Cross Platform (RDBMS to NoSQL) Database Validation Tool using Bloom Filter

Akansha Goyal, Arun Swaminathan and Rasika Pande
Department of Computer Engineering and IT
College of Engineering, Pune

Dr. Vahida Attar
Assistant Professor, Department of Computer Engineering and IT
College of Engineering, Pune

Abstract—Data - structured, semi-structured and unstructured is growing at an exponential rate. Most organizations are dependent on this huge volume of data to make real time decisions. To provide accurate information to these decision making models, it is crucial that the data is of superior quality. Since most organizations are migrating from relational to NoSQL databases, it is vital to validate data after the migration process. Considering the different structures of the two databases, the process of validation is a formidable task. This paper talks about an approach to validate data between cross platform databases using denormalized schema structures and bloom filters. Based on our experimental results, we have been able to validate huge datasets and pinpoint the exact corrupted records in constant space and linear time complexity up to a desired error probability.

Keywords— Relational database, NoSQL database, Validation, Data Migration, Cross-Platform, Denormalization, Bloom Filter, Big Data, Columnar NoSQL, Document store NoSQL, Schema Conversion

I. INTRODUCTION

Data forms the core part of every organization today. Due to the numerous database choices available, each organization uses diverse databases, each database specific to the problem they are trying to solve. Due to the high volume, different structures and criticality of data, migration is a risky affair. Also, due to the immense differences in the architecture between the relational and non-relational databases as well as the different data models of the diverse NoSQL databases [1] and unknown schema mapping techniques implemented by the user, there should be a well-defined strategy for testing and quality assurance of the data migration process. According to our research, validation of data between the two databases remains a significant problem and no such cross-platform open source validation tool is easily available.

We have attempted to provide a generalized algorithm to validate the transfer of data between Relational and NoSQL databases specifically the column family and document store type, irrespective of the type of relational and NoSQL databases and the schema mapping technique used. The process of validation is based on some key factors such as identifying null values and duplicate, mismatched and extra records. For the purpose of validation, we have used a bloom filter which is a space-efficient probabilistic data structure used to test whether or not an element is a member of a set.

This paper is structured as follows. Section II targets the need to migrate to NoSQL databases, existing schema mapping techniques, existing validation tools and bloom filters. Section III illustrates our methodology. Section VI demonstrates the application of our algorithm and corresponding results by providing two case studies on Cassandra and MongoDB. Section V is a conclusion of our work.

II. RELATED WORK

To understand the indispensability of this cross-platform validation tool, it is essential for us to understand the urgency to migrate from Relational to NoSQL databases.

A. Need for NoSQL database

With the advancements in the IT Industry, the use of Big Data, Cloud Computing, Web Applications and the Internet of things, the demands of the databases are continuously changing [2, 3, 4]. The extensive use of Analytics burgeoning in every industry has also contributed to the problem of storing and managing data. There is a need not only to manage data which is high in velocity, volume, veracity and variety but also to provide speed, scalability, reliability, continuous availability, cost reduction and location independence at the same time. Since the existing popular relational databases fail to deliver these demands [5], most organizations are migrating from relational to semi and unstructured databases. Primarily, there are four data storage strategies in NoSQL databases [6] which are column family, document store, key-value and graph.

B. Existing schema mapping techniques

According to our research, there is no standardized method to transform a SQL schema to an equivalent NoSQL schema. The most common practice among columnar NoSQL databases is the DDI (Denormalization, Duplication and Intelligent keys) design principle. Denormalization and duplication mean to re-aggregate related SQL tables into a single NoSQL table despite having redundant data in the single NoSQL table. Each row of the newly created NoSQL table should select one intelligent key called the row key for unique identification. Among the document store databases, broadly two data models exist to represent relationships between documents, that is, embedded data and references. References introduce normalization to describe document relationships, whereas embedding allows all related information to be stored in a single document. This leads to denormalization and duplication of data.

978-1-4673-9802-2/16/$31.00

Therefore typically, there are two general approaches [1] when trying to migrate data from RDBMS to NoSQL.

1. Direct Mapping, that is, data from the source database is translated into the data structures of the target database, without much change to the schema.

2. Intermediate Mapping, that is, data is translated into an intermediate format and then from this into the final one. The intermediate format is usually a denormalized schema which consists of joins and skipping some columns in RDBMS.

