

Data migration from SQL to NoSQL using snapshot- Livestream Migration

Avnish Panwar

Asst. Professor, Department of Comp. Sc. & Info. Tech., Graphic Era Hill University,
Dehradun, Uttarakhand India 248002

Article Info

Page Number:1600 - 1608

Publication Issue:

Vol 70 No. 2 (2021)

Abstract

The process of moving data from a source database to a destination database is known as data migration. For a variety of reasons, including higher data handling capacity, improved speed, and scalability, many businesses are choosing to convert their databases from one kind (e.g., RDBMS) to another (e.g., NoSQL). Sqoop [3], mongoimport [2], and mongify [1] are a few techniques and technologies that have been developed to help with this transition from RDBMS to NoSQL databases. NoSQL databases use different models, as opposed to the relational model employed by RDBMS, including document, graph, and key-value. Large data volumes were the main focus of the design of NoSQL databases. The database migration model we provide in this paper can effectively transfer both real-time and historical data in parallel. Our Java-based model focuses on transferring data from MongoDB, a document-oriented NoSQL database, to MySQL, an RDBMS. The prototype we created can migrate both live data and a snapshot of the database at a particular moment in time simultaneously. Our experimental evaluation shows that, in terms of performance for both snapshot and live data migration, our model beats competing approaches.

Article History

Article Received: 05 September 2021

Revised: 09 October 2021

Accepted: 22 November 2021

Publication: 26 December 2021

Keywords: Database migration, Schema Design Reengineering, RDBMS, NoSQL

I. Introduction

Relational databases have been widely used by organisations for storing and analysing enterprise data for the past three to four decades [4]. These databases' use of the relational paradigm enables the systematic and orderly storage of information. Relational databases work well in situations where data adheres to a predetermined structure and where data growth is moderate. However, there has been an exponential increase in data generation recently from a variety of sources. Big data, which includes the three fundamental ideas of volume, variety, and velocity (the "3 V's of big data"), is a term that is frequently used to refer to this huge and varied body of data. Both organised and unstructured big data are possible. A new kind of database known as NoSQL (sometimes referred to as non-relational or not merely SQL) has evolved to address the problems brought on by huge data. To meet the demands of huge data, software businesses like Amazon, Facebook, and Google have embraced NoSQL databases. NoSQL databases offer adaptable data formats, scalability, and the capacity to effectively manage massive amounts of data. They have been created

to take into account big data's many forms, quick expansion, and high velocity. Companies like Amazon and Google have created their own NoSQL databases to meet their unique requirements. For instance, Google created Bigtable whereas Amazon introduced DynamoDB [5]. These databases were created to meet the specific requirements and difficulties that these businesses faced when handling and processing massive amounts of data.

Amazon developed the fully managed NoSQL database service known as DynamoDB. For applications that require consistent and predictable performance, it delivers seamless scalability, minimal latency, and excellent performance. It is appropriate for a variety of use cases, from modest applications to large-scale systems.

Bigtable is a distributed, extremely scalable NoSQL database that was created by Google. Massive volumes of organised and semi-structured data can be stored there. Google uses Bigtable to offer quick and effective data retrieval over a distributed infrastructure for a variety of applications, including Google Search and Google Maps.

Companies like Amazon and Google were able to customise the features and functionality of these databases to suit their own needs by developing their own NoSQL databases. These databases have demonstrated to be crucial in handling the big data concerns and supporting the heavy workloads of these businesses.

High performance, availability, and scalability are characteristics of NoSQL databases [8]. NoSQL databases, in contrast to relational databases, enable data storage without the requirement to predetermine a rigid database schema. In contrast, before saving data, relational databases demand that users establish the structure, including table names, columns, and data types.

NoSQL databases are exceptional at horizontal scaling, which allows them to effectively handle growing data loads by dispersing the data over several servers. Unlike relational databases, which often rely on vertical scalability, which entails the addition of more resources to a single server, this approach allows for greater flexibility.

