Influencing Migration Processes by Real-Time Data

Roman Ceresnak
University of Zilina,
Zilina, Slovakia
roman.ceresnak@fri.uniza.sk

Karol Matiasko
University of Zilina
Zilina, Slovakia
karol.matiasko@fri.uniza.sk

Adam Dudas
Matej Bel University,
Banska Bystrica, Slovakia
adam.dudas@umb.sk

***Abstract*—The current trend of information systems and informatization moving forward in big steps also influences database systems. This direction of a data transfer from the databases with a rigid structure to the systems with a free data structure created a vast amount of challenges, including problems with backward compatibility, the inconsistency of data types or possibly of data editing during the transformation transfer. As a solution to this problem, we created MigrationLayer which can manage the structural and data compatibility between a relational and nonrelational database which is divided into three modules: The first is module of transformation, respectively, of the data transfer from the relational database to the nonrelational database. The second module is based on the catching of the data during its' modification which could influence the data entering the system, and the third module is a module of backward compatibility, which, based on the data change, evokes a consolidation edit of the structure and of course, the data. Experiments show that MigrationLayer can catch the data incoming to the transformation module in real-time with sufficient effectiveness. This leads to reduction of the time needed for the modifications after the whole migration ends. The module of the backward compatibility effectively helps us to maintain the backward compatibility between the structures of relational and nonrelational database.**

I. Introduction

A development of human society influenced many areas of interest - such as industry, healthcare, transport, and - last but not least - informatization. With the data growth, it was necessary to adjust various data storages, which were considered to be sufficient enough in the previous phases of data research. A new type of storage space called NoSQL database was created. These databases use new techniques, which support parallel processing and data replication in several nodes to ensure better performance and data availability [9].

In contrast with the relational databases [6], NoSQL databases process and manage the vast data, which are characterized by 3V (volume, variability, velocity) [4]. NoSQL databases are needed for support of various applications, which need various levels of performance, consistency, availability, and scalability [1]. For example, social media such as Twitter and Facebook [14] generate terabytes of daily data exceeding the relational databases' processing possibilities. These applications demand high performance, but they don't have to demand strong consistency.

Nowadays, there are various types of NoSQL databases such as document databases, key-value databases, columns and graph databases [21]. However, objective which is common in all of these database types is the use of the data replication for improvement of efficiency, availability, and scalability of the data. The majority of NoSQL databases supports the alternative consistency instead of the database transactions ensuring strong data consistency. The possible consistency warrants that all actualizations achieve all replicas after certain delay. This principle works in specific applications such as social media, advertisement records, and so on. Some of the user applications, however, demand strong data consistency. For example this can mean that the data related to bank account does not have to be consistent in every data actualization.

Many companies turn to the NoSQL databases and use it to store and manage data, and existing relational database applications need to be transformed into NoSQL databases. However, the data schemes are different between these two types of databases, which means a steep learning curve for the users. Besides, the join operation is not as time-efficient in a NoSQL database as in a SQL database. Due to that, the conversion of a scheme is essential in an exchange of relational database for NoSQL database and, similarly, an import of the data from the relational scheme to NoSQL. Besides, it is crucial to ensure the high efficiency of the read operation after the schema conversion. Even if relational databases such as Oracle, SQL, Server and MySQL are different, they all share the same relational scheme. Each NoSQL database has its own data scheme [1].

This paper examines the data migration between relational database Oracle and a variety of widely used NoSQL databases (Key-Value, DynamoDB). DynamoDB is being used in various industries and by various organizations such as Airbnb, BMW, Nike, Netflix, Samsung, Duolingo, and so on. Similarly, Oracle is used by PNC Honeywell, Samsung Electronics, and so on. MongoDB and Oracle follow various models and principles of the design. The first one of these is the key-value database model, while the second one is the relational database.

We designed and implemented a new way of effective migration of the data from the relational database to the nonrelational database. We emphasize an effective manipulation with the data entering the process and correct influencing of the transformed data. This migration mechanism is influenced by a vast amount of the data, which can occur in the databases with various characteristics, data types, and structures of data storage.

