2014 Ninth International Conference on P2P, Parallel, Grid, Cloud and Internet Computing

Schema Conversion Model of SQL Database to NoSQL

Gansen Zhao[1], Qiaoying Lin[1,2], Libo Li[1,2], Zijing Li[1,2]

[1]*School of Computer Science, South China Normal University, Guangzhou, China*

[2]*Guangzhou Key Lab on Cloud Security and Assessment, Guangzhou, China*

gzhao@scnu.edu.cn

Abstract

With the increasing maturity of NoSQL databases as well as the situation of reading more than writing on large volumes of data, many applications turn to NoSQL and pick it as data storage system. Migrating from SQL database to NoSQL and providing efficient query become growing imperative. However, join operation is not supported in NoSQL database and read separately for multiple times which brings poor performance is unavoidable. This paper proposes a schema conversion model for transforming SQL database to NoSQL which can provide high performance of join query with nesting relevant tables, and a graph transforming algorithm for containing all required content of join query in a table by offering correctly nested sequence. A rigorous proof about the transform algorithm has been conducted and the experiment has verified the correctness of the conversion model and the high performance of join operation.

Keywords: Database Migration, Schema Conversion, Denormalization, NoSQL Database

1. Introduction

With the explosive growth in the size of dataset, big data processing has attracted more and more attention. Generally, relational database is one of the most popular ways to manage data in practical scene. However, faced with a substantial increase in the amount of data, relational database is suffering severe stress. Many researches of denormalization have been done in order to design relational database schemas that can optimize the read performance and ensure the consistency when altering data at the same time. Besides, with the demand of effectively storing and managing different types and structures of data, and high reliability and scalability, NoSQL database has been developed to cover the shortage of relational database [1, 2, 3]. NoSQL database provides a mechanism for storage and retrieval of data that is modeled as unstructured other than the tabular relation used in relational databases. It is suitable for read-heavy OLAP systems where short data inconsistency is tolerable such as big data scenarios and web applications. Furthermore, with Hadoop or other distributed computing framework, distributed data mining and analysis is available to all kinds of software innovators and entrepreneurs, including but not limited to big guns like Google and Yahoo!. But there is a problem that the data to analyze is derived from relational database. So there is great demand for efficient and reliable solution to migrate data from relational database to NoSQL database.

Many companies turn to NoSQL database and use it for storing and managing data, and relational database of existing applications must be transformed to NoSQL database. However, data schemas are completely different between these two kinds of databases, which means steep learning curve for users. In addition, join operation is not supported in NoSQL database and read separately for multiple times is needed which brings poor performance. As a result, schema conversion is important for replacing relational database with NoSQL database, and so as importing data from relational schema to NoSQL. In addition, it is very significant to ensure high reading efficiency after converting schema. Though there are various relational databases such as Oracle, SQL Server, MySQL, all of them share same relational schema. However, every NoSQL database has its own data schema [4]. There are four basic categories of NoSQL databases: key-value, document, columnar, and graph database. The graph database is meant for a specific usage, i.e. highly inter connected data in social networking applications, this is why it is left out of scope of this work. And what we lay emphasis are the other three kinds of databases. In this paper, we will introduce a system that we had implemented to verify our solution.

We propose a general schema conversion model for converting relational database to NoSQL database which help migrating to NoSQL and improves reading efficiency. We design the schema conversion procedure by applying the idea of table nesting to improve the performance of cross table query. To store structured data with references in NoSQL database, we consider the references as relationship between parent layer and children layer of semi-structured data in NoSQL database. The idea of nesting is that storing referred tables as children tags in data item of the referring table. In addition, semi-structured data with multiple layers is able to describe structured data with multiple references between tables, which covers all situations of table references. This idea of table nesting ensures that there must be a one-to-one mapping between the table in relational database and dataset in NoSQL. Take MongoDB [5] for example, if there are n tables in relational database, there must be corresponding n collections in MongoDB.

