Hindawi
Scientific Programming
Volume 2020, Article ID 8813350, 13 pages
https://doi.org/10.1155/2020/8813350

Research Article

Automatic NoSQL to Relational Database Transformation with Dynamic Schema Mapping

Zain Aftab,[1] Waheed Iqbal,[1] Khaled Mohamad Almustafa,[2] Faisal Bukhari,[1] and Muhammad Abdullah[1]

[1]*Punjab University College of Information Technology (PUCIT), University of the Punjab, Lahore, Pakistan*
[2]*College of Computer and Information Sciences, Prince Sultan University Riyadh, 11586 Riyadh, Saudi Arabia*

Correspondence should be addressed to Waheed Iqbal; waheed.iqbal@pucit.edu.pk

Received 19 March 2020; Revised 2 June 2020; Accepted 5 June 2020; Published 1 July 2020

Academic Editor: Shah Nazir

Recently, the use of NoSQL databases has grown to manage unstructured data for applications to ensure performance and scalability. However, many organizations prefer to transfer data from an operational NoSQL database to a SQL-based relational database for using existing tools for business intelligence, analytics, decision making, and reporting. The existing methods of NoSQL to relational database transformation require manual schema mapping, which requires domain expertise and consumes noticeable time. Therefore, an efficient and automatic method is needed to transform an unstructured NoSQL database into a structured database. In this paper, we proposed and evaluated an efficient method to transform a NoSQL database into a relational database automatically. In our experimental evaluation, we used MongoDB as a NoSQL database, and MySQL and PostgreSQL as relational databases to perform transformation tasks for different dataset sizes. We observed excellent performance, compared to the existing state-of-the-art methods, in transforming data from a NoSQL database into a relational database.

1. Introduction

Traditional SQL-based relational database management systems (RDBMS) are famous due to efficient data management and ACID (atomicity, consistency, isolation, durability) properties. However, ACID properties restrict the ability of RDBMS to offer high scalability. Recently, the use of NoSQL databases has increased with cloud-based and large-scale applications as compared to traditional relational databases. Nonrelational databases support the management of structured, unstructured, and semistructured data in a nontabular form to offer more flexibility in handling big data efficiently [1]. Moreover, these databases are cost effective, highly available, schemaless, and scalable for managing massive data [2–5]. The schemalessness feature of NoSQL databases allows exceptional flexibility in managing heterogeneous data by allowing a different structure for each record, which also saves the time required to define the schema. Due to the schema flexibility, many companies adopted NoSQL databases for their data warehouses [6–8]. These features are attractive; however, most of the NoSQL databases offer eventual consistency instead of strong consistency. Many applications can afford to relax consistency for a short period; for example, social networks, analytic services, and data collection applications using sensors can afford a few milliseconds delay before the data propagate to all nodes. The eventual consistency model ensures that the updates are spread to all nodes within a specific time interval to ensure data consistency [9, 10].

ETL (extract, transform, load) tools are commonly used to extract data from a source database, transform it into the destination format, and then load the data into the destination database [11, 12]. The extract method involves fetching data from the source, which should be correct and accurate as the subsequent processes depend on it [13]. The transform phase follows a series of complicated data cleaning and conversion rules to prepare the data to be loaded to the destination format [14]. The transformation method in ETL is the most critical part, which requires complex validations to ensure schema differences in the

source and destination databases [13]. Mostly, domain experts having expertise on the source and destination databases are needed to supervise the transformation method to ensure proper data transformation, which is costly for many organizations [15]. The loading phase pushes the data to the destination source. The loading is relatively more straightforward process as it requires connecting with destination sources and dispatching data in batches. One of the most famous and industry practical ETL tools is Talend Open Studio (TOS) (https://www.talend.com/products/talend-open-studio/), which also offers NoSQL to SQL data transformation [16, 17]. However, TOS requires manual schema mapping, which is very difficult in the case of NoSQL, as most of these are schemaless and, in part, introduce challenges to the manual identification of the schema.

NoSQL databases have many advantages over relational databases; however, there are a few disadvantages. For example, these databases lack a standard interface and query language to manage data efficiently. Moreover, flexibility in schema introduces challenges to "extraction and transformation" methods for ETL tasks. NoSQL databases do not support joins and have no OLTP (online transaction processing) support, like SQL databases, which makes it challenging to perform complex analysis [18]. With the increase in data, the big data warehouses are being developed and requiring data from NoSQL data sources, and with these big data warehouses, the need for ETL tools is increasing. Many tools that perform ETL from SQL database to NoSQL database are available [19]; however, there are very few tools that can be used for the NoSQL database to SQL database ETL process. These tools require schema of the NoSQL database to work and execute queries sequentially, which makes them slow. The schema identification from a NoSQL database is one of the big problems due to the schemaless nature of NoSQL databases [20, 21]. Manual identification of the NoSQL database schema is a complicated, time-consuming, erroneous task and, therefore, is an expensive process. Hence, automatic schema discovery and mapping are essential. These issues give rise to the need for an ETL method that could automatically identify schema from the NoSQL database and perform ETL rapidly in a parallel manner.

In this paper, we proposed and evaluated an efficient ETL method for performing NoSQL to relational database migration automatically, which is capable of mapping schema from source NoSQL to destination RDBMS dynamically. The proposed solution requires a source NoSQL database and a destination RDBMS database to begin the ETL task. The system extracts the source data and then dynamically identifies the schema to populate it to the destination RDBMS. Once the extraction is completed, the system transforms the data into destination format and finally loads the data in batches. The proposed ETL tool can be easily used by non-ETL professionals to perform NoSQL to SQL data transformation with minimum cost, effort, and time. Our experimental evaluation using different sizes of NoSQL databases shows excellent performance compared to the ETL tool used in the industry. The main contributions of this paper include the following:

(1) Development of an efficient and automatic ETL method for NoSQL to relational database transformation
(2) Automatic schema identification of the given source NoSQL database and mapping it to a given destination relational database
(3) Performance evaluation of the proposed solution using multiple sizes of different NoSQL databases
(4) Comparison of the proposed system with existing state-of-the-art Talend Open Studio in terms of NoSQL to relational database ETL tasks
(5) Use of multiple relational database implementations in the evaluation to validate the performance of the proposed method

The rest of the paper is organized as follows. Related work is presented in Section 2. We provide the details of the proposed system in Section 3. The experimental design and setup are discussed in Section 4. Evaluations and results are presented in Section 5. Finally, conclusions and future work are discussed in Section 6.

2. Related Work

NoSQL databases are attraction attention because of flexible schema and high scalability features [22–24]. Some researchers have the problem of schema identification in document-based NoSQL databases [21]. The document-based NoSQL databases store data in JSON format because of the flexibility of this format [25]. Due to this flexibility in storing data, identifying schema from JSON files is a complex task. Izquierdo and Cabot [26] proposed a model-based approach, in which the overall schema is inferred from a small set of documents, and a model is generated based on that inference. Frozza et al. [27] introduced an algorithm that analyzes the schema from the document-based NoSQL database (MongoDB was chosen in their paper) [27]. However, the proposed algorithm does not consider the heterogeneity of data type against the same key in different documents; also, the algorithm is not efficient.