Novelty of the approach presented in this article can be described as follows:

- We develop a new framework capable of effective work with the data incoming into the system and capability of influencing effectively the grouped data which have already entered the migration process,
- We design and implement new framework maintaining the backward compatibility between the relational and the nonrelational database, not only regarding the data but also regarding the data structure,
- We compare appropriate tools for the change of data structure, which enters into the processes,
- We have experimentally verified our framework from the point of view of efficiency,
- The method reduces the users' access to the data modification after the transformation.

The rest of this article is structured as follows:

Section II of this article presents and analyses research related to the work described in this article. We are focused on the area of NoSQL databases and transfer of data between SQL and NoSQL databases. Third section of the article is focused on description of our own architecture for real-time data catching. Fourth part of the work contains experimental verification of the architecture and presents result from these experiments. The work is closed with the conclusion of the research and potential future research areas related to the topic of this article.

II. Related Works

A comparison between the relational data models and nonrelational data models was already stated in many articles such as [21], [22], [11], [2] or [12]. In these works, the authors proved what types of operations are a better match for the database from an efficiency perspective and offer faster response compared to other types of databases. In the article [7], the authors analyzed and compared performance of SQL databases and NoSQL databases with the key-value type. The experiments in the articles [3] and [17] provided comparison of traditional database Oracle and database MongoDB.

Based on these comparisons and experiments, the article [16] provides more detailed results and analysis regarding the data efficiency than the article [3]. Authors of these research works identified problems and challenges in the big data processing with the help of MapReduce. The four main categories of these problems and challenges are (1) data storing, (2) big data analysis, (3) online processing, and (4) security and protection of personal data. Researchers [18] analyzed migration between various NoSQL databases oriented on the columns. It was suggested that it represents the data in a standard format, and it was responsible for the translation of the source database to the target database. Results presented in the article [18] are about 10% more effective than those presented in the article [10].

While comparing two articles ([8] and [20]) focused on the data transformation problem, we had a chance to see situations in which researchers use three parts during the migration of the data from the relational database to the nonrelational. Results of the research presented in the [20] are more effective and more easily expandable, based on the experiments and possible data migration, and offers a more general module for future work opposite work [8].

In many cases, the researchers use Apache Sqoop [19] and DataX [15] while transferring the data from the relational database to the nonrelational database. Apache Sqoop is based on the data transfer from the relational database to Apache Hadoop. Apache DataX is created on the principle of the data exchange between heterogeneous data sources.

It is possible to identify two basic contributions while solving the problem. The first contribution [5] is a framework for the transfer from SQL to NoSQL with the help of MapReduce. The principle of this method is based on the storing of all tables of the relational database to one table in HBase. The second contribution [16] represents three guidelines for transforming the relational scheme to HBase scheme based on data of the HBase model. It expresses the relationships between their schemes as a file of nested mapping schemes on the automatic transformation of relational data to the HBase representation. These three guidelines include a grouping of the correlated data to column family, addition of foreign key references if the one side needs the access to the data on the other side and merger of connected tables of the data to decrease the number of the foreign keys.

We came across work that deals with a time-oriented database architecture during our research in the paper [18], which manages undefined values and proposes a comprehensive classification of systems on transactions, accesses, and indices. Since various data types can enter our system, whether they are structured or unstructured, research related to the modeling of undefined values is described in the mentioned work. Furthermore, it covers synchronization processes using groups of data. The critical components of the mentioned article are solutions for effective data acquisition with emphasis on undefined values and states.

In summary, the previous works are focused primarily on making the reading performance better or on the migration of the data from NoSQL database in order to get high availability and scalability. However, they do not provide a sufficiently effective solution for the incoming data, which can enter during the migration process which is already running.

III. Architecture for Real-Time Data Catching

Research presented in this article is focused on a solving the situation resulting from neglecting of real-time data catching in the migration of the data from the relational database to the nonrelational one. In this process, the data entering the process during the data transformation run play a significant role - not only because of the possibility of influencing the data grouping when the transformation is already happening but mostly because of the necessary modification of the data in the target database. As was published in [7], the select operation is not equally effective in the nonrelational databases as in the relational databases.

