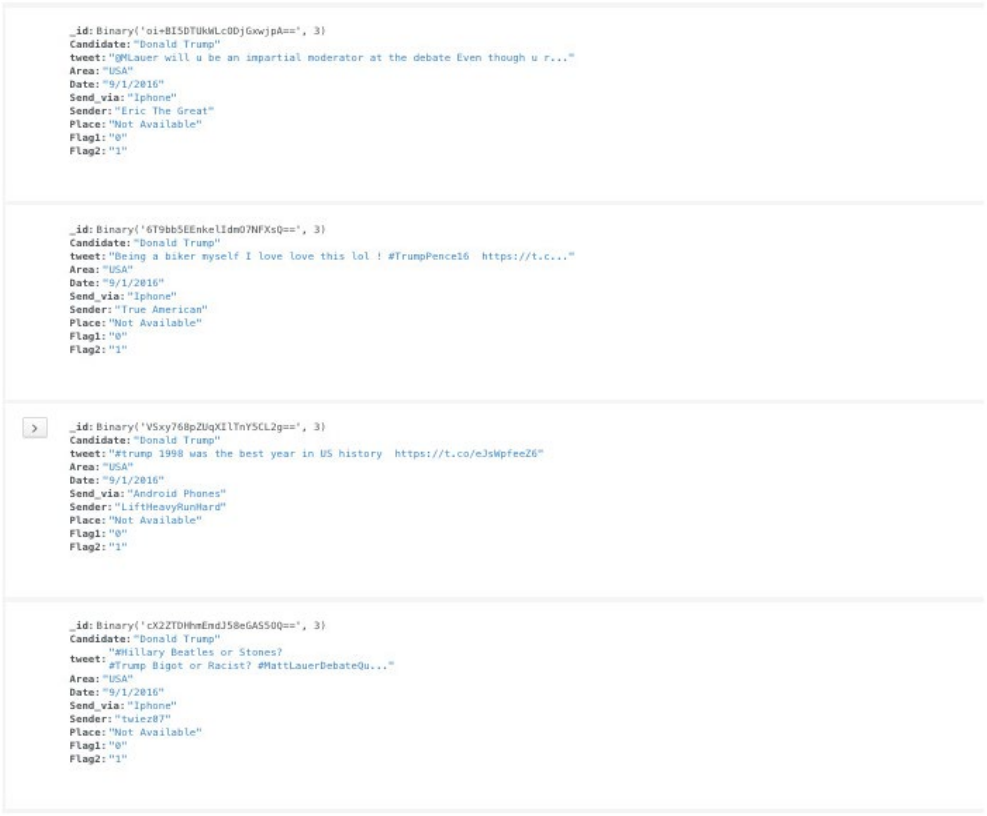
(b) Amazon RDS

Id	Candidate	tweet	Area	Date	Send_via	Sender	Place
12164838...	Donald Trump	Hate politics but neither #...	USA	9/1/2016	Twitter Website	Customers On...	Not Available
0b1f419e...	Donald Trump	So #DonaldTrump - when...	USA	9/1/2016	Twitter Website	BAM	Not Available
91e5840b...	Donald Trump	Everyone in the #US kno...	USA	9/1/2016	Twitter Website	Dar Weathers	Not Available
713032bd...	Donald Trump	@henri_picker So you are...	USA	9/1/2016	Twitter Website	John	Not Available
a84146c5...	Donald Trump	Liberal HACK Matt Lauer "...	USA	9/1/2016	Twitter Website	Bennie E	Not Available
5ee40ac5...	Donald Trump	@TwitterSurveys stop the...	USA	9/1/2016	Iphone	Johanesburger...	Not Available
59fbf573-...	Donald Trump	Lying media: Like #Crook...	USA	9/1/2016	Ipad	Richard	Not Available
9e447a00...	Donald Trump	#NYTimes #ForeignPolicy...	USA	9/1/2016	Twitter Website	Foreign Confi...	Not Available
733a9f40-...	Donald Trump	@JoeBiden is a genuine t...	USA	9/1/2016	Twitter Website	NancyLineJaco...	Not Available
4ca24585...	Donald Trump	Well, out 4 a few hrs on R...	USA	9/1/2016	Android Phones	Tammy	Not Available
05a225a8...	Donald Trump	Love or loathe it, #Donald...	USA	9/1/2016	Other Sources	Chet's Ascertai...	Not Available