Many researchers have compared NoSQL and RDBMS databases. For example, Li and Manoharan [5] presented a survey on different NoSQL and SQL databases and compared their different aspects, including reading, writing, and deleting operations; the authors also discussed the benefits of NoSQL. Shim [28] discussed the CAP theorem; on its basis, a NoSQL database can only provide any of the two characteristics of consistency, availability, and performance. Brewer [29] stated the details of the CAP theorem and gave a proposition, in which, to configure the database, one can make a trade-off between the three CAP characteristics to allow a NoSQL database to have all the three characteristics of consistency, availability, and performance together. Boicea et al. [30] also presented a survey, in which they compare the syntax, speed, and features of MongoDB with Oracle, where MongoDB does not support OLTP and joins, whereas these are the key properties of SQL-based databases. Okman et al. [31] also presented a survey on the security

issues of NoSQL databases. The authors discussed the different security issues of MongoDB and Cassandra DB. Pore and Pawar [32] presented a survey, in which they discussed differences between SQL and NoSQL databases, highlighting the different properties supported by the SQL and not by NoSQL, such as ACID properties, transactions support, schema, and normalization.

ETL tools are widely used in data warehousing, where data are aggregated from different sources to perform data analytics. Ordonez et al. [33] discussed the integration of unstructured data into a relational data warehouse for better analytics and reporting. Gour et al. [34] improved the ETL process in data warehouses and listed some challenges to the improvement of the ETL processes. Skoutas and Simitsis [35] presented the concept of automatic designing of ETL processes using attribute mapping and identification of ETL transformations using semantic web technologies. Bergamaschi et al. [36] proposed a tool for the semantic mapping of the attributes from heterogeneous data sources by identifying the similarity of source schema to the data warehouse schema, which helps add the new data source to the already existing data warehouse. Bansal [37] proposed a framework for semantic ETL big data integration, which needs a manual ontologies creation. Prasser et al. [38] presented the anonymization of data in the ETL process of the biomedical data warehouse, and a plugin for the Pentaho Data Integration tool was built [39], which allows for data anonymization integration into the ETL processes and supports extensive data sources by using the stream-based processing of Pentaho.

Building solutions for data transformations between different sources is a hot topic. Ramzan et al. [40] proposed a data transformation and cleaning module to migrate data from a relational database to NoSQL-based databases. Kuszera et al. [41] presented a MapReduce method to migrate relational databases to NoSQL-based document and column family stores. Sellami et al. [42] proposed a set of rules for the automatic transformation of data warehouse into graph-based NoSQL stores. Hanine et al. [43] proposed and developed a sample application, which is used to migrate the data from RDBMS to NoSQL databases; the advantages of NoSQL over the SQL databases were discussed. Yangui et al. [44] proposed an ETL tool to transform the multidimensional data model into a NoSQL-based document store.

Nowadays, many tools are available for processing raw data by cleaning and transforming it into a specific destination format [45, 46]. Song et al. [47] discussed and reviewed the techniques used to transform XML data into a relational database. Zhu et al. [48] proposed an efficient transformation method of XML-based data into a relational database by analyzing complex XML schema. Many IT companies, including IBM, Informatica, Pervasive, Mongo Labs, Talend, and Pentaho, developed their ETL tools (http://www.jonathanlevin.co.uk/2008/03/open-source-etl-tools-vs-commerical-etl.html), which are specific to a given technology and require manual configurations and expertise to perform the ETL tasks.

There have been very few efforts to build NoSQL to relational database ETL techniques. Maity et al. [49] proposed a generic framework to transform NoSQL to relational store; however, the work required manual schema identification and also the efficiency of the system was not discussed. Some existing tools can also be used for NoSQL to relational database ETL tasks, but these tools also require manual schema mapping. In this paper, we propose and evaluate an efficient ETL tool for NoSQL to relational database transformation, which is capable of identifying the schema of NoSQL database automatically and then loading the data into SQL-based relational database efficiently.

3. Proposed System

3.1. Overview. The proposed system dynamically identifies schema and then extracts, transforms, and loads the data from NoSQL to a relational database. It is illustrated in Figure 1. The system workflow consists of the following steps:

(1) A new NoSQL to SQL ETL job is submitted to the Job Manager, which invokes the integrated Schema Analyzer to identify the schema from the source NoSQL database. The Schema Analyzer forwards the schema in JavaScript Object Notation (JSON) format. We explain the Schema Analyzer in Section 3.2.

(2) The JSON schema file is parsed and converted into a SQL query according to the destination database format for database creation, as will be explained in Section 3.3.

(3) After SQL database schema creation, the ETL processes are initiated for parallel processing of the data from NoSQL to SQL; the initiation of ETL processes will be explained in Section 3.4.

(4) ETL processes extract the data from the source database in batches; after extraction, data is processed to create queries in the format of the destination database, and then data is loaded to the destination database concurrently; the transformation from source to destination database will be explained in more detail in Section 3.5.

3.2. Schema Analyzer. The automated schema detection of NoSQL databases is a challenging task, and the proposed system achieves it automatically to create the destination SQL database schema. For this, we have used an open-source tool named *Variety* (https://github.com/variety/variety), a Schema Analyzer for MongoDB. Variety is developed to extract the schema of one collection at a time. To integrate it into our proposed system, we automated it to go over each collection in the NoSQL database and produce a schema in JSON format. MongoDB is a document-based database, which stores records in JSON format with great flexibility and without any restriction on the data type of values in the documents, where challenges to data processing for identifying the schema will be introduced. Moreover, data type restriction

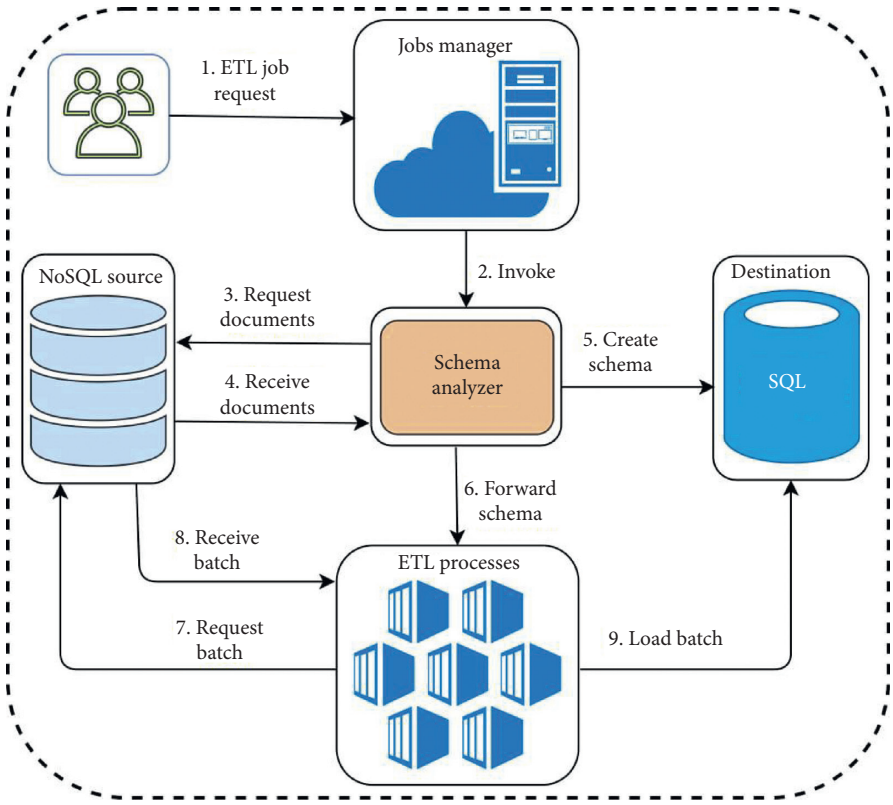

Figure 1: Proposed system overview.

on keys is not constrained in Variety. For example, one JSON document may have a key with a value type of number, and another document in the same collection could have the same key with a value of type string. Therefore, the key names in different documents may be the same while containing different types of data. This kind of heterogeneity in MongoDB documents requires some special handling. We have customized and relaxed the data type validation in the Variety tool to speed up the overall schema analysis process.