We created an architecture that makes data catching in the transformation process possible in real-time. This prevents additional modification after the data transfer from relational database Oracle to nonrelational database DynamoDB.

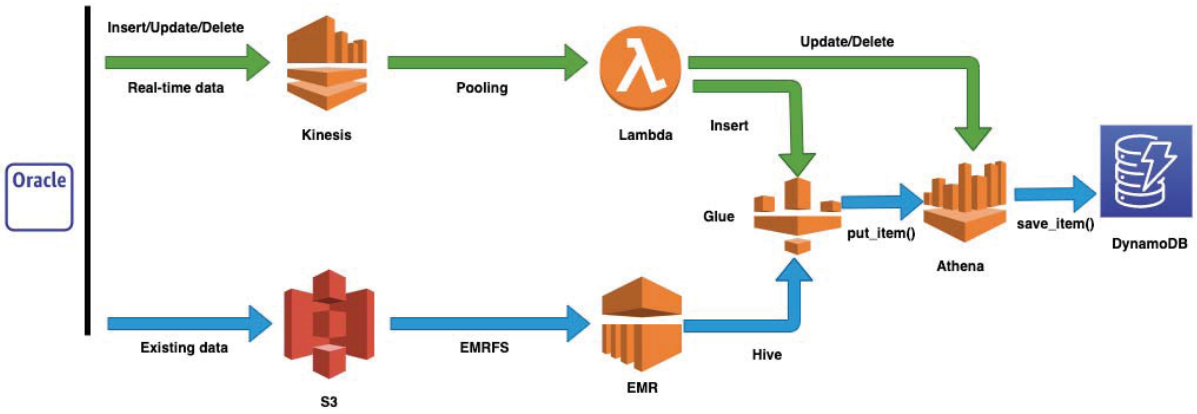

Fig 1. Design of new architecture for real-time data catching

This architecture consists of two parts:

- Upper branch (real-time data)
- Bottom branch (grouped data)

Real-time data are data, which enter the system during the run of transformation process. This means they are not stored in the database. However, they are automatically transferred to the process of transformation. The data operating in database Oracle for a longer time, for example, several days, months, or years, are called grouped data. It is necessary to subject these data to a particular type of data change. The data had a structure, and respectively, they were stored in the relational database. They have to be changed for the efficiency so that they could be manipulated effectively. Since there is considerable amount of the nonrelational databases, mentioned structure could suit the database type for which the transformation process was specific.

Various tools which fulfill a request for the significant modification of the data structures based on defined rules for the data modification process to another type can be used – some examples of these tools are Apache Flink, Spark, Samza Hadoop or Hive.

A. Uppear Branch

It is necessary to secure a state in which new instructions - such as insert, update, and deletion of the data - enter the system during the run of transformation process.

The values we are recording are for table *customers*, presented in the Fig. 2. The table consists of four INT type attributes, and a primary key is defined as a composite primary key, based on attributes *customer_id, order_id,* and *product_id.*

Fig 2. Customer data table

The upper branch serves as was mentioned above. The data entering the system are caught by the Amazon Kinesis tool. Amazon Kinesis makes catching, processing, and analyzing the data streams in real-time more accessible, helping us in getting the necessary information and quickly react to them. In the Oracle configuration we set up *binlong_format* to ROW to catch the transaction with module *binlogstreamreader*. We also set up the parameter value *log_bin* to allow *binlog*. A script for these settings looks as follow:

```
[oracleld]
secure-file-priv = ""
log_bin=/data/binlog/binlog
binlog_format=ROW
server-id = 1
tmpdir=/data/tmp
```

