

Translation of Query for the Distributed Document Database

Muon Ha and Yulia A. Shichkina(✉)

St. Petersburg State Electrotechnical University, St. Petersburg, Russia

Abstract. This article presents the results of the automatic process of building queries to the distributed document database based on SQL queries. Queries are submitted in the form of a graph. Next, taking into account the structure of the distributed database and information about sharding and replications, a graph is modified. Based on the database elements information to which queries are referred, sets are built. By operating on these sets, the optimal structure of document databases is determined, which is further optimized by the query graph. At the end of the article, the results of testing the proposed approach to the synthetic database generated by a special program are presented. Testing showed the correctness and efficiency of the application described in the approach article.

Keywords: NoSQL · Database query · Collection · Document · Data conversion · Data format · Database structure optimization

1 Introduction

Today there are many types of NoSQL database with various features which are divided into four categories: Key-value; Document, Column (or column families) and Graph.

Since the early 90s, the relational databases are most popular. However, with the increasing volume of data, there is a growing trend towards switching relational databases to NoSQL databases. At the same time, switching from relational databases to NoSQL results to a number of problems, such as the creation of the optimal structure of the NoSQL document or database columnar family, transferring data from the relational database format to the NoSQL database format, the translation of queries, taking into account the schema of the computing system used for storage of a distributed database.

This article presents the results of research on automation of the database translating from a relational format to the format of a distributed document database, in particular MongoDB and the automatic constructing of MongoDB databases based on existing SQL queries. Formalized approaches to solve the task of designing the effective structure of a distributed document database, taking into account the specifics of data and the planned database query, do not exist. In the literature, there are many results of studies on optimizing MongoDB database and queries in separate application areas or optimization for individual parameters.

© Springer Nature Switzerland AG 2021
O. Gervasi et al. (Eds.): ICCSA 2021, LNCS 12956, pp. 396–405, 2021.
https://doi.org/10.1007/978-3-030-87010-2_29

2 Related Works

Due to the popularity of document databases over the past 10 years, various researchers have been conducted by a large number of tasks which have worked on optimizing work with databases and on developing recommendations for efficient data to work in these databases.

So, in [1], the method of translation of data from SQL to NoSQL based on "Log-Base" is given. The essence of the method is to extract the modified data from the database based on log files and copy them to NoSQL. The disadvantage of this method is that the extraction of operations from the journal is heterogeneous. This approach is possible only if the source DBMS supports the work with log files and provides API to developers.

Some ETL - tools (Extract-Transform-Load) [2] allow to transfer data from the relational database in NoSQL. For example, Mongify [3] is a new tool for translating data from relational tables to document repository. This tool helps to migrate, without worrying about primary and external keys. The disadvantage of this method is that subsequent records will not be processed until the translation process of previous records is completed. Kettle in [4] provides powerful ETL capabilities using a new metadata-based approach that allows developers to determine the method of data migration and their conversion on one node, as well as in the cloud/cluster. The Apache Sqoop tool [5] is designed to effectively move the large data between structured data warehouses, such as relational databases and NoSQL databases. Many studies [6, 7] are aimed at working a hybrid database system using SQOOP as a data converter. The problem occurs in converting a database in these systems is synchronizing database objects.

Optimization of the structure of a distributed database is a big problem for all researchers in the sciences of data. It is known that sharding and replication are popular and powerful techniques for scaling distributed systems for data. The process of organizing sharding in NoSQL was investigated in [8–11]. But global reconfiguration operations focused on the data described in these works are really ineffective. This is due to the fact that their implementation depends on special mechanisms, and not from solving the main problems of algorithmic and systemic design. Another common solution is formed through 2 steps: the first is storing the tables or the entire database, and the second is the import of all data into a new configuration [12]. This approach leads to a significant period of unavailability of the database. Another option may be the creation of a new server cluster with a new database configuration, and then filling it with data from the old cluster [13–15].

