Journal of
Health & Medical Informatics

Zeng et al., J Health Med Informat 2017, 8:5
DOI: 10.4172/2157-7420.1000295

Research Article Open Access

Evaluation of Relational and NoSQL Approaches for Cohort Identification from Heterogeneous Data Sources in the National Sleep Research Resource

Ningzhou Zeng[1], Guo-Qiang Zhang[1], Xiaojin Li[2] and Licong Cui[1]*

[1]*Institute for Biomedical Informatics and Department of Computer Science, University of Kentucky, Lexington, USA*
[2]*Department of Electrical Engineering and Computer Science, Case Western Reserve University, Cleveland, OH, USA*

Abstract

Patient cohort identification across heterogeneous data sources is a challenging task, which may involve a complicated process of data loading, harmonization and querying. Most existing cohort identification tools use a relational database model implemented in SQL for storing patient data. However, SQL databases have restrictions on the maximum number of columns in a table, which necessitates the breaking down of high dimensional data into multiple tables and as a consequence affects query performance. In this paper, we developed two NoSQL-based patient cohort query systems based on an existing SQL-based system for the cross-cohort query in the National Sleep Resource Research (NSRR). We used eight NSRR datasets in our experiment to evaluate the performance of the NoSQL-based and SQL-based systems in data loading, harmonization and query. Our experiment showed that NoSQL-based approaches outperformed the SQL-based and are rather promising for developing patient cohort query systems across heterogeneous data sources.

Keywords: Heterogeneous data; Data integration; High-dimensional data; Relational database; NoSQL database

Introduction

Patient cohort identification is the selection of patient subgroups satisfying predefined criteria from a large population in Electronic Health Records (EHR) systems [1]. It is important for clinical trial recruitment [2], outcomes research [3] and other research studies [4,5]. However, the process of identifying patient cohorts can be challenging and expensive when the patient data come from heterogeneous sources. Researchers have developed automated tools and systems to query patient cohorts from disparate data sources [6-10].

The existing cohort identification tools use SQL-based relational databases as the database model for managing patient data. However, one limitation of SQL databases is the restriction on the maximum number of columns that can be stored in a table. As a result, a single table may not be sufficient to store patient data with extremely high dimensions (or a large number of data elements). In such cases, splitting data into multiple tables is an alternative strategy. However, such splitting may cause extra data loading effort and affect query performance for data elements across multiple tables. NoSQL databases may provide a better choice to handle such high dimensional patient data.

This paper provides an evaluation on SQL and NoSQL approaches for patient cohort identification across multiple data sources. We compare three patient cohort identification systems, utilizing MySQL, Mongo DB and Cassandra as the backend database, respectively. We use eight de-identified patient datasets from the National Sleep Research Resource to compare the performance of data loading and querying using these three systems.

Background

National sleep research resource (NSRR)

Funded by the National Heart, Lung and Blood Institute, NSRR was designed to share de-identified sleep data obtained from NIH-funded cohort studies and clinical trials with the sleep research community [11]. NSRR provides a web-based data portal [12] that aggregates and organizes signal and clinical data from over 26, 000 patient subjects. NSRR has over 2, 500 registered users since it's launching in 2014. Up to date, over 80 terabytes of data have been downloaded by the sleep research community.

Clinical data in NSRR are formatted in comma-separated values (CSV) files by patient visits. Each patient visit has a corresponding CSV file with all the clinical data elements collected for this visit. Note that an NSRR dataset may involve one or multiple visits. For example, the SHHS dataset has two patient visits: shhs1 (1, 266 data elements) and shhs2 (1, 302 data elements); the CHAT dataset has two visits: baseline (2, 897 data elements) and follow up (2, 897 data elements); and the CFS dataset has a single visit: visit5 (2, 871 data elements).