We created this script because of the data catching from relational database Oracle that catches the transactions and sends them directly to service Amazon Kinesis Streams. The script that needed to perform this process is provided at the address *https://github.com/romanceresnak/real-time-data-capture/blob/master/kinesis.py.*

```
INSERT INTO customer
(customer_id, order_id, product_id, quantity)
VALUES (5000, 2345,234,356);                (1)
UPDATE customer
SET quantity = 55
WHERE customer_id = 55;                     (2)
DELETE customer WHERE order_id = 50;        (3)
```

The given script represents illustrative JSON data generated by a python script. An attribute type defines the transactions recorded in a JSON type of record:

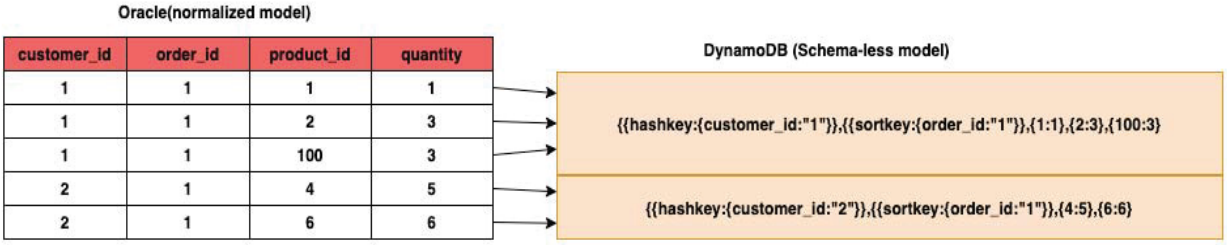

Fig 3. Mapping of data from Oracle DB to the DynamoDB

- *WriteRowsEvent(INSERT)*
- *UpdateRowsEvent(UPDATE)*
- *DeleteRowsEvent(DELETE)*

Here is an illustration of the data in JSON format.

```
{"table": "customer", "row": {"values": {"order_id": "1",
"quantity": 100, "customer_id": "74187", "product_id":
"1"}}, "type": "WriteRowsEvent", "schema": "test"}
```

```
{"table": "customer", "row": {"before_values":
{"order_id": "1", "quantity": 1, "customer_id": "74187",
"product_id": "1"}, "after_values": {"order_id": "1",
"quantity": 99, "customer_id": "74187", "product_id":
"1"}}, "type": "UpdateRowsEvent", "schema": "test"}
```

```
{"table": "customer", "row": {"values": {"order_id":
"100", "quantity": 1, "customer_id": "74187",
"product_id": "1"}}, "type": "DeleteRowsEvent", "schema":
"test"}
```

B. Bottom Branch

Data can enter the system in two possible ways. The first way is the creation of an export database that already exists in Amazon background. The export of every database in the Amazon background creates the data export to the S3 bucket in CSV format. The second way is creating relational database export on any device and, subsequently, the uploading of the file in CSV format to the Amazon S3 bucket.

```
SELECT * FROM customer WHERE <condition_1>
INTO OUTFILE '/data/export/customer/1.csv' FIELDS
TERMINATED BY ',' ESCAPED BY '\\' LINES TERMINATED BY
'\n';

SELECT * FROM customer WHERE <condition_2>
INTO OUTFILE '/data/export/customer/2.csv' FIELDS
TERMINATED BY ',' ESCAPED BY '\\' LINES TERMINATED BY
'\n';
...
SELECT * FROM customer WHERE <condition_n>
INTO OUTFILE '/data/export/customer/n.csv' FIELDS
TERMINATED BY ',' ESCAPED BY '\\' LINES TERMINATED BY
'\n'
```

Amazon AWS S3 sync was used for this purpose. This tool works internally with a function of multipart uploading of S3. A pattern matching can exclude or include specific files. In such case, when the synchronization fails during the processing, we do not have to transmit the same files again. This sync tool compares the size and time of editing of the files between the local versions and S3 versions and synchronizes only the local files, whose size and editing time are different from S3 files. The instructions look as follow:

```
aws s3 sync /data/export/purchase/
s3://<romanceresnak.aws >/purchase/

aws s3 sync /data/export/<romanceresnak/result>/ s3://<
romanceresnak.aws >/<romanceresnak/order>/
...
aws s3 sync /data/export/<romanceresnak/result>/ s3://<
romanceresnak.aws >/<romanceresnak/order>/
```