There are different consistency models for relational and NoSQL databases. According to the BASE (Basically Available, Soft state, and Eventually Consistent) model, NoSQL databases prioritise high data availability even if it results in brief inconsistency. Relational databases, on the other hand, follow the ACID (Atomicity, Consistency, Isolation, and Durability) model, which places a strong emphasis on precise consistency.

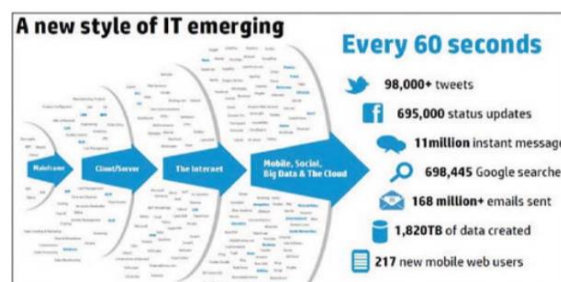


Figure 1: Data Generation in 1 Minute (60 Sec)

For typical relational databases, the exponential expansion of data generated by social media sites within a period of 60 seconds, as shown in Figure 1, poses serious difficulties. Companies are switching more frequently from relational databases to NoSQL databases, which are better suited to tackle these difficulties, to address the complexity and size of big data. In addition to a parallel technique, this work also introduces a hybrid migration model known as the Snapshot-Live Stream Db Migration Model. The suggested architecture and technique, which takes into account both snapshot and live data migration, provides an automated answer to the problem of document database migration.

II. Literature Review

Due to the requirement for processing large-scale and diverse data in current applications, there has been a substantial increase in interest in the data transfer from SQL (relational) databases to NoSQL databases. This research proposes a method for effective and automated data migration that combines snapshot and livestream migration approaches.

This study [1] offers a hybrid method for transferring data between SQL and NoSQL databases. The strategy combines the advantages of streaming migration with snapshots. The authors suggest a migration paradigm in which the SQL database is snapshotted at a specific moment and then subsequent updates are streamed in real-time to the NoSQL database. The outcomes of the experiment show how effective and efficient the hybrid strategy is.

An overview of various SQL to NoSQL database data migration methods is provided in this survey report. It analyses the drawbacks and advantages of various migrating strategies, such as snapshot and streaming migration. The study also emphasises the significance of taking data consistency, performance, and scalability into account during the migration process[2].

Real-time data migration from a relational database to a NoSQL database is the main topic of this research article. The real-time changes occurring in the source database are captured and replicated using the authors' suggested livestream migration technique. The paper provides a thorough design and analysis of the suggested technique, proving both its viability and performance advantages [3].

In this paper [4], a snapshot-based migration strategy for NoSQL databases is presented. It covers the difficulties of transferring massive amounts of data and offers a productive technique for taking copies of the source database and moving them to the NoSQL system. The authors assess the suggested strategy using a case study and show how well it manages data migration.

Data migration strategies fall under the area of snapshot differential database transformation [6]. In this method, changes between two separate database snapshots are identified via comparison. The updated data from the source database is converted and copied to the target database if changes are found in the second snapshot. Three approaches snapshot differential, log-based, and trigger-based—have been developed by Yong et al. [7] as part of their work on change data capturing approaches. Low data update rates are particularly suited to the trigger-based strategy. Furthermore, Wei et al. [5] described a Hadoop MapReduce technique for differential snapshots. They used concurrent programming based on Hadoop MapReduce to obtain a summary of the altered data using SQL queries and the MD5 technique.

Creating a database schema utilising graph traversing techniques like depth-first search and breadth-first search is one method suggested in [25]. A Table-like Structure (TLS) is produced as a result of this operation. Data migration is then carried out with DigiBrowser.

[26] proposes yet another clever strategy for RDBMS to big data transfer. The schema transformation and data transformation components make up this strategy. The schema transformation module converts the MySQL schema into an Oracle NoSQL schema by obtaining MySQL schema information from the JDBC driver and Java database metadata API. The Oracle NoSQL database stores the extracted table data from the MySQL database in JSON format once it has been transformed by the data transformation module.