Specific challenges for identifying patient cohorts from heterogeneous sources

High-dimensional data: Dealing with high-dimensional data is one of the challenges for patient cohort identification using relational databases due to the limitation of the maximum number of columns in a table. For example, MySQL has a hard limit of 4, 096 columns per table, but the actual maximum number for a given table may be even less considering the maximum row size and the storage requirements of individual columns [13-27]. High-dimensional data (or column-intensive data), if exceeding a single table's capacity, need to be split into multiple tables. For instance, in the CFS dataset, the "visit5" table needs to be split into 3 tables with the de-identified patient identifiers to connect the separated tables (Figure 1). The consequence of such splitting is that it would be more computationally expensive to query data elements located in different tables since it involves costly join

***Corresponding author:** Licong Cui, Institute for Biomedical Informatics and Department of Computer Science, University of Kentucky, Lexington, USA, Tel: (859) 257-3062; E-mail: licong.cui@uky.edu

Received November 27, 2017; **Accepted** December 04, 2017; **Published** December 05, 2017

Citation: Zeng N, Zhang GQ, Li X, Cui L (2017) Evaluation of Relational and NoSQL Approaches for Cohort Identification from Heterogeneous Data Sources in the National Sleep Research Resource. J Health Med Informat 8: 295. doi: 10.4172/2157-7420.1000295

operation of tables and matching of the unique identifiers. Therefore, the query performance may be significantly affected due to the split.

Heterogeneous data: Querying heterogeneous data to find patient cohorts is also a challenging task, as disparate data sources may use different representations to express the same meaning. For example, in NSRR, different coding's for patient gender are used in disparate datasets: 1 means male and 2 means female in the SHHS dataset, while 0 represents female and 1 represents male in the CHAT dataset. Such coding inconsistencies happen frequently as the number of disparate datasets increases, thus need to be harmonized to guarantee accurate queries.

Usually, there are two ways to handle coding inconsistencies. One way is to harmonize the inconsistencies in the data loading step, where the source data of each dataset need to be updated to share uniform coding's across all the datasets.

The other way is to address the inconsistency issue in the data query step, where a mapping of the heterogeneous coding's in each dataset to the uniform coding's needed to be incorporated when the patient cohort identification system performs the query translation. In this paper, we adapt the first way to perform harmonization in the data loading step so that we can evaluate both data harmonization and query performance of the SQL- and NoSQL-based systems.

NoSQL databases: NoSQL [28] databases have been rapidly emerged, becoming a popular alternative to the existing relational databases that can better store, process and analyse large-volume data.

Without a fixed data schema, NoSQL databases are more flexible in dealing with various data sources and formats. NoSQL databases have shown the potential in managing big biomedical data [29-31]. For example, Tao et al. [31] have developed a prototype query engine for large clinical data repositories utilizing MongoDB as the backend database. There are two main components in MongoDB: 1) MongoDB Query Language; 2) MongoDB Data Model.

1) MongoDB Database System: MongoDB [32] is a free, open source and cross-platform NoSQL database. It is a mature document-oriented NoSQL database with well-written documentation and large-scale commercial use. MongoDB also provides rich drivers for multiple programming languages.

- MongoDB Query Language: As a NoSQL database, MongoDB provides an expressive query language that is completely different from SQL. There are many ways to query documents: simple lookups, creating sophisticated processing pipelines for data analytics and transformation, or using faceted search, JOINs and graph traversals.
- MongoDB Data Model-Data as Document: The major feature of MongoDB is that it stores data in a binary representation called BSON (Binary JSON). The encoding of BSON extends the widely used JSON (JavaScript Object Notation) representation to include additional types such as int, long, date, floating point and decimal 128. BSON documents contain one or more fields and each field contains a value of a specific data type, including arrays, binary data and sub-documents. Documents that share a similar structure are organized as collections. One can think of collections as being analogous to tables in a relational database: documents are similar to rows and fields are the equivalence of columns.

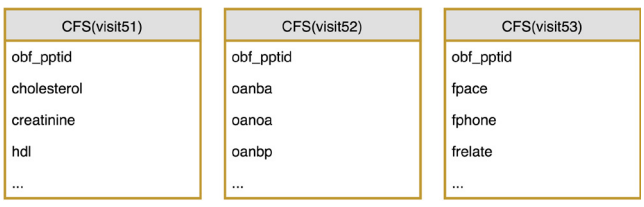

Figure 1: An example of splitting a table with a large number of columns into multiple tables in MySQL due to the restriction on the table column count.

2) Cassandra Database System: Apache Cassandra [33] is another free and open-source distributed NoSQL database management system, which is designed to store large amounts of data from multiple servers. Cassandra can be considered as a hybrid of key-value-and column-based NoSQL database.