IV. Comparison of Object-Mapping Methods

This section of the presented article is concerned with the experiment related to a comparison of methods belonging to most popular object-mapping methods in big data. We compare how the change of the framework will influence the velocity of the data change in the transformation process. We chose two tools and procedures for these purposes:

- Apache Hive method with external table
- MapReduce method

1) Apache Hive with external table

We created the external table Hive opposite the data on S3 and inserted it to another extern table opposite table DynamoDB, with characteristic org.apache.hadoop.hive.dynamodb.DynamoDBStorageHandler. In the following sample code, we assume that Hive table for DynamoDB was created as an addition to the table *customer* as the column of the type ARRAY <STRING>. Columns *product_id* and *quantity_id* are aggregated and grouped according to *customer_id* and *order_id*, and inserted to the table *customer* with columns Collect UDAF and Brickhouse. A code providing the modification can be seen on this address https://github.com/romanceresnak/real-time-data-capture/blob/master/HiveMapper.txt .

Unfortunately, data types MAP, LIST, BOOLEAN, and NULL are not supported by the DynamoDBStorageHandler, so data type ARRAY String was chosen. The column of ARRAY type in Hive is comparable to the attribute of type StringSet in DynamoDB. The sample code mostly shows how Brickhouse works only for the data which are going to be aggregated in such a way, that more values of attributes will

be aggregated to one attribute of type StringSet in DynamoDB.

2) Apache MapReduce

The role of mapper is to read every record from input data in S3 and map the input pairs key-value to the middle pairs key-value. It divides the source data from S3 into two parts (key and value part) separated by the sign TAB. The mappers' data are organized according to their mediator key (*customer_id* and *order_id*) and are sent to the reduction part of computation. The records are inserted into DynamoDB in a reduced step.

We attach the script written in programming language Python with the name mapper.py:

```
#!/usr/bin/env python.
    import sys

    # get all lines from stdin

    for line in sys.stdin:

        line = line.strip()

        cols = line.split(',')

    # divide source data into Key and attribute part.

    # example output : "1,1    1,10"

  print '%s,%s\t%s,%s' %
(cols[0],cols[1],cols[2],cols[3] )
```

Generally, the role of reduction is to accept an output created after the processing of the mapping (coupled key/list of values) and then to perform the operation of reduction on the list of values according to every key.

In this case, the reduction is written in the Python language and it is based on stream: STDIN/STDOUT/Hadoop. The reducer accepts the data organized and ordered by the mediator key set in the mapper, in the identification *customer* and *order* (columns (0), columns (1)), and stores all attributes for the chosen key in dictionary *item_data*. The *item_data* dictionary attributes are inserted to or deleted from DynamoDB whenever a new mediator key comes from the server *sys_stdin*. The script we used can be found at the following address: https://github.com/romanceresnak/real-time-data-capture/blob/master/MapReduce.py, marked in the program as reducer.py.

We started the MapReduce task after writing the scripts, which connected local user to the main EMR node and started streaming Hadoop tasks. A location or name of file Hadoop-streaming.jar can be changed, depending on the version of EMR used. Particular messages occurring during the run of reducer are stored in a directory marked as *choice-output*. The values of hash key and the keys of the range are also recorded to find out which data cause exceptions or errors.

```
$ hadoop fs -rm -r
s3://<transformation>/<nosql/result>

$ hadoop jar /usr/lib/hadoop-mapreduce/hadoop-
streaming.jar \

      -input s3://< transformation
      >/<romanceresnak/input> -output s3://
      <romanceresnak/result>/<romanceresnak/output>\

      -file /<transformation>/mapper.py -mapper /<
      transformation >/mapper.py \

      -file /< transformation >/reducer.py -reducer /<
      transformation >/reducer.py
```

3) Connection of the branches

The connection of the branches always happens after the successful data modification in the upper and bottom branches of proposed model. This data transformation can last anywhere from couple of minutes to several hours and any amount of data updates, deletes and inserts can be enter the system during this modification.