Additionally, we propose a graph transforming algorithm to generate nesting sequence among relational tables to ensure the correctness of schema conversion for all contents needed by a relevant query statement are contained in the table. Considering a table as a vertex, reference between tables as an edge, construct a graph based on tables and their relationship in a relational database. The sequence of eliminating edge based on the transforming algorithm is corresponding to the nested sequence. After the schema conversion, accessing once on one table is needed for any query statement, which enables effectively reading on NoSQL database. Even though more space is needed when using this conversion model, it is workable with lower storage cost.

The rest of the paper is organized as follows. Section 2 introduces the related work about conversion from relational database to NoSQL database. Section 3 expatiates the detail design of schema conversion model we proposed and the graph transforming algorithm. In section 4, we present the conversion experiment between MySQL and MongoDB based on our model. Finally, section 5 concludes the paper with some directions and advances the further work.

2. Related Work

For obtaining high reading capability and magnificent availability and scalability of relational database, some related researches have been done, which mainly focus on the following orientations: denormalization, migrating data to NoSQL database, and converting schema to NoSQL database. Denormalization as a process seeks to improve the response time for data retrieval while maintaining good system performance for row insertions, updates, and deletions [6]. It can enhance query performance when it is deployed with a complete understanding of application requirements [7]. Data migration aims to transferring bulk data from structured data stores such as relational database to distributed database, and acquires schema-free, easy replication support and highly scalability. Schema conversion provides an approach towards transferring database's structure in addition to migrating data.

2.1 Denormalization

YMA PINTO [8] proposes a framework for systematic database denormalization in an effort to improve data retrieval performance and reduce response time without losing data integrity. The framework is consists of denormalization view design, denormalization view maintenance and denormalization view exploitation. Denormalization view design determines data storage and access. Denormalization view exploitation speeds up query processing by denormalizing pre-joined tables, report tables, fragmenting tables, redundant data, repeating groups, derivable data and hierarchical speed tables while denormalization view maintenance synchronizes denormalized schema and base tables.

In [9], the authors propose restructure data of web application into independent data services, and gain magnitude throughput improvement compared to conventional master-slave replication approach by scaling individual data services. When denormalizing an application's data, transaction and read query must be taken into consideration. With regard to transaction, mark real database transactions require ACID properties, atomic set and UDI queries that are not part of a transaction with the data fields and keep them in a single service. As for read query, rewrites certain read query into a collection of sub-queries, where each sub-query is capable of accessing only one service, and replicates data from one table to others, which enables transforming join query into simpler query.

2.2 Migrating Data

Many tools have been developed to migrate data from relational database to NoSQL database. Apache Sqoop [10] and DataX [11] are two popular ones.

Apache Sqoop is a tool designed for efficiently transferring bulk data between Apache Hadoop and structured data storage such as relational database. It can import data from relational database into HDFS, transform the data in Hadoop MapReduce, and then export the data back into a relational database. Although Sqoop can complete the job to import data to Hadoop/Hive/HBase from relational database, it is not able to import the dependencies of tables in relational database. As a result, if users need to migrate data from relational database to Hadoop/Hive/HBase, and query on HBase after migration, they must access more than one table in HBase which costs a lot of time.

DataX is a tool designed to exchange data between heterogeneous data source, e.g. HDFS, MySQL, Oracle, at very high speed. DataX provides a framework to schedule process, exchange data, which is called DataX engine. All data processing, like data extraction, data loading, are handled by DataX plugins which are plugined into DataX engine. The reader plugins are responsible for extracting data from data source, as the writer plugins are for loading data to data destination. But just like Sqoop, DataX also provides no solution for importing relationship of tables in relational database to NoSQL.

2.3 Converting Schema

JackHare [12] is a framework for SQL to NoSQL translation using MapReduce. It gives us an approach to exploit HBase as the underlying data store to execute ANSI-SQL queries and relieve the learning curve of analyzing big dataset in NoSQL database rather than relational database. One steps of its solution is to transform data from relational database to HBase. It provides a concrete conversion model which stores all tables of a relational database in a single HBase table. In the conversion, data of an SQL table are migrated into a column family and column schema of the relational table is remapped into qualifier of that column family. Additionally, a special column family is adopted for foreign keys of relational database. However, query involving many foreign keys will not have high performance among such data model for great amount of join operations to process.