In [16], the authors describe the system called Morphus. This system provides the support for the reconfiguration of the NOSQL repository online. Morphus allows to perform reading and writing operations simultaneously with data transfer between servers. Morphus works for NOSQL storage, which includes Master-Slave replication, range partitioning and flexible data placement. In [17], the authors represent the principles and mechanisms for implementing auto-sharding in the MongoDB database. They offer an improved algorithm based on the frequency of working with data, to solve the problem of uneven distribution of data in auto-sharding. Improved balancing strategy can effectively balance data between segments and improve the performance of the simultaneous reading and writing cluster. In [18], the authors solve the problem of choosing a key

for sharding in MongoDB. The authors modeled the environment and measured reading/writing performance of various keys and found that shard key with random and good locality can give decent performance during writing and reading.

The analysis shows that at present, the problems of translation data and queries of the database from one format to another are very relevant. In this case, no good solution to these issues was found. There are private solutions for individual areas, for direct translation of tables in the collection, to transfer relational database to document databases, taking into account link types, to extract data from document databases using SQL queries, query optimization methods.

This article proposes a formalized approach, which takes into account the schemes of relational database, schemes of SQL-query and schemes of sharding and replication to build a distributed document database and automatically construct query to the distributed MongoDB database.

3 Application of an Information Graph to Build a Distributed Database

In the series works [21, 22], we have given two methods for optimizing the structure of a document database without and with Embedded Documents. These methods are in the solutions for the problem of consolidating databases of various types with a translation database from one to another. And we also developed a method and tool to translate queries from MySQL in MongoDB [20]. But these methods are only for centralized databases. The purpose of the current study is to develop an approach to translate queries to a distributed document database.

In our study, an approach based on information graphs was applied to study the dependencies of subqueries from the location and availability of data in accordance with the specified scheme of the distributed database and information about shards and replicas.

Based on the information graph, we will further mean an oriented acyclic multigraph, the vertices of which correspond to the sub-queries within the main query, and the arc - data transmission between them.

In the method of constructing an information graph of query for a distributed relational database, taking into account the information about the fragmentation of the tables, three types of links between the query elements are considered. We illustrate these three options in the examples.

Example. Suppose that the T1 tables are performed consistently queries Q1 and Q2. Both of T1 and T2 tables participate in Q1 query, two tables are involved in the Q2 query: a table with Q1 query results and T3 table. The source graph of query is shown in Fig. 1a. It is possible to occur the following options for performing queries, taking into account two fragments of T1 table.

Option 1. Q1 query is performed for each fragment of T1 table and the entire T2 table. The results are aggregated on each of the shards after the query Q1 is executed. The Q2 is then satisfied to T3 and the above-mentioned table with the results of aggregated data after executing the Q1 query over the T1 table fragments and T2 table. This graph is illustrated in Fig. 1b.

Option 2. Q1 query is divided into two parts. First, it is performed for each fragment of T1 table and does not affect T2 table. After the first part of the query Q1, the results are carried out on each of the shards. The second part of the query Q1 is then running to T2 table and the results of aggregated data after the first part of the Q1 query over the fragments of T1 tables. After that, the Q2 query is satisfied with T3 table and the table with the results of the Q1 query. This graph is illustrated in Fig. 1c.

Option 3. Q1 query is divided into two parts. First, it is performed for each fragment of T1 table and does not affect T2 table. After executing the first part of the Q1 query, the aggregation of the results on each of the shards is not carried out. The second part of the Q1 query is immediately performed to T2 table and the data results after the first part of the Q1 query over the T1 table fragments. The Q2 query is also divided into two parts. First, the first part of the query Q2 is performed to T3 table and the table with the results of the Q1 query. After the first part of the query Q2, the results are aggregated on one of the shards. After execution, an aggregation of results on each of the shards is carried out. This graph is illustrated in Fig. 2c. The Q2 is then executed to T3 table and the table with the results of aggregated data after the first part of the Q2 query is performed. This graph is illustrated in Fig. 1d.