All new data connected with operation insert are automatically stored to S3 with name transformation to the new.txt folder. After finishing the transformations of the grouped data, we used the script written in the programming language Python that makes it possible to connect all, which are result of transformations in the bottom branch - that is, the branch with the grouped data - and upper branch with the data entering the system during the transformation process. The script we created for this purpose is at the following address: https://github.com/romanceresnak/real-time-data-capture/blob/master/merge.py.

Amazon Glue will connect all the files into one sizable file called merge.txt and subsequently, data updates are done with the use of Amazon Athena, which is connected with the Lambda function. Amazon Athena compares the values situated in the queue of Lambda function with the values in file with the use of same principle as a relational database. If the operation update is in the queue of Lambda function, it is also situated in the file, this change is made by the management of the queue, a message from the front is deleted, and it continues like this until the queue is empty. After the emptying of the queue, the automatic storing of the values to the database DynamoDB follows.

We use the following settings for the server configuration:

TABLE I. SERVER CONFIGURATION

Oracle Instance	m4.2xlarge
EMR cluster	master : 1x m3.xlarge core : 2x m4.4xlarge
DynamoDB	2000 write capacity unit

We used following sizes of datasets:

TABLE II. TABLE OF DATA

Number of records	1,000,000 500,000,000 1,000,000,000
Database file size (.lbc)	4,9 GB 52,6 GB 108,4 GB
DynamoDB	2000 write capacity unit

We used our architecture and measured time of data modification. In the Table III, we present times for export to CSV file, upload to S3 (synchronisation) and import of the data to DynamoDB itself.

TABLE III. TABLE OF PERFORMANCE

Export to CSV	32 sec 4 min and 30 sec 6 min and 50 sec
Upload to S3 (sync)	22 sec 1 min and 48 sec 3 min and 30 sec
Import to DynamoDB	-

During the storing of the data, it is not possible to modify this data further. Therefore, the values entering the system during process of data storing are processed in an individual branch to DynamoDB. We created another script in Python to perform the mentioned operation, helping us catch the mentioned data. The script is located at the following address: https://github.com/romanceresnak/real-time-data-capture/blob/master/lambda.py.

4) Experiments on proposed model

It is essential to test the correctness of velocity operation and reliability of our newly created architecture. We designed following experiments for the verification of the model:

a) Finding out how the use of Apache influences the velocity of the process with an increasing amount of the data,
b) Finding out the advantages of the newly designed and implemented model with data modification after the transformation with an increasing amount of the data.

A. Comparison of computation time with the use of Apache MapReduce and Apache Hive

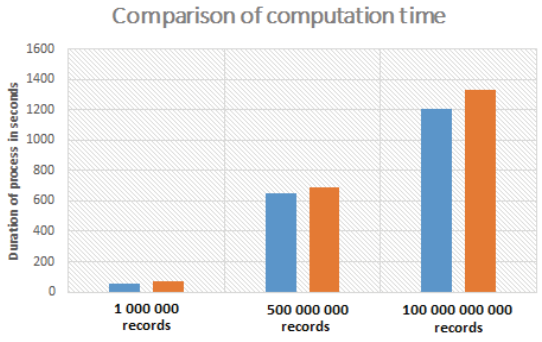

Fig 4. Comparison of computation time with the use of MapReduce (blue, left) and Hive (orange, right)

The values measured in this experiment will be used as agent in the setting of direction of our future research. As seen in Fig. 4 first pair of columns on the left, we measured computational time of 54 seconds for transformation with the use of MapReduce and 57 seconds with the use of Apache Hive. The difference between these values is related to the necessity of retyping which is needed during the data structure change from relational database Oracle to nonrelational database DynamoDB. Even with as small amount of the data as 1 000 000 records are, a difference can be observed.