Transforming Relational Database into HBase：A Case Study [13] presents three guidelines to transform relational schema into HBase schema based on the data model of HBase, and expresses the relationships between their schemas as a set of nested schema mappings to transform relational data into HBase representation automatically. The three guidelines include grouping correlated data in a column family, adding foreign key references if one side needs to access the other side's data, and merging attached data tables to reduce foreign keys. With the three transformation rules, schema of relational database can be transformed into HBase schema by generating schema mappings. However, multiple level nested is not considered in the paper. Besides, no concrete solution of database transformation is pointed out.

To sum up, the previous works mainly focus on improving reading performance or migrating data to NoSQL database to receive high availability and scalability, but do not provide an effective solution to eliminate reference dependencies among tables in relational database or transform the relationship between tables to NoSQL database. Query must span multi-tables to obtain data which originally need to join more than one table in relational database, resulting in low query efficiency.

3. Design of Schema Conversion

This chapter presents the design of schema conversion model which converts schema from SQL database to NoSQL database. The design of schema conversion is based on table nesting process which is considered as an abstract process in this design. First of all, we provide a graph model to describe general database schema, including relational schema and NoSQL schema. Through this graph model, we present the procedure of schema conversion, a sequence of extension operations that converts original schema into final schema.

3.1 Graph Model of Database Schema

Data in relational database model is organized by table, and relations between tables are represented by foreign keys. NoSQL database, however, does not like SQL database, it organizes data by numbers of dataset, and they are independent from each other. Facing such difference between SQL database and NoSQL database, we put forward a model which describes both SQL database and NoSQL database at the same time.

Basic Definition:

Table Dependency: table dependency means one table references another by foreign key.

Depth of Table Dependency: the number of edges among two tables in DAG.

Table Extension: extend a table means adding data or extra information into the table.

Graph Definition:

(Note that the database mentioned below includes SQL database and NoSQL database.)

DB: a database.

t_i: a table in the database.

T: the set of tables in the database

V_i: a vertex in the graph, corresponding to a table in the database.

$V = \{V_1, V_2, \cdots, V_k | 1 \le k \le n\}$: the vertex set in the graph, representing all tables in the database.

$e = \langle V_i, V_j \rangle$: a directed edge represents dependency from table t_i to table t_j.

$E = \{\langle V_i, V_j \rangle | 1 \le i, j \le n, i \ne j\}$: the edge set in the graph, representing all relations of dependency in the database.

$G = \langle V, E \rangle, |V| \ge 1$: DAG (Directed Acyclic Graph), a relational graph representing a database.

Definition of database using graph model:

$G = f(DB)$, for $\forall t \in DB.T$, there is $V_t \in G.V$ and $e_t = \langle V_t, V_j \rangle \in G.E$. Moreover, there is no other V_i and e belongs to G.

Particularly, there is no relation of dependency in NoSQL database. So, when using graph model to describe NoSQL database, there is $E = \emptyset$.

A sample of the graph model describing SQL database is shown below. In Figure 3-1, T represents table names, A represents attributes, FK represents foreign keys (table dependency). t1, t2, t3 and t4 represent four tables in this database, $< t1, t2 >, < t2, t3 >, < t3, t4 >$ represent three relations of dependency.

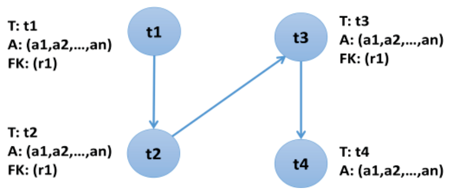

Fig. 3-1 Graph Model of Database

3.2 Conversion Procedure

a) Simple Extension

The definition of simple extension is the procedure of transformation from relational table to NoSQL, while this relational table references only one or none relational. There is an example in Figure 3-2.

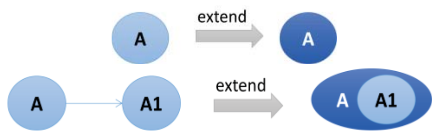

Fig. 3-2 Simple Extension

b) Vertical Extension

Define n as the depth of the relation of dependency.

① When n = 1, make a simple table extension. For example, in Figure 3-3, table B references table B1 (B->B1). After the conversion, B will contain the data of B1.