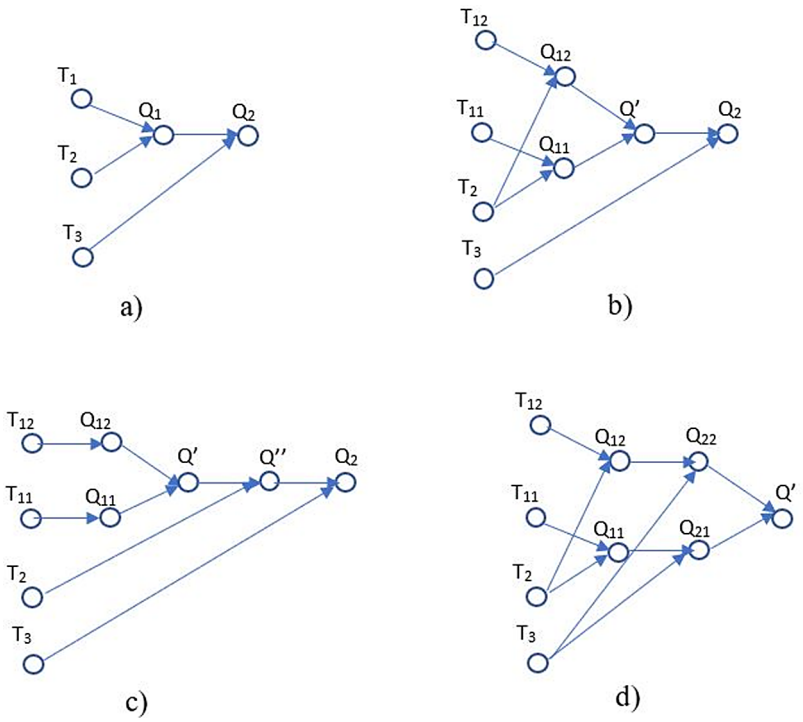


Fig. 1. Ways to represent a query to a distributed database using an information graph

4 Method for Constructing a Document Database Schema, Taking into Account the Structure of a Distributed Relational Database

1. Build an information graph without scarring and replication.
2. Create an information graph to a parallel form that minimizes data transmission from shard to shard using the method of optimizing parallel algorithms by the number of communications [19].
3. Perform an action 1–2 for all requests whose structure must be taken into account during optimizing the document database schema in order to accelerate the execution of these queries.
4. Based on the metadata on tables, fields, requests and links using the methods of optimizing a document database schema with nested or without nested documents, form the structure of the collections of documents.
5. Improve the structure of documents taking into account the graphs of information dependences of requests, minimizing the number of data combining operations.

5 Translation of Query from MySQL into a Distributed MongoDB Database

This section provides the results of the automatic constructing query to the MongoDB database. For testing, three options for the database structure were selected: not a distributed database, distributed database with shards and a distributed database with shards and replicas. The method described in [20] was selected for automatically constructing query. Figure 2. presents the MySQL database schema:

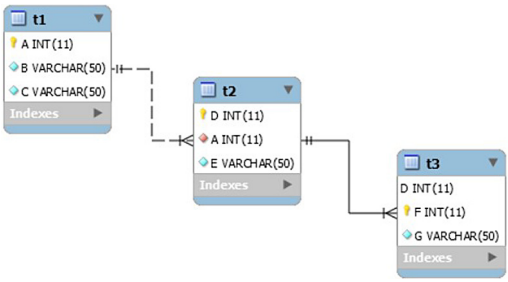


Fig. 2. MySQL database diagram

Select SQL_NO_CACHE count(*) From t1 where A in (Select A from t2 where D in (Select D from t3 where F%5=0));

As a result of the application of the database translating method described in [21], a database structure on MongoDB will be obtained, shown in Fig. 3:

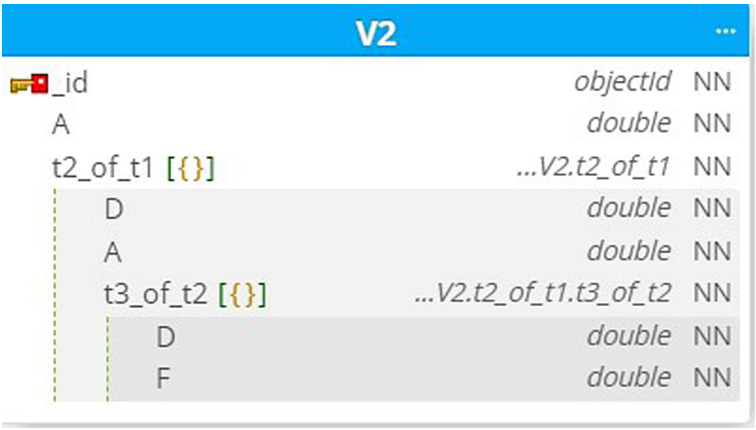


Fig. 3. MongoDB database collection structure

For the case of a non-distributed database using automatic translation query, a MongoDB format will be received, presented in Table 1.

Table 1. Source SQL query and automatically constructed MongoDB query.

MySQL	MongoDB
`Select SQL_NO_CACHE count(*) From t1 where A in (Select A from t2 where D in (Select D from t3 where F%5=0));`	`db.V2.aggregate([ {"$unwind": "$t2_of_t1"}, {"$unwind": "$t2_of_t1.t3_of_t2"}, { "$match": {"t2_of_t1.t3_of_t2.F": {"$mod": [5, 0] }} }, { "$project": { "A": "$A","_id":0} }, { "$group": {"_id":"$A"} }, { "$count" :"Count" } ])`

The proposed query execution plan for the case of a distributed database without replication is shown in Fig. 4.

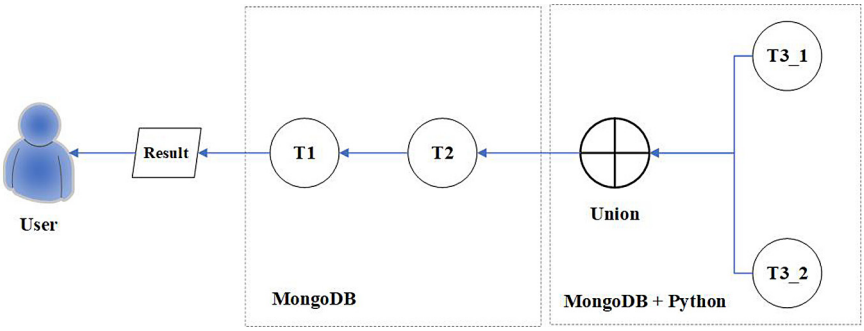


Fig. 4. The proposed plan for performing a query for a distributed database without replications

A query to a distributed database without replications, but with sharding was divided into subqueries that were parallel to shards. Since MongoDB does not support the parallel execution of subqueries, the method of translation query was applied to individual subqueries. To perform a general query, programs written in the Python programming language.

The proposed query execution plan for the case of a distributed database with replications is shown in Fig. 5.

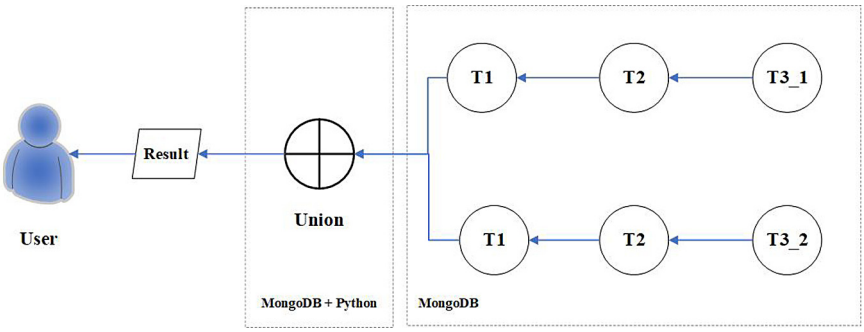


Fig. 5. The proposed plan for performing a query for a distributed database with replications

In contrast to the previous case, the query was divided into two subqueries, each of them chose all records in accordance with some conditions. The calculation of the number of records was made directly in the program written in the Python programming language.

6 Query Testing

Another test task was to determine which option parallel execution of the query time will be the smallest. To carry out this testing, synthetic databases with different volumes are generated, which are shown in Table 2.