When we used relational database with 500 000 000 records, the difference of computational time of this transformation was significant. As is presented in the middle pair of columns of Figure 4, the difference between using MapReduce and Hive was about 40 seconds in favor of MapReduce system. We do not assume that Apache Hive will become more effective with the increasing number of records.

Nevertheless, we experimented with our premise – we used dataset of 100 000 000 000 records and used the proposed transformation process on this dataset. We got the assumed conclusion – even for 100 000 000 000 records, time of computation was significantly higher with the use of Hive than while using MapReduce. The result of the third experiment shows, that method MapReduce was more effective (with time difference of approximately 124 seconds). This clearly shows that the tool fit for our further experimentation is system number 1 – Apache MapReduce, respectively Apache Hadoop. Even though MapReduce seems more effective in the presented case, it may not apply to every type of data. The results could be diametrically different in situations using the data types supported by apache Hive.

B. Advantages of proposed model with data modification after the transformation

Table IV presents comparison of experiments described in the section IV with and without the use of our proposed architecture while using the Apache Hadoop system.

TABLE IV. COMPARISON OF EXPERIMENTS WITH AND WITHOUT PROPOSED ARCHITECTURE WHILE USING APACHE HADOOP

Experiment	Number of records	Using our architecture	Without our approach	Number of operations
(1)	1,000,000	1 min	50 sec	3
(2)	10,000,000	6 min	6 min and 10 sec	18
(3)	500,000,000	12 min	13 min and 48 sec	36
(4)	1000,000,000	20 min	22 min and 24 sec	60

We used following operations for needs of testing:

```
UPDATE customer SET quantity = 55
WHERE customer_id = 55;
DELETE customer WHERE order_id = 50;
INSERT INTO customer (customer_id, order_id,
product_id, quantity)
VALUES (5000, 2345, 234, 356);
```

The values of attributes *quantity, customer_id,* and *order_id* were random based on the size of data which entered transformation process. These operations were sent to the database in the intervals of 20 seconds. In the Table IV, the

column on the far right shows the number of operations entering the process during the transformation.

While comparing the values in the table 4, we can state that in the first experiment (1) our proposed architecture was not significant addition from the velocity perspective compared to the process where modifications of data are already done in the database. In this case, the number of operations entering the process is equal to 3, which is not sufficiently effective from the point of view of file connecting with the use of grouped data and new data and following editing of the file.

The efficiency of our solution manifests itself more significantly with the increasing number of the data entering the process (experiment (2), (3), and (4), and we can state that these results point to the fact that our method is sufficient to use because of the higher efficiency. It is possible to relieve the user with the suggested architecture from performing the additional edits after the end of the transformation process.

VII. Conclusion

The majority of methods which transform data from the relational database to the nonrelational database is focused on the data already situated in the system. These methods focus on designing the new way of effectively de-normalizing the data to get these data transferred to the target database more easily. This article presents different point of view and shows that during the data transformation, catching the data in real-time is a significant part since it prevents necessity of data modification after the transformation process. Even if this architecture change brings additional costs, it significantly influences the time of computation needed for data editing in the target database. Thus, it makes the whole process of the data transformation more effective. We used this methodology on three simple transfers, and we showed that with the increasing number of data entering the system, our method becomes more and more effective. It's important that the data, which were influenced by the data entering the system, did not cause any data loss. This aspect makes our proposed method unique in comparison to other works e.g. [18].

According to our experiences, making of data rules from the original structure of data and query templates takes only a few minutes. On the other hand, power needed for execution of the actual implementation of the required changes depends mainly on the computational units provided, in which case architecture, which enables automatic scaling of computational units helps.

The data catching uses the fact that the data entering the process influence the data grouped in the system. It is necessary to point out that our algorithm catches the data and edits the grouped data, which did not enter the nonrelational database DynamoDB yet.

The changes which were no longer possible to make cause partial ineffectiveness of our method, but in the end, it is still sufficient to apply proposed method to the transformation process.