Algorithm 1 shows the overall strategy of the proposed Schema Analyzer for a single collection. The Schema Analyzer goes over each collection and identifies the required schema to be created in a relational database. Once the schema is identified, ETL processes are invoked to start performing transformation according to the received schema. The Schema Analyzer iterates over the NoSQL database collection, analyzing and recording all the distinct keys, and then outputs the schema in JSON format, which in part will be forwarded to the ETL processes. If a key has a value of type JSON document, or JSON array, it is then further parsed until a given depth (discussed in Section 3.3). The `isDocument` function receives the value against given key in a record; it returns true if the data type of value is a *JSON object*; otherwise, it returns false. In the same way, `isArray` receives the value of the key in the record and returns true if its data type is *JSON array* and false otherwise. Functions `parseDocument` and `parseArray` are written to perform the parsing of internal JSON documents/arrays recursively.

The asymptotic time complexity of Algorithm 1 is $O(r \times c)$, where r is the number of records, c is the number of columns in each record, and utility functions (`isArray` and `isDocument`) used in the algorithm are considered to consume constant time.

By default, MySQL and PostgreSQL provide support for inserting data using JSON format. For example, a new column type JSON is introduced in relational databases to store the entire JSON in one column. The stored JSON, a column, can be queried, but the performance of such queries for a large number of records is not good. Moreover, some utilities are also provided to import JSON documents; however, these utilities do not automatically create a schema and also do not address data type heterogeneity of the same keys in different documents. However, our proposed solution automatically creates the schema by analyzing JSON documents and also addresses the issue of data type heterogeneity.

3.3. Database Creation. The Schema Analyzer produces a schema in JSON format, which the system parses to build SQL queries for database and table creation. The queries are executed in the destination SQL database to create a database schema. In JSON, the objects are nested and also contain arrays. We dynamically parse those to create parent and child tables. After creating the tables, the remaining identification keys are parsed as respective columns of the tables, and the primary key is set to be _id, which is the default document ObjectId in a MongoDB collection. All the tables

```
Input: MongoDB collection (collection)
Output: Schema of collection in JSON format (schema)
foreach record r ∈ collection do
  foreach column_name c ∈ r do
    schema ⟵ schema ∪ c
    if isDocument(r["c"]) then
      parseDocument(r["c"])
    else if isArray(r["c"]) then
      parseArray(r["c"])
  end
end
```

Algorithm 1: Schema Analyzer strategy.

created for internal documents and arrays have their primary keys as auto-increment integer ids along with id of the main document. In our proposed system, we can configure JSON parsing for k internal levels; however, in our experimental evaluation, we used $k = 2$ to identify two levels of internal hierarchies in each JSON object. Figure 2 displays an example of JSON document in which the system dynamically parse to create SQL schema.

3.4. ETL Process Initiation. We implemented our ETL process in Node.js programming language and employed its Cluster (https://nodejs.org/api/cluster.html) package for initiating concurrent processes for ETL jobs. All processes are initialized by the master process to perform ETL. The master process dynamically distributes the work to each worker process. Once each process received its start point and limit of data by the master process, the data extraction is started.

Algorithm 2 shows the initiation of ETL processes. The master process is responsible for initiating the execution of standalone ETL processes. Each ETL process starts after receiving its individual logical partition information and connection information of MongoDB source as well as its MySQL/PostgreSQL destination. The master process uses the MongoDB connection information along with the collection name to process and identify the schema. The function initialize_schema takes the JSON schema of a collection as input and parses it to create a relational database schema using SQL queries. The variable limit represents the number of documents/records which are used to delegate the work the worker processes to perform the ETL. The master process is responsible for initializing schema in the destination database as it must be done before the execution of ETL processes. The system is capable of using n worker processes for the ETL job. In our experimental evaluation, we identified the optimal value of n to be used for improving the overall ETL job execution time. The identification of n is explained in Section 3.4.1.

The asymptotic time complexity of Algorithm 2 is $O(n \times l)$, where n is the number of processes and l is the time consumed by the function `create_process`, whereas other utility functions including `createConnections` and `initialize_schema` consume constant time.

```
{
  "_id":ObjectId("602f153e811c19417ae260ea"),
  "Key1":"Simple Key",
  "Key2":123,
  "IDoc":{
    "key1":"internal document key",
    "key2":1
  },
  "IArr":[
    {
      "k1":"internal array key"
    },
    {
      "k2":"another internal array key"
    }
  ]
}
```

Figure 2: A JSON document used for parsing and creating SQL queries.

3.4.1. Identifying Number of Worker Processes (n). The proposed ETL system uses multiple concurrent worker processes for each ETL job. We evaluate the MongoDB to MySQL ETL job for a different number of worker processes (n). We used $n = 2$ to 20, where n is the number of processes, and repeated each experiment three times. Figure 3 shows the average ETL execution time for using the different number of worker processes. Our proposed system shows the minimum time with $n = 8$ processes. Therefore, we used eight worker processes in our experimental evaluation.

3.5. Transformation and Loading. Each ETL process transforms the data from source database to destination concurrently. Figure 4 shows the workflow of the ETL process. Each process extracts data from the source NoSQL database in JSON format and sends it to the parser. The *Data Parser* parses the JSON data and forwards it to the Query Builder. The Query Builder receives the parsed data and starts to create insertion queries, in accordance with the schema of the MongoDB collection, in the format required by the destination database. After the Query Builder creates a batch insert query for the received data, it then forwards this query to the Query Executor. The Query Executor loads the data into the destination SQL database.

Input: SQL DB info (*dest*), Processes count (*n*), MongoDB connection info (*info*), collection name (*collname*), MongoDB schema (*schema*)
Output: Successful processes initiation
db ⟵ createConnection(*info*)
initialize_schema(*schema*)
length ⟵ db.getCollection(*collname*).count()
limit ⟵ (length/*n*)
for *j* ⟵ 0 to *n* do
  start ⟵ (length/*n*) ∗ j
  if *j* = *n* **then**
    limit ⟵ length − start
  create_process(*info*, *dest*, *schema*, *start*, *limit*)
end

Algorithm 2: Processes initiation.