Table 2. Databases for testing

	Option 1		Option 2		Option 3	
	Node 1	Node 2	Node 1	Node 2	Node 1	Node 2
Volume 1	1000	0	1000	849	997	849
Volume 2	10000	0	10000	8682	10000	8682
Volume 3	10000	0	10000	87461	10000	87461
Volume 4	1000000	0	1000000	874525	1000000	874525

Table 3. The execution time of the query.

	Option 1	Option 2	Option 3
Volume 1	0.0464	0.0189	0.0109
Volume 2	0.1423	0.112	0.0708
Volume 3	1.297	1.089	0.6801
Volume 4	13.445	11.531	6.884

Table 3 shows the execution time (in seconds) using the developed program:

The schedules of the query execution time for different variants of the database and data volume scheme are shown in Fig. 6

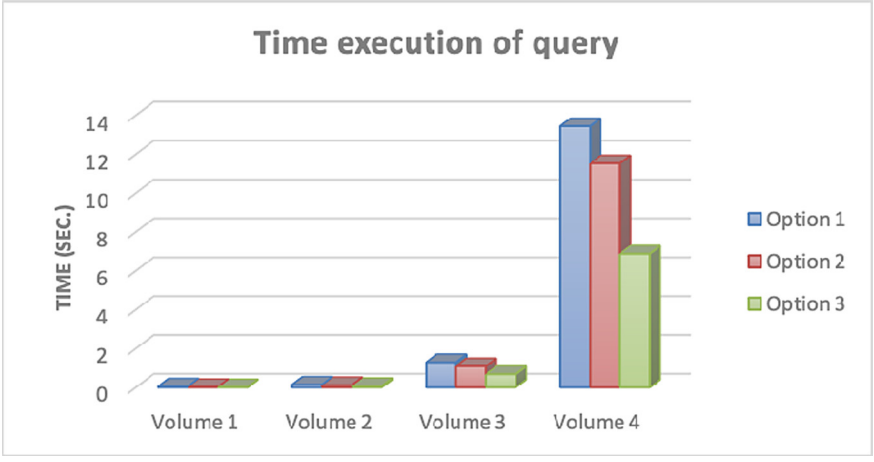


Fig. 6. The execution time of the query for the different volumes of databases

From the diagram in Fig. 6. It can be seen that the maximum parallelization of the database query with the maximum deferred operation of the program's result is faster.

7 Conclusion

Automatic translation of query from MySQL in MongoDB is a challenge, especially in the case of distributed data. In this article, we presented a method of automatically constructing queries to the MongoDB distributed database based on the MySQL query metadata of the distributed database architecture as well as using methods for optimizing the MongoDB database collections. The automatic query construct is based on the method which based on information graphs.

We have shown that various options for building an information graph of a query, taking into account the specifics of the query operations and the architecture of a distributed database.

Our test results showed the correctness and efficiency of the application to automatically build query to the distributed MongoDB database.

To build a more efficient query for the time database or other parameters, it is possible to use the query to the information graph of the pre-optimization method for optimizing parallel algorithms for various parameters based on information graphs.

In the future, we plan to conduct research on the classification of requests for the type of operations and the variants of their parallelization.