(c) Azure Cosmos DB (MongoDB API)

Id	Candidate	tweet	Area	Date	Send_via	Sender	Place
78f9746e-f...	Donald Trump	@Mojoman4Real @Rebel_Bill ...	USA	9/1/2016	Iphone	WhereEaglesDare	Not Available
6ca69f83-d...	Donald Trump	VIDEO : #LatinosForTrump Co-...	USA	9/1/2016	Twitter Website	Bennie E	Not Available
8d752dba-...	Donald Trump	#CrookedHillary's "Coronation"...	USA	9/1/2016	Android Phones	Karen P. Cox	Not Available
5ba7e689-9...	Donald Trump	I just needed to re-read a piece...	USA	9/1/2016	Twitter Website	Edward Goldberg	Not Available
cc349daa-d...	Donald Trump	Joe Biden said #Trump's rhetor...	USA	9/1/2016	Twitter Website	Stacie Walker	Not Available
d852b268-...	Donald Trump	Thanks @tponews it's great to ...	USA	9/1/2016	Iphone	?Women4Trump?...	Not Available
2ccf5283-1...	Donald Trump	Trump Website Crashes Due to...	USA	9/1/2016	Twitter Website	Sister Patriot	Not Available
a6c16f61-f0...	Donald Trump	Finally a Backlash? CBS Derides...	USA	9/1/2016	Twitter Website	William J.	Not Available
cd4ad23e-f...	Donald Trump	#1A #2A #MAGA #Hillary #Hill...	USA	9/1/2016	Iphone	World Zig Zag	Not Available
15585f99-7...	Donald Trump	Exclusive: #Trump Camp Mulls ...	USA	9/1/2016	Iphone	Leon Report	Not Available
edd2064e-...	Donald Trump	#Trump's first foreign visit as a ...	USA	9/1/2016	Android Phones	Soy Juan Miller	Not Available

(d) Azure SQL Database



(e) Google MongoDB

Id	Candidate	tweet	Area	Date	Send_via	Sender	Place
cbe003b9-c722...	Donald Trump	#Trump what d...	USA	9/1/2016	Iphone	Sid Johnson	Not Available
b6b4e12f-110f-...	Donald Trump	#Coulter: #Hilla...	USA	9/1/2016	Twitter Website	Rodney L. Bruce	Not Available
24a03a4e-5934-...	Donald Trump	NOVEMBER 8th...	USA	9/1/2016	Iphone	TRUMP 2016	Not Available
0e291280-ec90-...	Donald Trump	#Trump's #Im...	USA	9/1/2016	Other Sources	Doc Hamrick	Not Available
de2fe826-2c67-...	Donald Trump	Trump's immig...	USA	9/1/2016	Ipad	Speak out!	Not Available
f0a7ea35-2fac-...	Donald Trump	President of Me...	USA	9/1/2016	Iphone	Stefanie, M.S.	Not Available
11ff3f7e-1acc-4...	Donald Trump	#VoteTrump, o...	USA	9/1/2016	Iphone	Tim Alexander	Not Available
9b3bb720-90bc...	Donald Trump	Gotta love #Ne...	USA	9/1/2016	Twitter Website	Robert Suppen...	Not Available
ed664618-905b...	Donald Trump	You're as ridicul...	USA	9/1/2016	Twitter Website	Cherry	Not Available
b023f165-1b1e-...	Donald Trump	I assume that s...	USA	9/1/2016	Ipad	Donald J. Trump	Not Available
6a0cd85b-5c71...	Donald Trump	.@Follow_Trum...	USA	9/1/2016	Iphone	Rods and Guitars	Not Available
d919810f-265a-...	Donald Trump	The New York P...	USA	9/1/2016	Other Sources	DailyNewsSpot	Not Available
74e15501-dae0-...	Donald Trump	#DonaldTrump ...	USA	9/1/2016	Iphone	Twitting like nu...	Not Available
8bd767cb-4402...	Donald Trump	Lt. Gen. Michae...	USA	9/1/2016	Iphone	Sid Johnson	Not Available
2bd3b7e2-92b5...	Donald Trump	@tony_tonyt @...	USA	9/1/2016	Twitter Website	25° North	Not Available
7921f6a6-1484-...	Donald Trump	.@AriMelber to ...	USA	9/1/2016	Ipad	Ginger	Not Available
de927b01-c7cb...	Donald Trump	Gotta love #Ne...	USA	9/1/2016	Twitter Website	Robert Suppen...	Not Available
0b726d85-6202...	Donald Trump	@FoxNews @T...	USA	9/1/2016	Iphone	AssAndElephant	Not Available
c85890ff-82b2-...	Donald Trump	#Clinton #cam...	USA	9/1/2016	Other Sources	Renita Saint	Not Available
7ae7ed12-f1b7-...	Donald Trump	Donald Trump's...	USA	9/1/2016	Android Phones	Matthew Mulder	Not Available