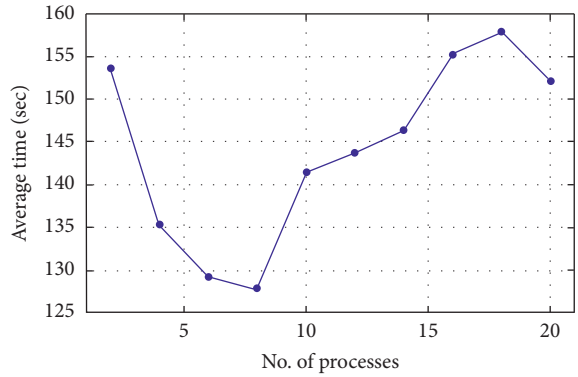

Figure 3: Average ETL job from MongoDB to MySQL execution time for a different number of worker processes (*n*).

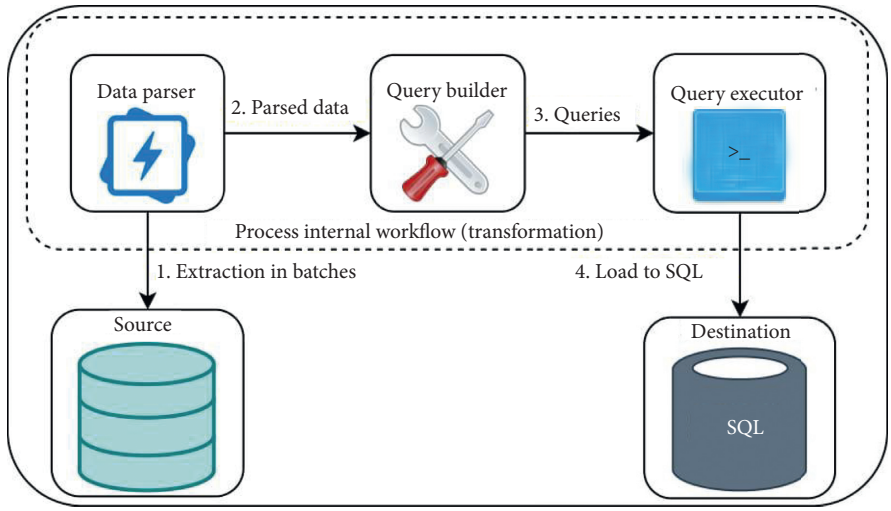

Figure 4: Single process workflow in proposed method.

Algorithm 3 shows the pseudocode for transformation and loading steps in the proposed system. Each ETL process opens connections with the source MongoDB and destination SQL database. Then, the database records are read in batches from the source database, and then the corresponding SQL queries are created for the retrieved NoSQL records. This is done by the ***createQuery*** function. The createQuery function reads the document from the MongoDB database and then parses the document, including its subdocuments and subarrays to create a query, using the parsed document data, for each subdocument and subarray. The set of queries made from the received document's data

are then added to the already created queries for batch execution. Finally, the batch of queries are then executed at the destination RDBMS. This process continues until complete NoSQL data is not transformed and loaded into the SQL database. This is done by each ETL process for its logical partition.

The asymptotic time complexity of Algorithm 3 is $O(b \times l)$, where b is the batch size and l is the maximum limit. In the algorithm $n = b \times l$ represents the total number of iterations required to complete the given ETL task. The utility functions `createConnections`, `readBatch`, and `createQuery` are constant time functions.

4. Experimental Design and Setup

We evaluated all experiments on a Core i7 machine with an octa-core CPU, 16 GB physical memory, and 2 TB hard disk running Ubuntu 14.04 (x64) operating system. In the following subsections, we briefly explain the baseline method, dataset generation, and experimental design used in the proposed system evaluation.

4.1. Baseline ETL Method. We used an open-source big data transformation tool, *Talend Open Studio (TOS)*, for NoSQL to SQL data transformation as the baseline tool to compare the proposed solution. *TOS* is a business intelligence tool providing various components for use in ETL jobs, and it is very famous in the industry. It provides a graphical user interface (GUI) for the creation of jobs, selection of different components, and definition of the flow of jobs by connecting the components. *TOS* automatically generates the Java-based code based on a manually created configuration.

We have selected TOS as a baseline because it is a free and open-source business intelligence tool. It provides a wide variety of components that can be used to perform business analytics for better business decision making. It is one of the most widely used business intelligence tools in the market. Talend was recognized as the market leader in 2019 by Gartner Magic Quadrant for data integration tools. Since it is widely used for ETL processing, it was one of the best choices as a baseline for comparison with our proposed system. In our experimental evaluation, we have configured TOS to use multithreaded executions to perform the ETL tasks.

In evaluations, we used different components of TOS including `tMongoConnection`, `tExtractJSONFields`, `MongoDBExtract`, `tMap`, `tMySQLInput`, `tMySQLOutput`, `tPostgresqlInput`, and `tPostgresqlOutput` for designing the two jobs. Figure 5 shows the configuration of the MongoDB to MySQL ETL job using TOS interface.

4.2. Dataset Generation. To evaluate our proposed method, datasets with different number of documents for MongoDB were generated with 100k, 500k, 1000k, and 5000k records in the NoSQL database (MongoDB collections). Each dataset was generated in the JSON format for MongoDB using the *Faker.js* package of Node.js (https://www.npmjs.com/package/faker). *Faker.js* provides multiple helper functions for generating meaningful data for experimentation. We build a data generator that uses various helper functions provided by *Faker.js* for data generation. Each helper function provided by *Faker.js* is able to generate a JSON object with particular set of key-value pairs. No two helper functions can generate a JSON document object containing the same keys and JSON object structure. We have used three different helper functions of *Faker.js* to generate records in each dataset. This ensured heterogeneity in the dataset. The datasets are generated using a data generator (*Faker.js*). The following different collections/databases are generated to evaluate the proposed system:

(1) *transaction*: it contains amount, date, business, name, type, and account keys

(2) *userCard*: it contains name, userName, e-mail, phone, website, subdocument address (street, suite, city, zipCode, and subdocument geo (longitude, latitude)), and subdocument company (name, catch phrase, and bs) keys

(3) *contextualCard*: it contains name, userName, avatar url, e-mail, dateofbirth, phoneNumber, website url, and subdocument address (streetName, secondaryAddress, city, zipCode, and subdocument geo (latitude, longitude)), and subdocument company (company name, catch phrase, and bs) keys

Each dataset is generated using the transaction, userCard, and contextualCard helper functions of the *Faker.js* in order to randomly generate meaningful data. This way of data generation ensures that each dataset has heterogeneous record schema. Figure 6 shows the relational tables (schema) for different JSON documents identified using the proposed Schema Analyzer.

4.3. Experimental Design. We evaluated our proposed NoSQL to SQL data transformation solution in two different experiments using MongoDB as a NoSQL source database and using two different destination SQL databases. Table 1 shows a summary of the conducted experiments.

In **Experiment 1**, we profile the execution time of MongoDB to MySQL ETL job using the proposed method and then compare the results with the job execution time using the baseline tool.

In **Experiment 2**, we profile the execution time of MongoDB to PostgreSQL ETL job using the proposed method and then compare it with the job execution time using a baseline ETL tool. In both experiments, we used a database with 100k, 500k, 1000k, and 5000k records to perform ETL jobs. Each experiment is repeated three times.