Our method could be appropriate while using the application or threads, which can influence each other and, based on the priorities, edit the data in real-time. We can imagine incoming data could serve on the principle of the transaction serving as superior tread during the data editing in parallel intervention to the user's account. We will apply this knowledge and designs to our future work.

Acknowledgment

This work was supported by Grant System of University of Zilina No. 1/2020. (8056).

References

[1] D. Abadi, "Consistency tradeoffs in modern distributed database system design," Comput. Comput. Mag., 2012.
[2] S. Binani, A. Gutti, and S. Upadhyay, "SQL vs. NoSQL vs. NewSQL- A Comparative Study," Commun. Appl. Electron., 2016.
[3] A. Boicea, F. Radulescu, and L. I. Agapin, "MongoDB vs Oracle - Database comparison," in Proceedings - 3rd International Conference on Emerging Intelligent Data and Web Technologies, EIDWT 2012, 2012.
[4] R. Casado and M. Younas, "Emerging trends and technologies in big data processing," Concurr. Comput., 2015.
[5] W. C. Chung, H. P. Lin, S. C. Chen, M. F. Jiang, and Y. C. Chung, "JackHare: a framework for SQL to NoSQL translation using MapReduce," Autom. Softw. Eng., 2014.
[6] E. F. Codd, "A Relational Model of Data for Large Shared Data Banks," Commun. ACM, 1983.
[7] R. Čerešňák and M. Kvet, "Comparison of query performance in relational a non-relation databases," in Transportation Research Procedia, 2019.
[8] S. Defit, "Intelligent Data Transformation Based on Knowledge Based," 2009 International Conference on Information and Multimedia Technology, Jeju Island, 2009, pp. 393-395, doi: 10.1109/ICIMT.2009.102.
[9] D. DeWitt and J. Gray, "Parallel Database Systems: The Future of High Performance Database Systems," Commun. ACM, 1992.
[10] K. Grolinger, M. Hayes, W. A. Higashino, A. L'Heureux, D. S. Allison, and M. A. M. Capretz, "Challenges for MapReduce in Big Data," 2014.
[11] L. Issac, "SQL vs NoSQL Database Differences Explained with few Example DB," The Geek Stuff, 2014.
[12] K. M. W., "SQL vs. NoSQL," 2010.
[13] M. Kvet, S. Toth, and E. Krsak, "Concept of temporal data retrieval: Undefined value management," Concurrency Computat Pract Exper, vol. 32, no. 13, Jun. 2019, doi: 10.1002/cpe.5399.
[14] A. Lakshman and P. Malik, "Cassandra - A decentralized structured storage system," in Operating Systems Review (ACM), 2010.
[15] H. Lei, M. Blount, and C. Tait, "DataX: An approach to ubiquitous database access," in Proceedings - WMCSA'99: 2nd IEEE Workshop on Mobile Computing Systems and Applications, 1999.
[16] C. Li, "Transforming relational database into HBase: A case study," in Proceedings 2010 IEEE International Conference on Software Engineering and Service Sciences, ICSESS 2010, 2010.
[17] W. Naheman and J. Wei, "Review of NoSQL databases and performance testing on HBase," in Proceedings - 2013 International Conference on Mechatronic Sciences, Electric Engineering and Computer, MEC 2013, 2013.
[18] M. Scavuzzo, E. Di Nitto, and S. Ceri, "Interoperable data migration between NoSQL columnar databases," in Proceedings - IEEE International Enterprise Distributed Object Computing Workshop, EDOCW, 2014.
[19] D. Vohra and D. Vohra, "Apache Sqoop," in Practical Hadoop Ecosystem, 2016.
[20] Z. Wei, J. Dejun, P. Guillaume, C. H. Chi, and M. Van Steen, "Service-oriented data denormalization for scalable web applications," in Proceeding of the 17th International Conference on World Wide Web 2008, WWW'08, 2008.
[21] C. Wodehouse, "SQL vs. NoSQL: What's the difference?," Upwork, 2016. .
[22] Xplenty, "The SQL vs NoSQL Difference: MySQL vs MongoDB," Xplent Blog. 2017.