- Cassandra Query Language (CQL): CQL is a query language for Cassandra database. It enables users to query Cassandra using a language similar to SQL. Language drivers are available for Java (JDBC), Python (DBAPI2), Node.JS (Helenus), Go (gocql) and C++ [34].
- Cassandra Data Model: Cassandra consists of nodes, clusters and data centres. A group of nodes or even a single node is a cluster and a group of clusters is a data centre. It provides support for clusters across multiple data centres. Cassandra is a combination of key-value and column-oriented database management system. The main components of Cassandra data model are keyspace, tables, columns and rows. A key space in Cassandra is a namespace that defines data replication on nodes. A cluster contains one key space per node. A table is a set of key-value pairs containing a column with its unique row keys. Rows are organized into tables. The first part of primary key of a table is partition key, which clusters the rows by the remaining columns of the key.

Materials and Methods

Clinical data from eight datasets in NSRR [12] are used as data sources in this paper, including Sleep Heart Health Study (SHHS) [13-15], Childhood Adenotonsillectomy Trial (CHAT) [16-18], Cleveland Family Study (CFS) [19-21], Heart Biomarker Evaluation in Apnea Treatment (HEARTBEAT) [22], Study of Osteoporotic Fractures (SOF) [23], MrOS Sleep Study (MrOS) [24], Hispanic Community Health Study/Study of Latinos (HCHS) [25] and Multi-Ethnic Study of Atherosclerosis (MESA) [26-30]. Table 1 summarizes the eight datasets in terms of the patient visit, number of data elements and number of patient subjects.

To evaluate SQL- and NoSQL-based approaches for patient cohort identification, we adapt the existing NSRR Cross Dataset Query Interface (CDQI) [31-35] based on MySQL and develop two NoSQL-based query systems using MongoDB and Cassandra, respectively. Figure 2 shows the general system architecture of the three systems. It consists of four major components: (i) database management system; (ii) Ruby driver for the database management system; (iii) query translation; and (iv) web-based cross dataset query interface. The database component serves as the data warehouse to store the actual datasets. The web-based query interface receives queries composed by users, which are then translated into the statements in the corresponding query language. The Ruby driver then executes the translated query statements to retrieve data from the database. After receiving the query results, the interface presents them to the end users.

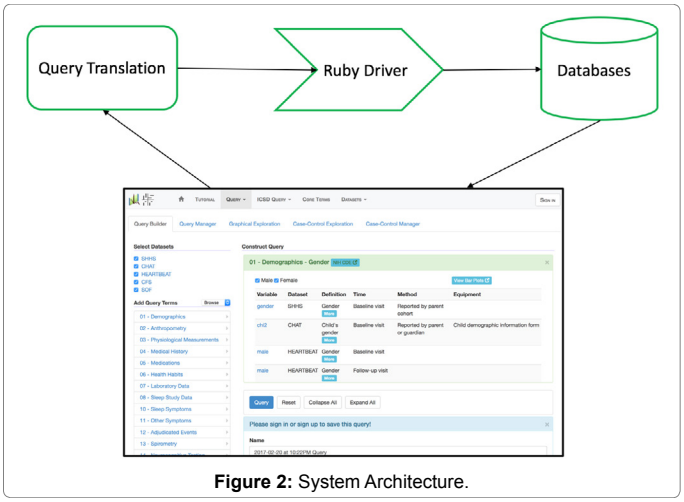

Figure 2: System Architecture.

Dataset	Visit (s)	Number of data elements	Number of subjects
SHHS	shhs1	1, 266	5, 804
	shhs2	1, 302	4, 080
CHAT	baseline	2, 897	464
	Follow up	2, 897	453
SOF	visit8	1, 114	461
MrOS	visit1	479	2, 911
	visit2	507	2, 911
HCHS	sol	404	16, 415
	sueno	505	2, 252
MESA	sleep	723	2, 237

Table 1: Summary information for each of the eight datasets.

Web-based query interface

We adapted the code base of the SQL-based NSRR CDQI in Ruby on Rails (RoR) to develop the two NoSQL-based query interfaces. RoR follows the model-view-controller architectural pattern, providing rich interaction with different types of databases and supporting HTML, CSS and JavaScript for developing interactive user interfaces. The query translation, Ruby driver and backend databases were newly implemented for MongoDB and Cassandra, respectively.