5. Evaluation and Results

5.1. Schema Detection Evaluation. We evaluate our proposed method by repeating each experiment three times. In each iteration of experiments, the schema detection is performed using the proposed Schema Analyzer. The Schema Analyzer analyzes the documents of the database and generates the schema in a JSON format, as explained in Section 3.2. Table 2

Input: MongoDB connection information (*info*), Destination DB info (*dest*), Data location start point (*start*), length to read (*limit*), Batch size (*batch_size*)
Output: Data loaded to destination SQL database.
db ← createConnection(*info*)
sql ← createConnection(*dest*)
queries ← {}
$n \leftarrow 0$
while *start* < *limit* **do**
  docs ← db.readBatch(*batch_size*)
  foreach doc ∈ docs **do**
    queries ← queries ∪ createQuery(*doc*)
    $n \leftarrow n + 1$
  end
  execute(*queries*)
  start ← start + *n*
end

Algorithm 3: Transformation and loading.

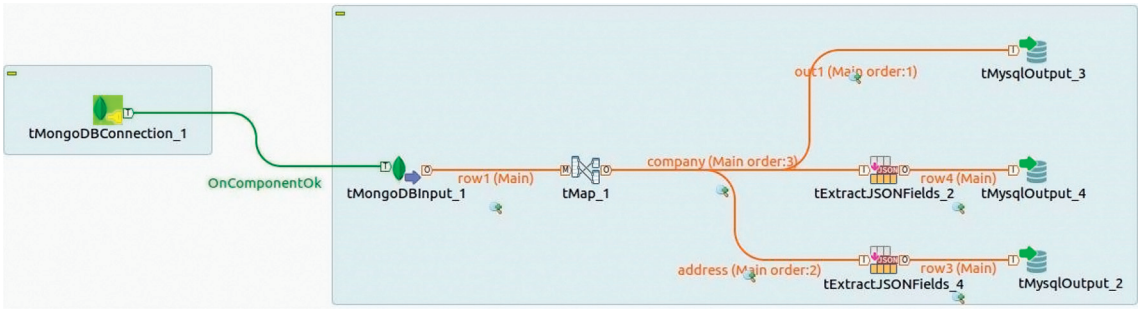

Figure 5: MongoDB to MySQL ETL job configuration using Talend Open Studio interface.

shows the schema analysis time for each dataset with 100k, 500k, 1000k, and 5000k database records. The schema detection time increases with the increase in the number of records in the dataset. Schema detection takes on average 5.39, 32.45, 51.23, and 257.60 seconds of each iteration of the experiment using 100k, 500k, 1000k, and 5000k database records, respectively. The result shows that the proposed schema detection method is efficient as it takes only 4.29 minutes for 5000k records to analyze and identify the schema for a large NoSQL database.

5.2. Experimental Results. In each iteration of experiments, we profile the execution time of the proposed ETL method, using MongoDB as a source and using MySQL and PostgreSQL as two separate destination databases, and compare it with the baseline method. We use 100k, 500k, 1000k, and 5000k database records for the ETL. Table 3 shows the ETL time required in each iteration of Experiment 1 and Experiment 2 for the proposed and baseline methods with the different record sizes. Our proposed method takes significantly less time using the record sizes 500k, 1000k, and 5000k in each iteration of the experiment as compared to the baseline method, for both MySQL and PostgreSQL databases as a destination. For 100k record size, two iterations of each experiment using the proposed method take less time as compared to the baseline; however, one iteration of Experiments 1 and 2 takes 0.86- and 0.31-second extra time using the proposed method as compared to the baseline method. Overall, the proposed method takes significantly less time as compared to the baseline. Moreover, we observe that the ETL from MongoDB to PostgreSQL is more efficient as compared to the ETL from MongoDB to MySQL.

Figure 7 shows the relative comparison of the proposed method with the baseline using different dataset sizes for Experiments 1 and 2. Our proposed method yields 3.4%, 68.3%, 71.0%, and 83.2% less time for MongoDB to MySQL ETL (Experiment 1) as compared to the baseline method for 100k, 500k, 1000k, and 5000k database records, respectively. The proposed method also outperforms the baseline method in MongoDB to PostgreSQL ETL (Experiment 2) by reducing the execution time by 13.4%, 59.9%, 59.4%, and 63.6% for the database record sizes 100k, 500k, 1000k, and 5000k, respectively.

Table 4 shows the throughput in each iteration of Experiments 1 and 2 for the proposed and baseline methods. Our proposed method outperforms the baseline method by yielding significantly higher throughput. The proposed

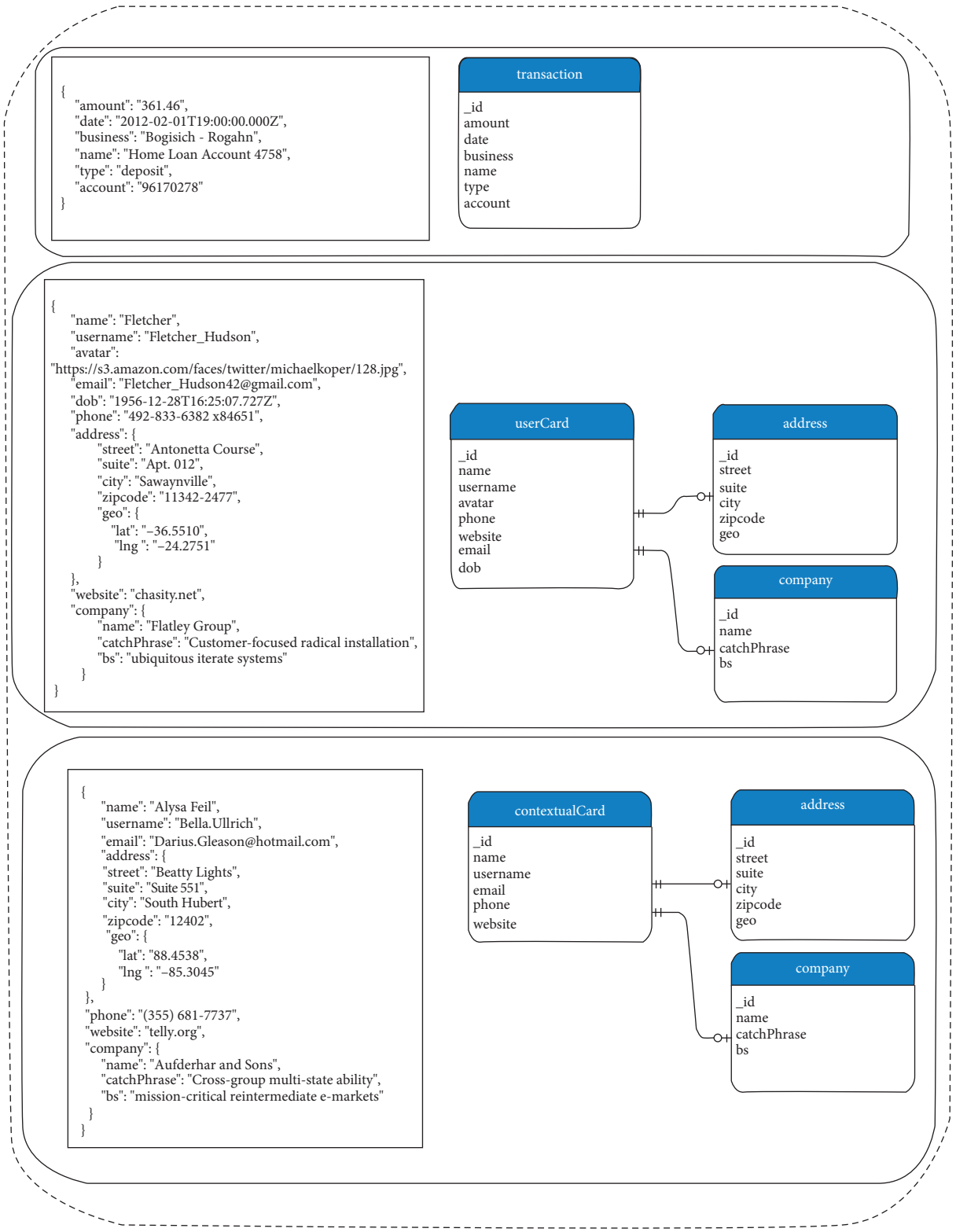

Figure 6: Relational tables (schema) for different JSON documents identified using the proposed Schema Analyzer.

method shows excellent throughput in ETL from MongoDB to PostgreSQL.