References

1. Shahra, G., Yap, J.: Cache augmented database management systems. In: Proceedings of the ACM SIGMOD Workshop on Databases and Social Networks, pp. 31–36. ACM, New York (2013)
2. Vassiliadis, P.: A survey of extract–transform–load technology. Int. J. Data Warehouse. Min. **5**, 1–27 (2009). https://doi.org/10.4018/jdwm.2009070101
3. Consulting, A.: Mongify–move data from sql to MongoDB with ease. In: http://mongify.com/. Accessed 26 Mar 2021
4. Casters, M., Bouman, R., Dongen, J.: Pentaho Kettle solutions. Wiley, Indianapolis, Indiana
5. Ting, K., Cecho, J.: Apache Sqoop Cookbook. O'Reilly Media, Sebastopol (2013)
6. Joldzic, O., Vukovic, D.: The impact of cluster characteristics on HiveQL query optimization. In: 2013 21st Telecommunications Forum Telfor (TELFOR) (2013). https://doi.org/10.1109/telfor.2013.6716360
7. Kim, T., Chung, H., Choi, W., et al.: Cost-based join processing scheme in a hybrid RDBMS and hive system. In: 2014 International Conference on Big Data and Smart Computing (BIGCOMP) (2014). https://doi.org/10.1109/bigcomp.2014.6741428
8. How to change the Shard Key (2021). http://stackoverflow.com/questions/6622635/how-to-change-the-shard-key. Accessed 26 Mar 2021
9. Mongo DB (2021). http://www.MongoDB.org. Accessed 26 Mar 2021
10. Chang, F., Dean, J., Ghemawat, S., et al.: Bigtable: a distributed storage system for structured data. ACM Trans. Comput. Syst. **26**, 1–26 (2008). https://doi.org/10.1145/1365815.1365816
11. Lakshman, A., Malik, P.: Cassandra. ACM SIGOPS Operating Syst. Rev. **44**, 35–40 (2010). https://doi.org/10.1145/1773912.1773922
12. Change shard key MongoDB faq (2021). http://docs.MongoDB.org/manual/faq/sharding/#can-i-change-the-shard-key-after-sharding-a-collection. Accessed 29 Mar 2021
13. Barker, S.K., et al.: Shuttledb: database-aware elasticity in the cloud. In: 11th International Conference on Autonomic Computing, ICAC 2014, Philadelphia, PA, USA, 18–20 June, pp. 33–43 (2014)

14. Das, S., Nishimura, S., Agrawal, D., El Abbadi, A.: Albatross: lightweight elasticity in shared storage databases for the cloud using live data migration. Proc. VLDB Endowment **4**, 494–505 (2011). https://doi.org/10.14778/2002974.2002977
15. Elmore, A., Das, S., Agrawal, D., El Abbadi, A.: Zephyr: live migration in shared nothing databases for elastic cloud platforms. In: Proceedings of the 2011 International Conference on Management of Data-SIGMOD 2011 (2011). https://doi.org/10.1145/1989323.1989356
16. Ghosh, M., Wang, W., Holla, G., Gupta, I.: Morphus: supporting online reconfigurations in sharded NoSQL systems. In: 2015 IEEE International Conference on Autonomic Computing (2015). https://doi.org/10.1109/icac.2015.42
17. Liu, Y., Wang, Y., Jin, Y.: Research on the improvement of MongoDB auto-sharding in cloud environment. In: 2012 7th International Conference on Computer Science and Education (ICCSE) (2012). https://doi.org/10.1109/iccse.2012.6295203
18. Kookarinrat, P., Temtanapat, Y.: Analysis of range-based key properties for sharded cluster of MongoDB. In: 2015 2nd International Conference on Information Science and Security (ICISS) (2015). https://doi.org/10.1109/icissec.2015.7370983
19. Shichkina, Y., Kupriyanov, M., Al-Mardi, M.: Optimization algorithm for an information graph for an amount of communications. Lecture Notes in Computer Science, pp. 50-62 (2016). https://doi.org/10.1007/978-3-319-46301-8_5
20. Ha, M., Shichkina, Y.: The query translation from MySQL to MongoDB taking into account the structure of the database. In: 2021 IEEE Conference of Russian Young Researchers in Electrical and Electronic Engineering (EIConRus). St. Petersburg and Moscow, Russia, pp. 383–386 (2021). https://doi.org/10.1109/ElConRus51938.2021.939659
21. Shichkina, Y., Ha, M.: Creating collections with embedded documents for document databases taking into account the queries. Computation **8**(2), 45 (2020). https://doi.org/10.3390/computation8020045
22. Ha, V.M., Shichkina, Y.A., Kostichev, S.V.: Determining the composition of collections for key-document databases based on a given set of object properties and database querie. Comput. Tools Educ. (3), 15–28 (2019). https://doi.org/10.32603/2071-2340-2019-3-15-28