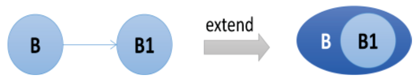

Fig. 3-3 Vertical Extension (n = 1)

② Assume n = k, make simple extension one after another. For example, in Figure 3-4, table B references table B1, B1 references table B2 and so on. After the conversion, table B1 will turn into B1* (B1→B2→ •••→ Bn) which contains the information from B2 to Bn. And B→B1* is similar to ①.

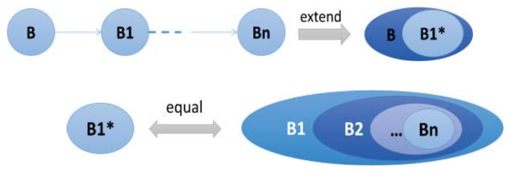

Fig. 3-4 Vertical Extension (n = k)

③ When n = k + 1, make simple extension between (k+1)th table and the result which got by ②. For example, in Figure 3-5, table A references table B, table B references table B1 and so on. After the conversion, table B will turn into B* (B→B1→...→Bn) which contains the information from B1 to Bn. And A->B* is similar to ①.

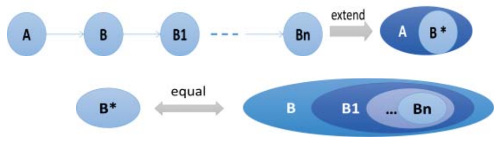

Fig. 3-5 Vertical Extension (n = k+1)

c) Horizontal Extension

Define n as the numbers of a table directly references other tables.

① When n = 1, make a simple extension.

② Assume n = k, make simple extension one after another. For example, in Figure 3-6, table A references table C1, C2, •••, Cn at the same time. After the conversion, similar to ①, table C1 is extended into table A, table C2 is extended into table A and so on. As the result, the set C^+ = {C1,C2,...Cn} will be extended into table A.

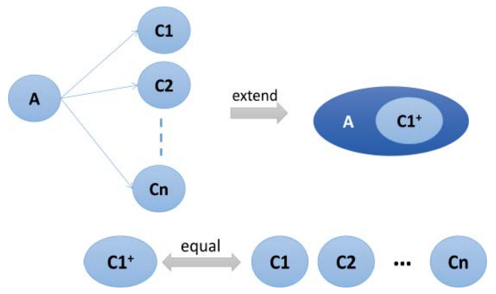

Fig. 3-6 Horizontal Extension (n = k)

③ When n = k + 1, make simple extension between (k+1)th table and the result which got by ②. For example, in Figure 3-7, table C is (k+1)th table, table A references table C. After the conversion, table C is extended into table A by simple extension. And C^+ = {C,C1,C2...Cn} at this moment.

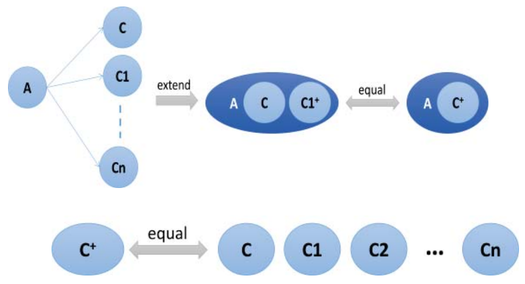

Fig. 3-7 Horizontal Extension (n = k +1)

3.3 Algorithm of Graph Transforming

A point with child means it possess incomplete information. In order to gather missing information, this dataset needs to integrate data from its children point. We must also make sure that the child has already possessed complete information as we integrate from that child point, or it might cause some data loss.

In DAG, we call a vertex as a leaf node when it's out degree is zero. A table corresponding to a leaf node does not reference to any other table in the graph. We first convert all leaf nodes to NoSQL database according to conversion(a) rule, and remove the edges that reference to them. Recursively process the graph until all nodes are processed.

This algorithm solves the general problem of transforming data from a relational database (probably some tables and their foreign keys) to NoSQL database without data loss. It outputs a sequence of foreign keys, which shows the sequence of data integration from child point to parent point.

While we saying data integration from child to parent, we do not consider specific data or specific relationship between them. Neither do we care about how data integration from child to parent performs. We assume that there is no data loss among such data integration.