Figure 8 shows the relative comparison of throughput gain between our proposed method during Experiments 1 and 2 and the baseline method. The proposed method yields 1.13x, 2.52x, 2.46x, and 2.75x times higher throughput as compared to the baseline method during the ETL of MongoDB to MySQL (Experiment 1) for the database record sizes 100k, 500k, 1000k, and 5000k, respectively. In Experiment 2, the proposed method yields 1.03x, 3.23x, 3.46x,

Table 1: Experimental details.

Experiment	Description
Experiment 1: MongoDB to MySQL ETL	MongoDB to MySQL transformation with the proposed method using 100k, 500k, 1000k, and 5000k database records, compared with the baseline method
Experiment 2: MongoDB to PostgreSQL ETL	MongoDB to PostgreSQL transformation with the proposed method using 100k, 500k, 1000k, and 5000k database records, compared with the baseline method

Table 2: Schema detection time (sec).

Iterations	Number of records			
	100k	500k	1000k	5000k
1	5.33	32.12	50.52	253.22
2	5.30	31.94	51.94	263.22
3	5.55	33.30	51.24	256.35
Average	5.39	32.45	51.23	257.60

Table 3: ETL time for MongoDB to MySQL and PostgreSQL using the proposed solution and the baseline method.

Record size	MySQL		PostgreSQL	
	Baseline	Proposed	Baseline	Proposed
	43.15	31.12	25.32	23.99
100k	30.93	31.79	25.60	24.12
	35.37	31.78	25.39	25.70
Average	36.48	31.56	25.44	24.60
	210.75	75.49	150.79	37.40
500k	200.29	96.03	144.57	49.97
	201.19	74.11	148.62	53.37
Average	204.08	81.88	147.99	46.91
	333.41	131.62	234.40	65.13
1000k	329	134.78	239.69	74.66
	324.55	133.50	234.85	65.66
Average	328.99	133.3	236.31	68.48
	1615.36	586.18	1225.37	190.99
5000k	1647.16	587.60	1189.66	231.76
	1632.93	605.69	1240.32	189.83
Average	1631.82	593.16	1218.45	204.19

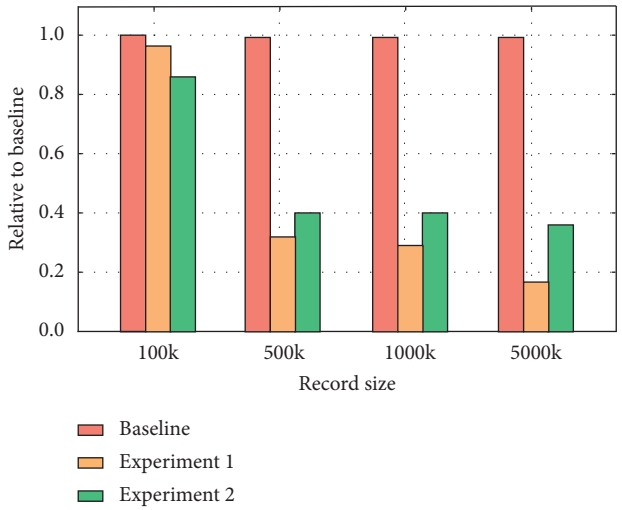

Figure 7: Execution time comparison of transformation and loading operations during Experiments 1 and 2 relative to baseline with different sizes of NoSQL databases.

and 6.02x times higher throughput compared to the baseline method for the record sizes 100k, 500k, 1000k, and 5000k, respectively.

Figure 9 shows the throughput improvements by increasing the number of database records using the proposed and baseline ETL methods for MySQL and PostgreSQL as the destination databases. The proposed method shows a significant increase in throughput with the increase in database record size. However, throughput almost remains the same using the baseline ETL method to increase the number of database records. Our proposed method improves the throughput of the given large size of ETL jobs, because of its concurrent processes execution.

In our proposed solution, the insertion speed of the destination relational database may affect the execution time of ETL jobs. We evaluate our proposed ETL method using two different destination databases. Results show that the PostgreSQL database performs better than the MySQL database as a destination because of the less insertion time of PostgreSQL as compared to MySQL. However, we do not

Table 4: Throughput (record per second) of the ETL job for MongoDB to relational databases (MySQL and PostgreSQL) using the proposed and baseline methods.

Record size	MySQL		PostgreSQL	
	Baseline	Proposed	Baseline	Proposed
	2317	3213	3949	4168
100k	3233	3146	3906	4146
	2827	3147	3939	3891
Average	2793	3169	3931	4068
	2372	6623	3316	13369
500k	2496	5207	3459	10006
	2485	6747	3364	9369
Average	2451	6192	3380	10915
	2999	7598	4266	15354
1000k	3040	7419	4172	13394
	3081	7491	4258	15230
Average	3040	7503	4232	14659
	3095	8530	4080	26179
5000k	3036	8509	4203	21574
	3062	8255	4031	26339
Average	3064	8431	4105	24698

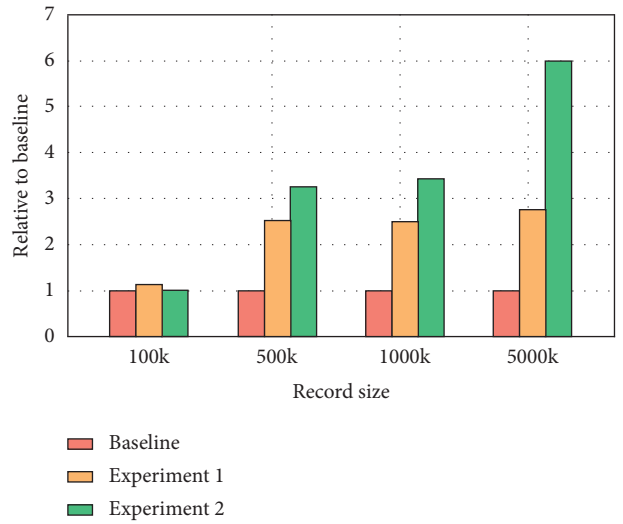

Figure 8: Throughput comparison of Experiments 1 and 2 to baseline with record sizes 100k, 500k, 1000k, and 5000k.