(f) Google Cloud SQL

Figure 6: Ported data in the supported cloud data storage services

The graph in Fig. 7 shows the efficiency of the migration tool. In the evaluation scenarios, we used source databases consisting of one thousand rows (in case of SQL data store) and one thousand documents (in case of NoSQL data store). The same Twitter dataset was used for populating the source databases.

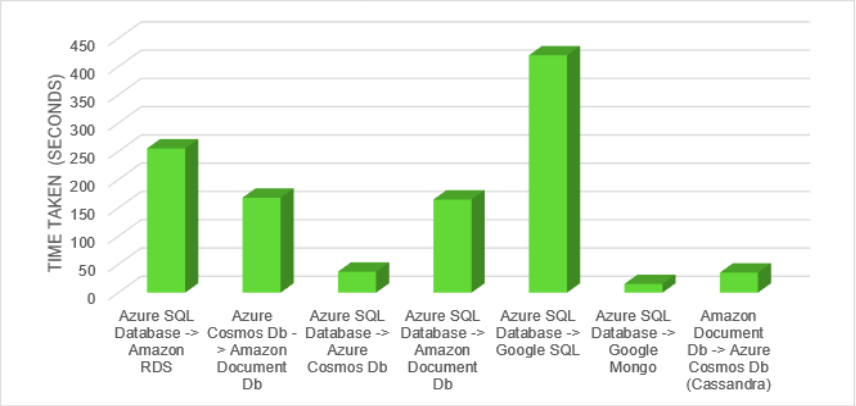


Figure 7: Time taken for migrating 1000 rows by different scenario

Our middleware's functionality may seem to be similar to the functionality of ORMs (Object Relational Mappers) and ONDMs (Object to NoSQL Data Mappers) as it also handles the data in the form of objects of user-defined data models. To persist them into relational DBs, we convert the objects into respective SQL queries which is similar to ORMs and ONDMs. To the best of our knowledge, there is no active ONDM framework available for C#.NET. But our middleware is polyglot and supports data interoperability (SQL to NoSQL and vice versa), cloud interoperability (data migration among different clouds) as well as data migration between different NoSQL categories. So, our solution is essentially better than ORMs and ONDMS due to the enhanced features stated above.

6 Conclusion and future work

The inherent differences in heterogeneous persistence models in addition to the different implementation APIs of these models render it time-consuming and requiring substantial efforts to port the data or changing the data stores among cloud providers. This paper highlights the need for polyglot persistence, research work done towards this issue and proposes a middleware solution to alleviate it. The experimental results to assess the latency overhead of using the proposed middleware against using just the native APIs are also presented. It provides an abstraction layer to hide the intricacies while changing the backend data store among the supported data stores. After evaluating the effectiveness, accuracy, and performance of the middleware, we conclude:

- It makes the development of portable applications far easier if used instead of the native APIs. Since, there is a minimal change required in code and most of the time it is required to change the configuration file only, changing the backend will be easier and feasible.
- It lessens the time and effort required for porting the data store backend of the

application to another cloud.