Even though the second approach requires an extra transformation, it is more flexible once the structure of the intermediate form is properly defined. Also, direct mapping may not be able to import the dependencies of tables in relational to non-relational databases which makes it impossible to perform join queries on the NoSQL database. The user must query more than one table [7] in the NoSQL databases to provide the same functionality as that of the relational databases.

Apart from this, there may even exist an abstract schema to make optimum use of that particular NoSQL database's features [8] like replication and consistency. Hence, it is difficult to base our algorithm depending on what type of schema mapping is chosen for the migration process. Therefore, we have assumed a generalized approach irrespective of the type of schema mapping used.

C. *Existing validation tools*

There are very few NoSQL validation tools available. Some of them are as follows -

1. Pentaho Data Integration is a client ETL application which supports NoSQL platforms such as MongoDB and HBase. It allows executing ETL jobs in and out of big data environments such as Hadoop. It also allows the use of user-defined validation rules such as null values allowed, decimal symbols, maximum string to verify data during the transformation process from the source to the target database.
2. Talend Open Studio is an ETL tool which supports various NoSQL databases such as Cassandra, MongoDB, HBase, CouchBase, CouchDB and provides the functionality to connect and integrate them.
3. Jaspersoft ETL is a tool distributed by TIBCO which supports NoSQL platforms such as Cassandra and MongoDB. Jaspersoft in partnership with Talend allows the use of Talend Open Studio in its own environment.

All these existing tools have some major drawbacks like they do not provide row and column comparisons over the complete data set. In this paper, we aim to overcome this drawback.

D. *Bloom filter*

Bloom filters are being used in the Industry for various operations. For example, Quora implemented a sharded bloom filter in the feed backend to filter out stories that people have seen before. Facebook used bloom filters for type ahead search to fetch friends and friends of friends to a user typed query. Apache HBase deployed bloom filter to boost read speed by filtering out unnecessary disk reads of HFile blocks which do not contain a particular row or column. There are numerous ameliorations to the classic bloom filter available such as Counting Bloom Filter [9], Dynamic Bloom Filter [10], Weighted Bloom Filter [11], K divided Bloom Filter [12], One Hashing Bloom Filter [13] and more.

The common factor among all these variations is the enormous space and time saving capability in contrast to the other data structures such as self-balancing binary search trees, hash tables or simple arrays and linked lists. Bloom filters also have the unusual property that the time needed either to add items or to check whether an item is in the set is a fixed constant, O(k). No other constant-space set data structure has this property and hence, we have decided to use this to solve the problem we have identified.

As we have seen, there is no generalized method for cross-platform database validation and the existing ones have major drawbacks. Our solution has solved this problem by not only revealing the exact corrupted records for massive datasets but also by achieving constant space and linear time complexity.

III. METHODOLOGY

In this section, we propose our approach to validate data transformation from various RDBMS to NoSQL databases. *Fig. 1*, outlines the flow of our algorithm.