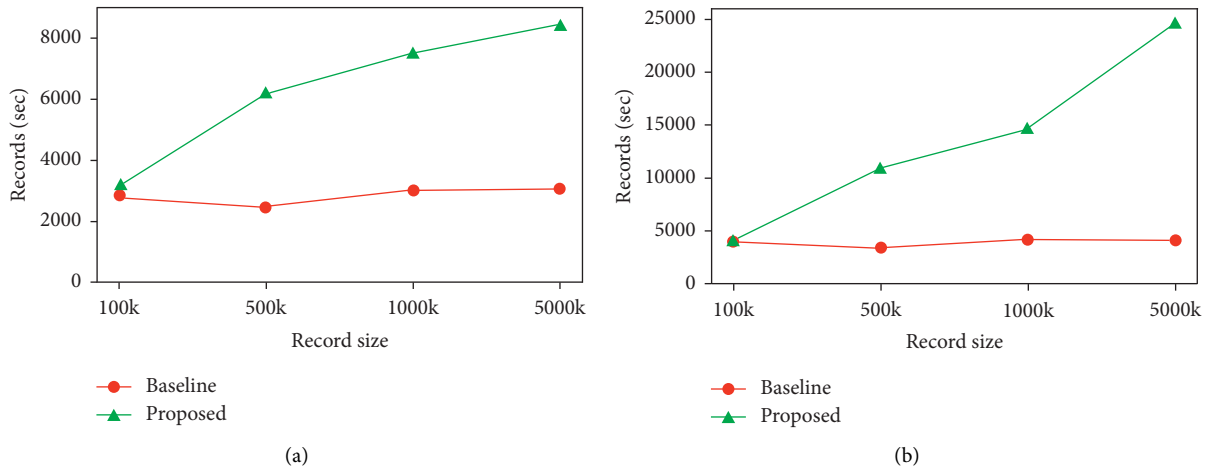

Figure 9: Throughput improvements for increasing number of records using the proposed ETL method and the baseline method for (a) MySQL and (b) PostgreSQL destinations.

observe any storage overhead for using PostgreSQL and MySQL in our proposed solution. The proposed system is generic to support any destination database; however, a basic connector needs to be added for any SQL database.

The conventional ETL systems like Talend Open Studio are dependent on manual schema configurations for successful ETL execution. However, our proposed system automatically identifies the schema to reduce the effort of manual schema configurations. Moreover, the proposed ETL method shows significantly less execution time as compared to the existing state-of-the-art baseline tool.

6. Conclusion and Future Work

NoSQL databases are most suitable for scalable systems and are on the rise. The existence of relational databases for easier management of the data is an important feature, due to the ACID and the SQL properties. In this paper, we have presented an efficient NoSQL to relational database migration system. Our experimental evaluation shows that the proposed method can automatically identify the schema of NoSQL for a relational database and then use concurrent processes to transform and load the data into the destination databases. Our experimental evaluation shows the scalability and performance compared to the existing state-of-the-art open-source tools. The proposed method is of benefit to ETL tasks specifically required to transfer data from the NoSQL database to the relational database.

As an extension to this project, different NoSQL implementations in the proposed system will be incorporated as the source databases, as well as incorporation of Hadoop and Spark for parallel data transformations [50] in order to improve the transformation time.

Data Availability

The data used in this research are available from the corresponding author upon request.

Conflicts of Interest

The authors declare no conflicts of interest.

Acknowledgments

The authors would like to thank Prince Sultan University, Riyadh, KSA, for partially supporting this work.

References

[1] A. Raut, "NOSQL database and its comparison with RDBMS," *International Journal of Computational Intelligence Research*, vol. 13, no. 7, pp. 1645–1651, 2017.

[2] J. Pokorny, "NoSQL databases: a step to database scalability in web environment," *International Journal of Web Information Systems*, vol. 9, no. 1, pp. 69–82, 2013.

[3] R. Kanwar, P. Trivedi, and K. Singh, "NoSQL, a solution for distributed database management system," *International Journal of Computer Applications*, vol. 67, no. 2, pp. 6–9, 2013.

[4] D. McCreary and A. Kelly, *Making Sense of NoSQL*, Manning, Shelter Island, NY, USA, 2014.

[5] Y. Li and S. Manoharan, *A Performance Comparison of SQL and NoSQL Databases*, IEEE, Piscataway, NJ, USA, 2013.

[6] Z. Bicevska and I. Oditis, "Towards NoSQL-based data warehouse solutions," *Procedia Computer Science*, vol. 104, pp. 104–111, 2017.

[7] M. Stonebraker, "SQL databases v. NoSQL databases," *Communications of the ACM*, vol. 53, no. 4, pp. 10-11, 2010.

[8] J. Han, H. Haihong, G. Le, and J. Du, "Survey on NoSQL database," in *Proceedings of the 2011 6th International Conference on Pervasive Computing and Applications*, Port Elizabeth, South Africa, October 2011.

[9] D. G. Chandra, "Base analysis of NoSQL database," *Future Generation Computer Systems*, vol. 52, pp. 13–21, 2015.

[10] D. Bermbach and S. Tai, "Eventual consistency: how soon is eventual? An evaluation of Amazon S3's consistency behavior," in *Proceedings of the 6th Workshop on Middleware for Service Oriented Computing*, pp. 1–6, Lisbon, Portugal, December 2011.

[11] P. Vassiliadis, "A survey of extract-transform-load technology," *International Journal of Data Warehousing and Mining*, vol. 5, no. 3, pp. 1–27, 2009.

[12] P. Vassiliadis and A. Simitsis, "Extraction, transformation, and loading," *Encyclopedia of Database Systems*, Springer, Boston, MA, USA, 2018.

[13] S. K. Bansal and S. Kagemann, "Integrating big data: a semantic extract-transform-load framework," *Computer*, vol. 48, no. 3, pp. 42–50, 2015.

[14] P. Vassiliadis, A. Simitsis, and S. Skiadopoulos, "Conceptual modeling for ETL processes," in *Proceedings of the 5th ACM International Workshop on Data Warehousing and OLAP*, pp. 14–21, New York, NY, USA, 2002.

[15] T. Jörg and S. Deßloch, "Towards generating ETL processes for incremental loading," in *Proceedings of the 2008 International Symposium on DATABASE Engineering & Applications*, pp. 101–110, ACM, Villa San Giovanni, Italy, June 2008.

[16] J. Sreemathy, S. Nisha, C. Prabha, and G. P. RM, "Data integration in ETL using Talend," in *Proceedings of the 2020 6th International Conference on Advanced Computing and Communication Systems (ICACCS)*, pp. 1444–1448, IEEE, Coimbatore, India, March 2020.

[17] R. A. Nisbet, *Data Integration with Talend Open Studio*, CiteSeer, Princeton, NJ, USA, 2010.

[18] N. Leavitt, "Will NoSQL databases live up to their promise?" *Computer*, vol. 43, no. 2, pp. 12–14, 2010.

[19] Y.-T. Liao, J. Zhou, C.-H. Lu et al., "Data adapter for querying and transformation between SQL and NoSQL database," *Future Generation Computer Systems*, vol. 65, pp. 111–121, 2016.

[20] D. Swami and B. Sahoo, "Storage size estimation for schemaless big data applications: a JSON-based overview," in *Intelligent Communication and Computational Technologies*, pp. 315–323, Springer, Berlin, Germany, 2018.