- The data migration tool developed using this middleware eases out the process of porting the data across different data models and different clouds viz. Microsoft Azure, Amazon Web Services and Google Cloud.

For the future, we will focus on adding more data stores, more clouds, and porting the proposed middleware in other languages like Java, PHP, etc. We will also be comparing analytically the performance of this middleware with the other frameworks like CloudMapper of [Munisso and Chis (2017)] and ODBAPI of [Sellami, Bhiri, and Defude (2016)] as well as other ONDMs (Object to NoSQL data mappers) available in the market.

Funding Statement: The author(s) received no specific funding for this study.

Conflicts of Interest: The authors declare that they have no conflicts of interest to report regarding the present study.

References

Alomari, E.; Barnawi, A.; Sakr, S. (2015): CDPort: A Portability framework for NoSQL datastores. *Arabian Journal for Science and Engineering*, vol. 40, no. 9, pp. 2531-2553.

Amazon DocumentDB (2020): https://cloud.google.com/sql/docs.

Amazon RDS (2020): https://docs.aws.amazon.com/AmazonRDS/latest/UserGuide/CHAP_SQLServer.html.

Atzeni, P.; Bugiotti, F.; Rossi, L. (2012), June: Uniform access to non-relational database systems: The SOS platform. In *International Conference on Advanced Information Systems Engineering*, pp. 160-174. Springer, Berlin, Heidelberg.

Azure SQL Database (2020): https://docs.microsoft.com/en-us/azure/sql-database/sql-database-technical-overview.

Bansel, A.; Gonzalez-Velez, H.; Chis, A. E. (2016): Cloud-Based NoSQL data migration. *Proceedings-24th Euromicro International Conference on Parallel, Distributed, and Network-Based Proceeding*, pp. 224-231.

Bastiao Silva, L. A.; Costa, C.; Oliveira, J. L. (2013): A common API for delivering services over multi-vendor cloud resources. *Journal of Systems and Software*, vol. 86, no. 9, pp. 2309-2317.

Beslic, A.; Bendraou, R.; Sopena, J.; Rigolet, J. Y. (2013): Towards a solution avoiding vendor lock-in to enable migration between cloud platforms. *CEUR Workshop Proceedings*, vol. 1118, pp. 5-14.

CDPort github (2014): https://github.com/CDPort/API/blob/master/API-Cloud-Database/src/main/java/api/AmazonRDS.java.

Chetal, A.; Peterson, J.; Wallace, J.; Drgon, M. (2011): Interoperability and portability. *Cloud Security Alliance.*

da Silva, E. A. N.; Lucrédio, D.; Moreira, A.; Fortes, R. (2015): Supporting multiple persistence models for PaaS applications using MDE: Issues on cloud portability. *CLOSER-Proceedings of 5th International Conference on Cloud Computing and Services Science*, pp. 331-342.

Fowler, M.; Sadalage, P. (2012): NoSQL database and Polyglot persistence. *Personal Website:* https://martinfowler.com/articles/nosql-intro-original.pdf.

Gartner (2019): https://www.gartner.com/en/newsroom/press-releases/2019-04-29-gartner-identifies-key-trends-in-paas-and-platform-ar.

Gonidis, F. (2015): *A Framework Enabling the Cross-Platform Development of Service-Based Cloud Applications (Ph.D. Thesis)*. University of Sheffield, South East European Research Centre.

Gonidis, F.; Paraskakis, I.; Kourtesis, D. (2012) September: Addressing the challenge of application portability in cloud platforms. *7th South-East European Doctoral Student Conference*, pp. 565-576.

Gonidis, F.; Simons, A. J.; Paraskakis, I.; Kourtesis, D. (2013) September: Cloud application portability: an initial view. *Proceedings of the 6th Balkan Conference in Informatics,* pp. 275-282.

Google Cloud SQL (2020): https://www.mongodb.com/cloud/atlas/mongodb-google-cloud.