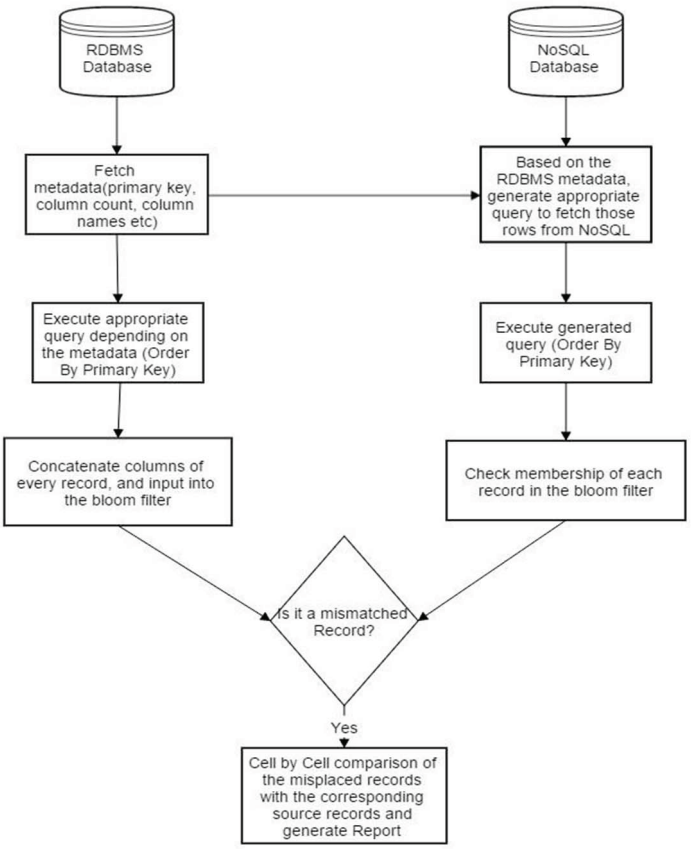

Figure 1. Validation flow diagram

Our algorithm broadly consists of the following steps -

1. Retrieve the metadata from the relational databases and use this metadata to create an identical mapping of the NoSQL database.

2. Transform both the databases into a similar format which forms the input to our core validation engine. The validation engine consists of two parts - 1) The bloom filter 2) The cell validation engine.

3. The source data forms the input to our bloom filter and the target data is validated against it.

4. The mismatched data found from the filter is passed on to the cell validation engine which tells us exactly which cell is corrupted.

A. Metadata mapping from RDBMS to NoSQL database

NoSQL databases by design do not have a fixed schema definition, and it is difficult to validate them with a relational database unless they share a common structure. In order to transform the distinct schemas into similar models, it is essential that the mapping of relationships between data for both the databases is known. Metadata for an entire NoSQL database such as column-count, column-name, data type, primary keys, etc. is generally not available at a table level. In case of columnar NoSQL databases, the column count per row is flexible. Similarly, document driven databases allow fields to differ per document and even support complex structures such as arrays and embedded documents. Our proposed algorithm for cross-database validation hence cannot assume the availability of a particular parameter in the database metadata. Thus, a mapping is required from RDBMS to NoSQL database.

On the other hand, relational systems provide a more straightforward access to their metadata. The provision of complex queries consisting of joins allows the denormalization of data from multiple tables. Hence, we have chosen the relational database for creating the mapping structure.

We begin by joining the appropriate tables in the relational database. Using the metadata obtained from the above operation, we construct the mapping based on which, data is extracted from the NoSQL database.

For example, let us assume in our RDBMS database we have the tables shown in *Fig. 2.1* and *Fig. 2.2.*

UserID	Name	CarID

Figure 2.1. User Table in RDBMS

CarID	CarName	CarManufacturer

Figure 2.2. Car Table in RDBMS

While migrating to NoSQL database, the user has stored these tables in the denormalized form shown in *Fig. 2.3*, so that he can access both the user and car details in a single query.

UserID	Name	CarID	CarName	CarManufacturer

Figure 2.3. User - Car Table in NoSQL

In a columnar NoSQL database, every row need not have all the columns given in *Fig. 2.3*. In our example, if the user has a car, all the columns will be retrieved whereas if he doesn't have a car, only the first two columns will be retrieved. Hence, in order to validate the migration between the two databases we join the user and car tables in the RDBMS database. Based on the metadata extracted from the join operation, the corresponding values are extracted from the NoSQL database. In case the column is not present, a null value is inserted in its place. It has to be kept in mind that the data is retrieved in the same order from both the databases so that bucketing can be done for larger datasets to increase efficiency in the later steps.

B. Transformation into a common structure

The datasets retrieved from their respective databases in the previous step will most likely be present in different data structures depending on the driver used to query the database. Since these structures are different in architecture, they cannot be compared directly. Hence, for each record we concatenate all the columns into a format which can form the input to the bloom filter. We have to keep in mind that if the primary key is null, it should be removed from this dataset. Also, if two records in the target database have the same primary key, they are considered as duplicate records.

C. Bloom filter

To avoid unnecessary comparisons and to reduce memory consumption, we have decided to use bloom filter before the cell - cell validation process.

Every record of the source dataset is input into the bloom filter. Then, every record of the target dataset is checked for membership by the bloom filter. The bloom filter gives us two possible outputs - 1) the target record is definitely not in the source 2) the target record is maybe in the source

There are several techniques proposed to reduce the probability of false positives. The probability of getting a false positive is given by

$$1 - e^{\frac{-kn}{m}} \quad (1)$$

Equation (1) describes the false positive probability where k is the number of hash functions, m is the number of bits in the array and n is the number of elements inserted.