[21] J. Yoon, D. Jeong, C.-H. Kang, and S. Lee, "Forensic investigation framework for the document store NoSQL DBMS: MongoDB as a case study," *Digital Investigation*, vol. 17, pp. 53–65, 2016.

[22] M. Abourezq and A. Idrissi, "Database-as-a-service for big data: an overview," *International Journal of Advanced Computer Science and Applications*, vol. 7, no. 1.

[23] P. Atzeni, F. Bugiotti, L. Cabibbo, and R. Torlone, "Data modeling in the NoSQL world," *Computer Standards & Interfaces*, vol. 67, Article ID 103149, 2020.

[24] C. Asaad, K. Baïna, and M. Ghogho, "NoSQL databases: yearning for disambiguation," https://arxiv.org/abs/.

[25] S. B. Jatin, *MONGODB Versus SQL: A Case Study on Electricity Data*, Springer, Berlin, Germany, 2016.

[26] J. L. C. Izquierdo and J. Cabot, "Discovering implicit schemas in JSON data," in *International Conference on Web Engineering*, Springer, Berlin, Germany, 2013.

[27] A. A. Frozza, R. dos Santos Mello, and F. D. S. da Costa, "An approach for schema extraction of JSON and extended JSON document collections," in *Proceedings of the 2018 IEEE International Conference on Information Reuse and Integration (IRI)*, pp. 356–363, IEEE, Salt Lake City, UT, USA, July 2018.

[28] S. S. Y. Shim, "Guest editor's introduction: the CAP theorem's growing impact," *Computer*, vol. 45, no. 2, pp. 21-22, 2012.

[29] E. Brewer, "CAP twelve years later: how the "rules" have changed," *Computer*, vol. 45, no. 2, pp. 23–29, 2012.

[30] A. Boicea, F. Radulescu, and L. I. Agapin, "MongoDB vs Oracle—database comparison," in *Proceedings of the 2012 Third International Conference on Emerging Intelligent Data and Web Technologies*, pp. 330–335, Bucharest, Romania, September 2012.

[31] L. Okman, N. Gal-Oz, Y. Gonen, E. Gudes, and J. Abramov, "Security issues in NoSQL databases," in *Proceedings of the 2011 IEEE 10th International Conference on Trust, Security and Privacy in Computing and Communications*, pp. 541–547, Changsha, China, November 2011.

[32] S. S. Pore and S. B. Pawar, "Comparative study of SQL & NoSQL databases," *International Journal of Advanced Research in Computer Engineering & Technology*, vol. 4, no. 5.

[33] C. Ordonez, I.-Y. Song, and C. Garcia-Alvarado, "Relational versus non-relational database systems for data warehousing," in *Proceedings of the ACM 13th International Workshop on Data Warehousing and OLAP—DOLAP'10*, pp. 67-68, Houston, TX, USA, 2010.

[34] V. Gour, D. S. S. Sarangdevot, J. R. N. R. Vidyapeeth, G. S. Tanwar, and A. Sharma, "Improve performance of extract, transform and load (ETL) in data warehouse," *International Journal on Computer Science and Engineering*, vol. 2, no. 3, pp. 786–789, 2010.

[35] D. Skoutas and A. Simitsis, "Designing ETL processes using semantic web technologies," in *Proceedings of the 9th ACM International Workshop on Data Warehousing and OLAP, DOLAP'06*, ACM, New York, NY, USA, pp. 67–74, 2006.

[36] S. Bergamaschi, F. Guerra, M. Orsini, C. Sartori, and M. Vincini, "A semantic approach to ETL technologies," *Data & Knowledge Engineering*, vol. 70, no. 8, pp. 717–731, 2011.

[37] S. Bansal, "Towards a semantic extract-transform-load (ETL) framework for big data integration," in *Proceedings of the 2014 IEEE International Congress on Big Data*, pp. 522–529, Anchorage, AK, USA, June-July 2014.

[38] F. Prasser, H. Spengler, R. Bild, J. Eicher, and K. A. Kuhn, "Privacy-enhancing ETL-processes for biomedical data," *International Journal of Medical Informatics*, vol. 126, pp. 72–81, 2019.

[39] M. Casters, R. Bouman, and J. Van Dongen, *Pentaho Kettle Solutions: Building Open Source ETL Solutions with Pentaho Data Integration*, John Wiley & Sons, Hoboken, NJ, USA, 2010.

[40] S. Ramzan, I. S. Bajwa, B. Ramzan, and W. Anwar, "Intelligent data engineering for migration to NoSQL based secure environments," *IEEE Access*, vol. 7, pp. 69042–69057, 2019.

[41] E. M. Kuszera, L. M. Peres, and M. D. D. Fabro, "Toward RDB to NoSQL: transforming data with metamorfose framework," in *Proceedings of the 34th ACM/SIGAPP Symposium on Applied Computing*, pp. 456–463, ACM, Limassol, Cyprus, April 2019.

[42] A. Sellami, A. Nabli, and F. Gargouri, "Transformation of data warehouse schema to NoSQL graph data base," in *International Conference on Intelligent Systems Design and Applications*, pp. 410–420, Springer, Berlin, Germany, 2018.

[43] M. Hanine, A. Bendarag, and O. Boutkhoum, "Data migration methodology from relational to NoSQL databases, World Academy of Science, Engineering and Technology," *International Journal of Computer, Electrical, Automation, Control and Information Engineering*, vol. 9, no. 12, pp. 2369–2373, 2016.

[44] R. Yangui, A. Nabli, and F. Gargouri, "ETL based framework for NoSQL warehousing," in *European, Mediterranean, and Middle Eastern Conference on Information Systems*, pp. 40–53, Springer, Berlin, Germany, 2017.

[45] A. K. Bhattacharjee, A. Mallick, A. Dey, and S. Bandyopadhyay, "Enhanced technique for data cleaning in text file," *International Journal of Computer Science Issues (IJCSI)*, vol. 10, no. 5, p. 229, 2013.

[46] H. Mohamed, T. Leong Kheng, C. Collin, and O. Siong Lee, "E-clean: a data cleaning framework for patient data," in *Proceedings of the 2011 First International Conference on Informatics and Computational Intelligence*, pp. 63–68, IEEE, Bandung, Indonesia, December 2011.

[47] E. Song, S.-C. Haw, and F.-F. Chua, "Handling XML to relational database transformation using model-based mapping approaches," in *Proceedings of the 2018 IEEE Conference on Open Systems (ICOS)*, pp. 65–70, IEEE, Langkawi Island, Malaysia, November 2018.

[48] H. Zhu, H. Yu, G. Fan, and H. Sun, "Mini-XML: an efficient mapping approach between XML and relational database," in *Proceedings of the 2017 IEEE/ACIS 16th International Conference on Computer and Information Science (ICIS)*, pp. 839–843, IEEE, Wuhan, China, May 2017.

[49] B. Maity, A. Acharya, T. Goto, and S. Sen, "A framework to convert NoSQL to relational model," in *Proceedings of the 6th ACM/ACIS International Conference on Applied Computing and Information Technology*, pp. 1–6, ACM, Kunming, China, June 2018.

[50] B. Iqbal, W. Iqbal, N. Khan, A. Mahmood, and A. Erradi, "Canny edge detection and Hough transform for high resolution video streams using Hadoop and Spark," *Cluster Computing*, vol. 23, no. 1, pp. 397–408, 2020.