Hill, Z.; Humphrey, M. (2010): November: CSAL: A cloud storage abstraction layer to enable portable cloud applications. *IEEE Second International Conference on Cloud Computing Technology and Science*, pp. 504-511.

InsideBigData (2019): https://insidebigdata.com/2019/12/28/big-data-cloud-computing-the-roles-relationships/.

Jia, T.; Zhao, X.; Wang, Z.; Gong, D.; Ding, G. (2016) June: Model transformation and data migration from relational database to MongoDB. *IEEE International Congress on Big Data*, pp. 60-67.

Kaur, K.; Sharma, D. S.; Kahlon, D. K. S. (2017): Interoperability and portability approaches in inter-connected clouds: a review. *ACM Computing Surveys*, vol. 50, no. 4, pp. 1-40.

Korte, F.; Challita, S.; Zalila, F.; Merle, P.; Grabowski, J. (2018) May: model-driven configuration management of cloud applications with OCCI. *CLOSER-Proceedings of the 8th International Conference on Cloud Computing and Services Science-Janua*, pp. 100-111.

Kostoska, M.; Gusev, M.; Ristov, S. (2015) September: an overview of cloud portability. *Future Access Enablers of Ubiquitous and Intelligent Infrastructures*, pp. 248-254. Springer, Cham.

Lewis, G. A. (2013) January: Role of standards in cloud-computing interoperability. *46th Hawaii International Conference on System Sciences*, pp. 1652-1661.

Migration Tool (2020): https://polyglot-db-migration.azurewebsites.net/.

Munisso, R.; Chis, A. E. (2017) March: cloudmapper: a model-based framework for portability of cloud applications consuming PaaS services. *25th Euromicro International Conference on Parallel, Distributed and Network-Based Processing*, pp. 132-139.

Petcu, D. (2011) October: Portability and interoperability between clouds: challenges and case study. *European Conference on a Service-Based Internet*, pp. 62-74. Springer, Berlin, Heidelberg.

Pulgatti, L. D. (2017): *Data Migration Between Different Data Models of NoSql Databases (Masters Dissertation)*. Graduate Program in Informatics, Sector of Exact Sciences, Universidade Federal do Paraná.

Rith, J.; Lehmayr, P. S.; Meyer-Wegener, K. (2014) March: Speaking in tongues: SQL access to NoSQL systems. *Proceedings of the 29th Annual ACM Symposium on Applied Computing*, pp. 855-857.

Roijackers, J.; Fletcher, G. H. (2013) July: On bridging relational and document-centric data stores. *British National Conference on Databases*, Springer, Berlin, Heidelberg, pp. 135-148.

Scavuzzo, M.; Tamburri, D. A.; Di Nitto, E. (2016) May: providing big data applications with fault-tolerant data migration across heterogeneous NoSQL databases. *IEEE/ACM 2nd International Workshop on Big Data Software Engineering*, pp. 26-32.

Sellami, R.; Bhiri, S.; Defude, B. (2015): Supporting multi data stores applications in cloud environments. *IEEE Transactions on Services Computing*, vol. 9, no. 1, pp.59-71.

Shirazi, M. N.; Kuan, H. C.; Dolatabadi, H. (2012) June: design patterns to enable data portability between clouds' databases. *12th International Conference on Computational Science and Its Applications*, pp. 117-120.

Singh, P.; Sawhney, R. S.; Kahlon, K. S. (2017) November: forecasting the 2016 US presidential elections using sentiment analysis. *Conference on e-Business, e-Services and e-Society*, pp. 412-423. Springer, Cham.

Strauch, S.; Andrikopoulos, V.; Bachmann, T.; Leymann, F. (2013) May: Migrating application data to the cloud using cloud data patterns. *Closer-Proceedings of the 3rd International Conference on Cloud Computing and Services Science*, pp. 36-46.

Yasrab, R.; Gu, N. (2016) July: multi-cloud PaaS Architecture (MCPA): a solution to cloud lock-in. *3rd International Conference on Information Science and Control Engineering*, pp. 473-477.