III. Live Stream Snapshot Database Migration

A complete solution for database migration in NoSQL systems is provided by the suggested Snapshot-Live Stream Db Migration Model (SLSDMM). The migration of snapshot and live stream data in parallel from a relational database management system (RDBMS) to the document component of a NoSQL database is the main goal of this architecture.

A quick procedure or function is designed to take a snapshot of the RDBMS tables in order to start the migrating process. This snapshot records the data as it was at that precise moment. A change data capturer is simultaneously turned on to monitor and record any real-time changes made to the source database. Data is frequently updated or produced quickly in big data applications. In order to ensure that all data, including both snapshot and live stream data, is properly copied to the new NoSQL database, this is the main goal of SLSDMM. Although the transfer of the snapshot data takes some time in practical applications, any alterations (additions, deletions, or updates) made during the migration of the snapshot database are likewise synchronised and mirrored in the new NoSQL database working model shown in figure 1.

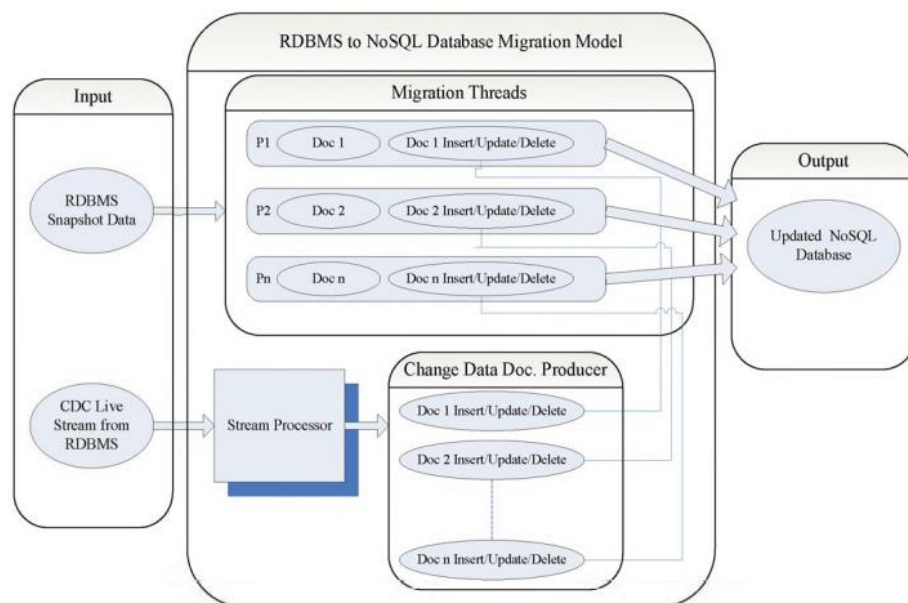


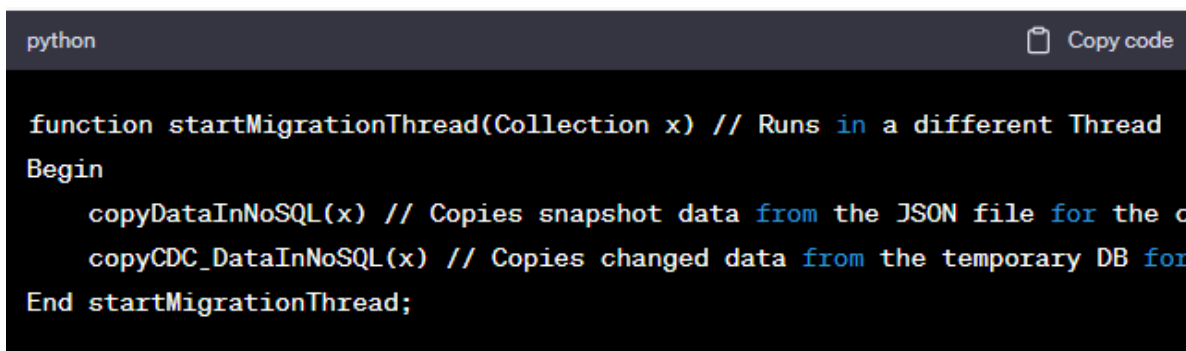
Figure 2: Live Stream Snapshot Database Migration