This is not strictly correct as it assumes independence for the probabilities of each bit being set. However, assuming it is a close approximation we have that the probability of false positives decreases as m increases, and increases as n increases.

Since the number of elements inserted can be predetermined, we can create buckets of an optimum number of elements to reduce the false positives. Also, the number of bits set m can be calculated based on the desired false positive probability p and the number of elements inserted in the bloom filter n.

$$m = \frac{-nlnp}{(ln2)^2} \quad (2)$$

Equation (2) gives the number of bits in the array.

Thus, we are able to figure out all the records in the target dataset which are not in the source dataset up to the desired false positive probability. These mismatched records are further passed on to the cell validation engine.

D. Cell validation engine

We query the source database for the mismatched records which we have obtained from the previous step using its primary key. If the mismatched record doesn't exist in the source, it is counted as an extra record otherwise we compare the source record and the mismatched record cell by cell to find the exact error field. Also, if the same primary key is present in more than one mismatched record, it is considered to be duplicate record. Finally, the report is generated showcasing the errors encountered in the process.

To further improve the performance, the buckets can be processed in parallel and the individual reports can be combined in the end.

IV. EXPERIMENTAL RESULTS

Based on the algorithm proposed in *Fig. 1*, this section verifies the correctness of our validation model using two distinct NoSQL databases - Cassandra and MongoDB. We performed experiments varying the number of records in each NoSQL database. We fixed the following parameters - false positive probability p = 0.01 and number of errors = 20. The number of bits m is calculated by (2). We used a system with the following specifications - 8GB RAM, Intel i7 4th Generation, 2.4 GHz X8.

A. MongoDB

Source database - MySQL version 5.7

Target database - MongoDB version 3.2

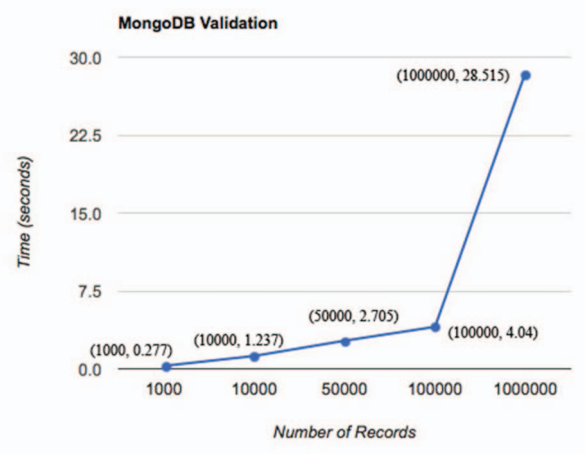

Figure 3.1. Variation of time taken for validation in MongoDB

To validate the migration from MySQL to MongoDB, there is a need to handle features that are specific to MongoDB. We have added support for validating embedded documents by querying multiple MySQL tables. The data extraction is also optimized by fetching data in buckets, the size of which can be determined based on system architecture. According to our experimentation, for 1 million records, the total validation time was 28.515 seconds only.

B. Cassandra

Source database - MySQL version 5.7

Target database - Apache Cassandra 2.1.12.1046

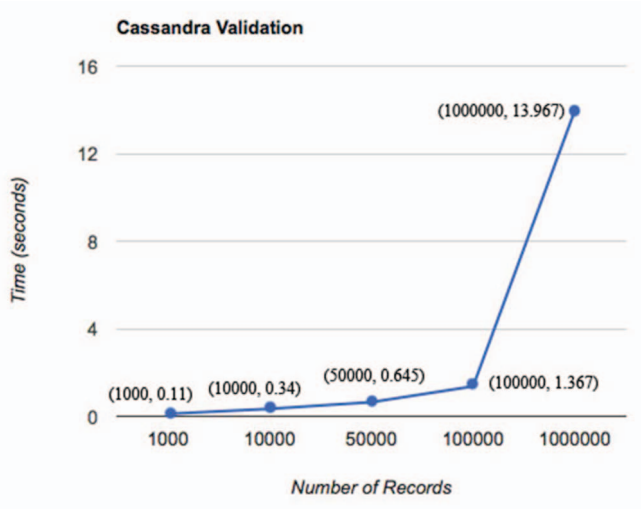

Figure 3.2. Variation of time taken for validation in Cassandra

To validate the migration from MySQL to Cassandra, the key change is not ordering the records. On compromising the bucketing of records, we make 1 bloom filter for the entire source database, and on it check membership of the entire target database. According to our experimentation, for 1 million records, the total validation time was 13.576 seconds only.