Definition:

a) Completed dataset point: A point u becomes completed dataset point, if and only if u does not have any child, or every child of u as v has already become completed dataset point before data in v integrates into u.

b) Set f(u) equals to the point set of direct predecessor of u.

$$f\,(u) = \{ v \mid v \in V \wedge < v , u > \in E \}$$

Set F(u) equals to the point set of predecessor of u.

$$F(u) \;=\; f^*(u) \;=\; \{ v \mid v \;\in\; f(u) \vee (w \in\; f^*(u) \;\wedge < v , w > \in E) \}$$

Set b(u) equals to the point set of direct successor of u.

$$b\,(u) = \{ v \mid v \;\in\; V \;\wedge < u , v > \in E \}$$

Set B(u) equals to the point set of direct successor of u.

$$B(u) \;= b^*(u) \;=\; \{ v \mid v \;\in\; b(u) \;\vee\; (w \in\; b^*(u) \;\wedge < w , v > \in E) \}$$

Declaration:

Array O[v] means the out degree of point v.

Point set Q is the set of points who's out degree is not 0.

Point set P is the set of points who's out degree is 0 and in degree is not 0.

Point set T is the set of points who's out degree and in degree are 0.

Sequence S is the foreign key sequence.

Algorithm:

Input: *Graph G = <V, E>*

Output: Sequence of foreign key S.

```
    $O[v] \leftarrow |b(v)|$
    $P \leftarrow \{ v \mid v \in V \wedge O[v] = 0 \};$
    $Q \leftarrow V - P;$
    $T \leftarrow \emptyset;$
    $S \leftarrow \{\};$
while $P != \emptyset$ do
    $u \leftarrow x \in P;$
    $P \leftarrow P - \{u\};$
    $T \leftarrow T \cup \{u\};$
    for v in f(u) do
        $S \leftarrow S.add(<v, u>);$
        $O[v] \leftarrow O[v] - 1;$
        if $O[v] = 0$ do
            $P \leftarrow P \cup \{v\};$
            $Q \leftarrow Q - \{v\}$
        end if
    end for
end while
output S
```

A sample process of this algorithm are shown below as Figure 3-7。In the beginning, there are three points and three edges as it is shown in step (1). Firstly, point C is nested into point A, as step (2) has shown. Then point C is also nested into point B as step (3). Finally point B is nested into point A, as step (4).

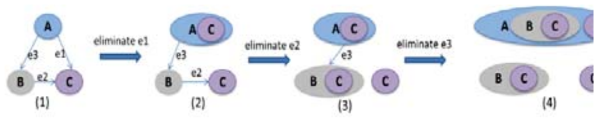

Fig. 3-8 Sample Process of the Algorithm

4. Experiment

In order to verify the correctness of our solution for schema conversion from relational databases to NoSQL databases, we implement a system to do the migration based on MySQL and MongoDB. Some representative SQL query statementsare used to demonstrate that we will definitely get the same result in original MySQL database and later MongoDB. In addition, because our solution lays emphisis on high query efficiency on transformed NoSQL databases, we design an experiment to compare the query efficiency between our Nested soulution and Non-Nested solution. The experiment shows that our solution gain marked improvement in query efficiency with the increasing of joining table in one query statement, comparing with Non-nested solution. While mentioning the problem of space redundancy, we figure out the size of data volumn in original relationnal database and MongoDB after migration.Moreover, the performance of query using Nested solution and Non-Nested solution will be given in this section. At last, we discuss the space redundancy of our schema conversion solution, and experiment data is also shown in this section.

4.1 Experiment Data

Our experimental data is derived from a business management system. It contains seventy-two tables. And there are about thirteen attributes and three foreign keys in each table. We conduct a comprehensive test in order to ensure that the correctness of the conversion. A case which takes four SQL query statements as input is designed to get the experiment results. This case involves four tables and three foreign keys. The relational schema of the case is showed in Figure 5-1, where T represents table's name, A represents attribute set, and FK represents foreign key set. In this case, the SQL query statement includes four tables' join connection. For example, if attributes in query criteria belong to t4 (*tcategory*) while some of the attributes need to be returned belong to t1 (*tagreementforfee*), we need to join four tables including t1, t2, t3, and t4 to get the result.