The concurrent migration of data from an RDBMS to a NoSQL database relies heavily on the migration thread component of the SLSDMM. This part functions as follows:

1. Pipeline Creation: Each document in the MongoDB (NoSQL) database is given its own pipeline by the migration thread. Each pipeline in the RDBMS relates to a particular table or document.
2. Migration of Snapshot Data: In parallel, the migration thread reads data from the RDBMS database's snapshot. Each document's data is processed separately, then copied into its associated NoSQL database document. This guarantees precise and effective data migration from each table.
3. Stream Processor: This component gets information from the change data capturer, which keeps an eye on the RDBMS database log for any alterations to the database. The stream processor extracts pertinent data for each individual database by grouping and filtering the altered databases that the change data capturer sends to it.
4. Transfer of Filtered Data: The Change Data Document Producer component of the model receives the filtered data that was acquired from the stream processor. This component runs in parallel and generates data records for all data that is added, updated, or removed from any NoSQL database document.

Model for Migration of Snapshot-Live Stream Database Algorithm

Using the snapshot and live stream data, the given algorithm starts the migration process from the RDBMS to the NoSQL database. "startMigrationThread" and "startCDCThread," the two key functions involved, are defined as follows:



```
python
function startMigrationThread(Collection x) // Runs in a different Thread
Begin
    copyDataInNoSQL(x) // Copies snapshot data from the JSON file for the c
    copyCDC_DataInNoSQL(x) // Copies changed data from the temporary DB for
End startMigrationThread;
```

Figure 3: Snapshot Database Migration Using Python

Each collection in the "listOfColls," which is a list of JSON files holding data for each NoSQL collection (snapshot data), is iterated over by the algorithm itself. It uses the collection as a parameter when calling the "startMigrationThread" method for each collection. This procedure moves the temporary DB's modified and snapshot data to the appropriate NoSQL collection.

The algorithm then moves on to the "startCDCThread" function, which is not explicitly stated in the supplied data, after processing all the collections. This function presumably manages the live stream of updated data and updates the NoSQL database as necessary.

IV. Result Analysis

We performed tests utilising our academic department's outcome database, which acted as the relational database, to assess the performance of the Snapshot-Live Stream Db Migration Model [10]. The department's results database is made up of a number of tables, including master tables for students, subjects, courses, and branches as well as extra tables like test marks and session management that record the results information for each course the department offers. Our suggested model's performance was compared to that of the mongify tool, the mongoimport command in serial mode, and the Sqoop HDFS map-reduce strategy. The same database migration procedure was used to compare the various approaches while gradually growing the database size. For evaluation purposes, two database sizes were taken into consideration. The second size was significantly greater, with about 11 billion records, whereas the first size was quite modest, with about 400,000 records. We wanted to evaluate the effectiveness and efficiency of our suggested methodology in handling database migrations, particularly in scenarios with huge volumes of data, thus we performed these performance comparisons. The evaluations' findings would shed light on the model's scalability and performance potential.

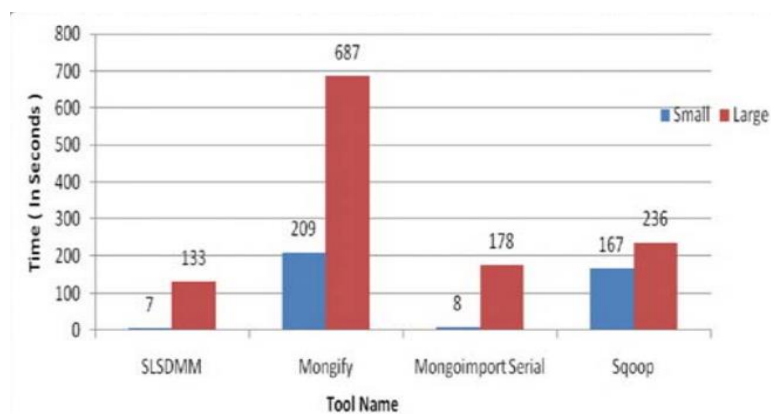


Figure 4: System performance Analysis

We found that the Snapshot-Live Stream Db Migration Model performs better than other approaches in terms of the time needed for migration after analysing the outcomes of various database migration methodologies. Our suggested solution also successfully migrates the live stream of modified data, which is a key benefit.