V. Conclusion

This paper proposes a generalized, fast and space efficient algorithm to validate RDBMS to NoSQL migration. The experiments achieved desirable values when we tested with different sizes of test data. The main limitation of this method is the small probability of false positives which can be eliminated by various optimizations made to the bloom filter. Future work will explore ways to parallelize the entire process and dynamically obtain bucket sizes for quicker evaluation.

References

1. Scavuzzo, M.; Di Nitto, E.; Ceri, S., "Interoperable Data Migration between NoSQL Columnar Databases," in Enterprise Distributed Object Computing Conference Workshops and Demonstrations (EDOCW), 2014 IEEE 18th International , vol., no., pp.154-162, 1-2 Sept. 2014
2. T. C. Hsu, D. M. Chang and H. J. Lee, "The Study of Application and Evaluation with NoSQL Databases in Cloud Computing," *Trustworthy Systems and their Applications (TSA), 2014 International Conference on*, Taichung, 2014, pp. 57-62
3. P. Bednar, M. Sarnovsky and V. Demko, "RDF vs. NoSQL databases for the semantic web applications," *Applied Machine Intelligence and Informatics (SAMI), 2014 IEEE 12th International Symposium on*, Herl'any, 2014, pp. 361-364.
4. A. Zimmermann, R. Schmidt, K. Sandkuhl, M. Wissotzki, D. Jugel and M. Mohring, "Digital Enterprise Architecture - Transformation for the Internet of Things," *Enterprise Distributed Object Computing Workshop (EDOCW), 2015 IEEE 19th International*, Adelaide, SA, 2015, pp. 130-138.
5. C. Győrödi, R. Győrödi, G. Pecherle and A. Olah, "A comparative study: MongoDB vs. MySQL," *Engineering of Modern Electric Systems (EMES), 2015 13th International Conference on*, Oradea, 2015, pp. 1-6.
6. Chao-Hsien Lee; Yu-Lin Zheng, "SQL-to-NoSQL Schema Denormalization and Migration: A Study on Content Management Systems," in *Systems, Man, and Cybernetics (SMC), 2015 IEEE International Conference on* , vol., no., pp.2022-2026, 9-12 Oct. 2015
7. Gansen Zhao; Qiaoying Lin; Libo Li; Zijing Li, "Schema Conversion Model of SQL Database to NoSQL," in *P2P, Parallel, Grid, Cloud and Internet Computing (3PGCIC), 2014 Ninth International Conference on* , vol., no., pp.355-362, 8-10 Nov. 2014
8. G. C. Deka, "Tutorial on NoSQL Databases," Mobile Cloud Computing, Services, and Engineering (MobileCloud), 2015 3rd IEEE International Conference on, San Francisco, CA, USA, 2015, pp. 1-3.
9. Bonomi, Flavio, et al. "An improved construction for counting bloom filters." Algorithms–ESA 2006. Springer Berlin Heidelberg, 2006. 684-695
10. D. Guo, J. Wu, H. Chen, Y. Yuan and X. Luo, "The Dynamic Bloom Filters," in *IEEE Transactions on Knowledge and Data Engineering*, vol. 22, no. 1, pp. 120-133, Jan. 2010.
11. J. Bruck, J. Gao and A. Jiang, "Weighted Bloom filter," *Information Theory, 2006 IEEE International Symposium on*, Seattle, WA, 2006, pp.
12. X. G. Liu, J. Lee, G. Wang, G. J. Xie and J. Liu, "K-Divided Bloom Filter Algorithm and Its Analysis," *Future Generation Communication and Networking (FGCN 2007)*, Jeju, 2007, pp. 220-224
13. J. Lu et al., "One-hashing bloom filter," Quality of Service (IWQoS), 2015 IEEE 23rd International Symposium on, Portland, OR, USA, 2015, pp. 289-298.