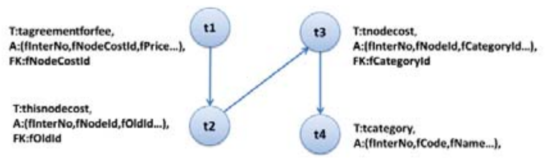
T:tagreementforfee,
A:(fInterNo,fNodeCostId,fPrice...),
FK:fNodeCostId
t1
t3
T:tnodecost,
A:(fInterNo,fNodeId,fCategoryId...),
FK:fCategoryId
T:thisnodecost,
A:(fInterNo,fNodeId,fOldId...),
FK:fOldId
t2
t4
T:tcategory,
A:(fInterNo,fCode,fName...),

Fig. 5-1 Relational Database Schema

4.2 Experiment Result

Based on the relational schema shown in Figure 5-1, this section verifies the correctness of our conversion model by using four representative SQL query statements which are showed in Figure 5-1. In Table 1, No.1 refers to a query statement on single table. No.2, No.3 and No.4 refer to multi-tables join queries. At last, we discuss the space redundancy of our schema conversion solution, and experiment data also is shown in this section.

Table 1 SQL Query Statement

No.	SQL Query Statement

1	SELECT fInterNo FROM tagreementpriceforfee
	WHEREf Price='0.0101';

2	SELECT * FROMtagreementpriceforfee,thisnodecost
	WHERE tagreementpriceforfee.fNodeCostId=thisnodecost.fInterNo
	ANDf Number=36;

3	SELECT tagreementpriceforfee.fPrice
	FROM tagreementpriceforfee, thisnodecost, tnodecost
	WHERE tagreementpriceforfee.fNodeCostId=thisnodecost.fInterNo
	AND thisnodecost.fOldId=tnodecost.fInterNo
	AND tnodecost.fNumber=38；

4	SELECT tagreementpriceforfee.fPrice FROM
	tagreementpriceforfee,thisnodecost,tnodecost,tcategory
	WHERE tagreementpriceforfee.fNodeCostId=thisnodecost.fInterNo
	AND thisnodecost.fOldId=tnodecost.fInterNo
	AND tnodecost.fCategoryId=tcategory.fInterNo AND
	tcategory.fPathCode='132306';

4.2.1Correctness

The correctness can be guaranteed by the fact that we can get the same result from MySQL database and MongoDB by using the same SQL query statement. As for No.1 SQL query statement in Table 1, there are 7 records with attribute "fPrice" got from MySql, while there are the same resultgot from MongoDB. Theexperiment for the rest SQL query statement also show the same consistency.

4.2.2 Performance

This paper gives the comparison of query performance between Nested solution and Non-Nest solution. As we all known, if the SQL query statement involves multi-table join, the query need to access more than one table in original relational database. Similarly, if the documents in MongoDB do not use the nested design also face the same problem. So the non-nested solution will spend more time on getting result from MongoDB than nested solution when dealing with the multi-tables join query.But in our nested solution, we can achieve the same query by accessing just a single collection of MongoDB. Increment of join-keys will not cause more times of access collection in MongoDB. The query times are showed in Figure 5-2.

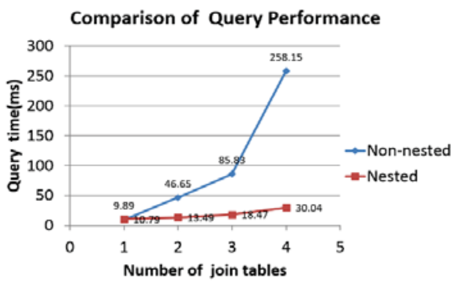

Fig. 5-2 Query Performance of Nested MongoDB and Non-nested MongoDB

As we have seen, with the increment of joining tables in SQL query statement, the time needed for our Nested solution did not change significantly, but the time for Non-Nested case grows more rapidly. Because when we adopt Non-Nested solution, loop for matching the join keys after scan with filtering will result in a great amount of time cost.