Furthermore, we discovered that the proposed model still performs better when big database sizes are evaluated. It demonstrates its scalability and efficiency by handling the migration procedure for databases with a significant amount of data.

These findings demonstrate the efficiency and dependability of the Snapshot-Live Stream Db Migration Model, especially when compared to competing methods. It is a good option for database migration operations because of its effectiveness in migrating both snapshot and live stream data, as well as its performance even for huge database sizes.

V. Conclusion

It is a very difficult operation to move data from an RDBMS to a NoSQL database, especially when dealing with data that is constantly changing and expanding quickly. The Snapshot-Live Stream Db Migration Model (SLSDMM), which utilises both snapshot and live stream data migration

methodologies in simultaneously, is our solution to this problem. The migration thread, the stream processor, and the altered data document producer are the three main parts of the SLSDMM model. The snapshot data from the relational database and the live stream of collected updated data are its two sources. This model effectively migrates both the live stream of modified data and the snapshot data into the NoSQL database. Experimental findings show that the suggested model performs better than current models. It successfully manages the migration of live data streams in addition to achieving speedier migration. This demonstrates the SLSDMM model's usefulness and efficiency in tackling the challenges of data transfer from RDBMS to NoSQL databases.

VI. Limitation and Future Direction

Although the snapshot-live stream migration method is a useful one for moving data from SQL to NoSQL databases, there are still some drawbacks and room for further development. These exclusions and potential future applications include:

1. **Data Consistency:** Maintaining data consistency during the migration procedure is one restriction. It can be difficult to ensure consistent data across both sources while snapshot and live stream data are transferred in tandem. The development of methods to address issues with data consistency could be the main topic of future research.
2. **Schema Transformation:** The current method presumes that the SQL and NoSQL schemas map one to the other. Complex data models, however, might call for more sophisticated schema transformations. Techniques for managing complex schema alterations during migration may be explored in further research.
3. **Rollback and error handling:** Handling problems or failures throughout the migration process is essential. Robust error handling capabilities and the capacity to roll back or recover from botched migrations could be added in the future to protect data integrity.
4. **Database Compatibility:** The present model is mostly focused on switching from SQL to NoSQL databases. It would be beneficial to expand the model to facilitate migration between various database types, such as graph databases or key-value stores.
5. **Real-time Data Synchronisation:** It would be advantageous to improve the live stream migration capabilities in order to accomplish real-time data synchronisation between the source and target databases. As a result, there would be a reduced chance of data loss or inconsistencies during the migration process.

Future Scope:

Future research can investigate clever algorithms or machine learning techniques to automatically map SQL data models to the best NoSQL schema architectures. By doing this, manual work might be reduced and data migration might be more effective. **Automated Data Consistency Checks:** It would be beneficial to develop automated methods to verify data consistency between snapshot and live stream data during migration. To find and fix any inconsistencies, this may utilise real-time

validation and verification procedures. Compatibility with Other Database Systems: Adding support for migration between various database systems, such as graph databases or time-series databases, to the scope of snapshot-live stream migration might create new opportunities for data transfer across various contexts.

References:

- [1] 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.
- [2] S. Gilbert and N. Lynch, "Brewer's Conjecture and the Feasibility of Consistent, Available, Partition-Tolerant Web Services," *Acm SigactNews*, vol. 33, no. 2, pp. 51-59, 2002.
- [3] G. Karnitis and G. Arnicans, "Migration of Relational Database to Document-Oriented Database : Structure Denormalization and Data Transformation," in *7th International Conference on Computational Intelligence, Communication Systems and Networks*, 2015, pp. 113- 118.
- [4] S. Ramzan and I. S. Bajwa, "An Intelligent Approach for Handling Complexity by Migrating from Conventional Databases to Big Data," *Symmetry (Basel)*, vol. 10, 698, 2018
- [5] W. Du and X. Zou, "Differential snapshot algorithms based on Hadoop MapReduce," in *Proceedings of the 2015 12th International Conference on Fuzzy Systems and Knowledge Discovery (FSKD)*, 2015, pp. 1203-1208.
- [6] P. Vassiliadis, "A Survey of Extract-Transform-Load A Survey of Extract - Transform - Load Technology," *Int. J. Data Warehous. Min.*, vol. 5, no. 3, pp. 1-27, 2009.
- [7] Y. Hu and W. Qu, "Efficiently Extracting Change Data from Column Oriented NoSQL Databases," *Smart Innov. Syst. Technol.*, vol. 21, pp. 587-598, 2013.
- [8] G. Karnitis and G. Arnicans, "Migration of Relational Database to Document-Oriented Database : Structure Denormalization and Data Transformation," in *7th International Conference on Computational Intelligence, Communication Systems and Networks*, 2015, pp. 113- 118.
- [9] S. Ramzan and I. S. Bajwa, "An Intelligent Approach for Handling Complexity by Migrating from Conventional Databases to Big Data," *Symmetry (Basel)*, vol. 10, 698, 2018.
- [10] B. Namdeo and U. Suman, "Performance Analysis of Schema Design approaches for migration from RDBMS to NoSQL Databases," in *2nd International Conference on Data & Information Sciences*, 2019.
- [11] V. Varga, K. T. Janosi-Rancz, and B. Kalman, "Conceptual design of document NoSQL database with formal concept analysis," *Acta Polytech. Hungarica*, vol. 13, no. 2, pp. 229-248, 2016.
- [12] C. Lee and Y. Zheng, "Automatic SQL-to-NoSQL Schema Transformation over the MySQL and HBase Databases," in *IEEE International Conference on Consumer Electronics - Taiwan*, 2015, pp. 426-427.
- [13] W. Labio and H. Garcia-Molina, "Efficient Snapshot Differential Algorithms for Data Warehousing," *Very Large Data Bases Conf.*, vol. 22, no. 1059, pp. 1-25, 1996.
- [14] Y. Hu and S. Dessloch, "Extracting deltas from column oriented NoSQL databases for different incremental applications and diverse data targets," *Data Knowl. Eng.*, vol. 93, pp. 42-59, 2014.
- [15] A. Khan, "Export to JSON from MySQL All Ready for MongoDB," 2018

- [16] "Maxwell's daemon." [Online]. Available: <https://maxwellsdaemon.io/>. [Accessed: 05-Feb-2020].
- [17] Avriela Floratou, Nikhil Teletia, David J. DeWitt, Jignesh M. Patel and Donghui Zhang, "Can the Elephants Handle the NoSQL Onslaught?", Proceedings of the VLDB Endowment, 2012.
- [18] Tilmann Rabl, Mohammad Sadoghi, Hans-Arno Jacobsen, Sergio G'omez Villamor, Victor Munt'es Mulero and Serge Mankovskii, "Solving Big Data Challenges for Enterprise Application Performance Management", The 38th International Conference on Very Large Data Bases August 27th — 31st 2012 Istanbul Turkey. Proceedings of the VLDB Endowment, 2012.
- [19] Laurie Butgereit, "Four NoSQLs in Four Fun Fortnights: Exploring NoSQLs in a Corporate in a Corporate IT Environment", SAICSIT '16 Proceedings of the Annual Conference of the South African Institute of Computer Scientists and Information Technologists, 2016.
- [20] John Klein, Ian Gorton, Neil Ernst, Patrick Donohoe, Kim Pham and Chriisjan Master, "A comparison between several NoSQL databases with comments and notes", PABS '15 Proceedings of the 1st Workshop on Performance Analysis of Big Data Systems, 2015.
- [21] Jiri Schindler, "I/O performance of NoSQL Databases(VLDB)", SIGMETRICS '13 Proceedings of the ACM; SIGMETRICS/international conference on Measurement and modelling of computer systems, 2013.