However, such solution improves query performance observably by sacrificing space redundancy. Space redundancy after the migration is related to the products of the number of records of nesting table and average size of each record of nested table. In the case of Section 5.1, R(ti) represents the number of records of table ti, S(ti) represents the average size of each record in table ti. The Space redundancy after migration should be R(t2)*S(t1)+R(t3)*(S(t2)+S(t1))+R(t4)*(S(t3)+S(t2)+S(t1)).In this paper, we design a space experiment to compare the used space between the experiments based on our Nested solution and Non-Nested solution. The space experiment data is derived from the system in the real world while the experiment database has 220 tables, 16403424 records, at the same time, the number of attributes in one table is up to 252. And this SQL database takes over 8.876G.And the result is shown in the Figure 5-3.

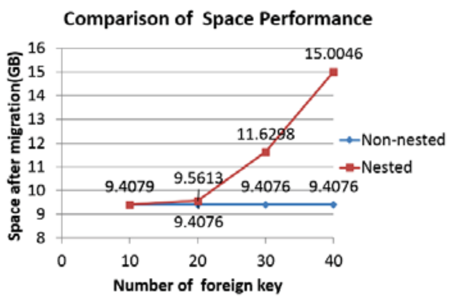

Fig. 5-3 Space Performance of Nested MongoDB and Non-nested MongoDB

In conclusion, our schema conversion solution actually causes the space redundancy to some extent. The more join keys in original database, the more space redundancy it comes. But in some scenario, we attached greater importance to the query efficiency and such space redundancy is tolerable.

5. Conclusion

This paper proposes relational database to NoSQL database schema conversion model, which uses nested idea to improve query speed of NoSQL database. In addition, we propose a general migration program and related algorithms based on the nested method. The experiment achieves a MySQL-MongoDB migration system, and use real-world data to verity the correctness of migration and compare the query performance. In addition, the correctness of the conversion algorithm has been conducted a rigorous proof. However, the downside of the model is the expense of a certain amount of space to exchange query efficiency. Future work will explore ways to minimize the spatial redundancy.

References

[1]Hech tR,Jablonski S.NoSQL valuation[C] //International Conference on Cloud and Service Computing. 2011.

[2] Han J, Haihong E, Le G, et al. Survey on NoSQL database[C]//Pervasive computing and applications (ICPCA), 2011 6th international conference on. IEEE, 2011: 363-366.

[3] Sadalage P J, Fowler M. NoSQL distilled: a brief guide to the emerging world of polyglot persistence[M]. Pearson Education, 2012.

[4] Scherzinger S, Klettke M, Störl U. Managing Schema Evolution in NoSQL Data Stores[J]. arXiv preprint arXiv:1308.0514, 2013.

[5] Kanade A, Gopal A, Kanade S. A study of normalization and embedding in MongoDB[C]//Advance Computing Conference (IACC), 2014 IEEE International. IEEE, 2014: 416-421.

[6] Bock D B, Schrage J F. Denormalization guidelines for base and transaction tables[J]. ACM SIGCSE Bulletin, 2002, 34(4): 129-133.

[7] Sanders G L, Shin S. Denormalization effects on performance of RDBMS[C]//System Sciences, 2001. Proceedings of the 34th Annual Hawaii International Conference on. IEEE, 2001: 9 pp.

[8] Yma P. A Framework for Systematic Database Denormalization[J]. Global Journal of Computer Science and Technology, 2009, 9(4).

[9] Wei Z, Dejun J, Pierre G, et al. Service-oriented data denormalization for scalable web applications[C]//Proceedings of the 17th international conference on World Wide Web. ACM, 2008: 267-276.
[10]Apache Software Foundation: http://sqoop.apache.org/

[11]DataX Wiki: ttp://code.taobao.org/p/datax/wiki/index/

[12] Chung W C, Lin H P, Chen S C, et al. JackHare: a framework for SQL to NoSQL translation using MapReduce[J]. Automated Software Engineering, 2013: 1-20.

[13] Li C. Transforming relational database into HBase: A case study[C]//Software Engineering and Service Sciences (ICSESS), 2010 IEEE International Conference on. IEEE, 2010: 683-687.