Query translation-dynamic generation of database query statement

Each time a user initiates a query through the web-based interface, the automated translation of this query (so-called query translation) into specified database query statement is needed. We illustrate the MongoDB-based query translation in the followings (MySQL-and Cassandra-based are similar).

The dynamic query translation relies on predefined general templates of MongoDB statement according to the types of queries. For example, the general template for querying a range of values for a numeric data element (or field) is predefined as:

Find (“dataset”=> <dataset.name>,

<field_1>=> {’$gte’=> <field_1_lower_value>,

’$lte’=> <field_2_upper_value>}, ...,

<field_n>=> {’$gte’=> <field_n_lower_value>,

’$lte’=> <field_n_upper_value>});

Where, the variables <dataset.name> and <field_n> represent the specific dataset and the field that the user intend to query; and <field_n_lower_value>, <field_n_upper_value> represent the user-specified minimum value and maximum value of the field, respectively. All the variables in the angle brackets can be replaced by real values to generate the actual MongoDB statement. For instance, “finding patients in the SHHS dataset aged (field_1) from 20 to 80 with height in centimetres (field_2) between 145 and 188” will have the following values for the variables in the template:

<dataset.name>: SHHS

<field_1>: age

<field_1_lower_value>: 20

<field_1_upper_value>: 80

<field_2>: height

<field_2_lower_value>: 145

<field_2_upper_value>: 188

Substituting the variables in the template with actual values obtains the following MongoDB statement:

Find (“dataset”=> “SHHS”,

“age”=> {’$gte’=> 20, ’$lte’=> 80},

“height”=> {’$gte’=> 145, ’$lte’=> 188});

Ruby driver for the database management system

As illustrated in Figure 2, we utilize certain types of databases (MySQL, MongoDB and Cassandra) as the data warehouse to store disparate datasets. All the three database management systems used in this study support a Ruby driver, which can seamlessly work with RoR to interact with the database management systems. Take MongoDB as an example, we use MongoDB Ruby driver [36] (version 2.4.1), which enables the connection to the MongoDB data warehouse and executes query statements to retrieve patient cohorts satisfying the query criteria.

Data modeling in NoSQL databases

Utilizing NoSQL databases require different data model compared to SQL relational databases.

- MongoDB: The data schema for MongoDB in this study consists of one database, called “nsrr” and one collection, called “nsrrdata”. All the eight datasets were integrated into the collection of “nsrrdata”. To differentiate records from different datasets, a key-value pair with a key as “source” was inserted into each record to indicate the source dataset of this record during the importing process. For those datasets which have more than one visit, another key-value pair with a key as “visitType” was inserted.

- Cassandra. The Cassandra database schema consists of a single cluster, called “nsrrcluster”, a single keyspace, called “nsrrdata” and eight tables corresponding to the eight datasets. Similar with MongoDB, one extra column named “visitType” was added for those datasets with more than one visit. A keyspace in Cassandra is a namespace that defines data replication on nodes. The replication strategy for replicas and the replication factor are properties from the keyspace. By selecting the replication strategy for replicas, one can determine whether data is distributed through different networks. In this work, we chose the Simple Strategy [37] since it was performed in a single cluster. Furthermore, the main purpose of this study is to compare performance rather than fault

recovery, so we set the replication factor as one. Another reason we used a single cluster is that a larger number of replicas would also interfere with the data loading time.

Data integration-loading and harmonization

Integration of disparate datasets into a data warehouse usually involves data loading and data harmonization:

- **Data loading procedure:** In MySQL-based NSRR CDQI, to load the NSRR datasets into databases, we need to perform data pre-processing. A dedicated program is needed to split the data "horizontally" into separate data files and store them in different tables. The detailed procedures for a given dataset are as follows. First, the program reads the CSV file of a patient visit in the dataset, calculates the required number of tables and splits the CSV file into multiple smaller CSV files. Then, the program reads the smaller files individually and imports them into corresponding tables. Apparently, the limitation of maximum table column count in MySQL does increase the complexity from the data loading point of view. Even though each of the eight datasets contains thousands of data elements or columns, importing data into NoSQL databases is fairly straightforward, since (1) following the data model mentioned above, we can easily import all eight datasets into the NoSQL databases; and (2) no data split is needed.

- **Data harmonization procedure:** We take three steps to harmonize coding inconsistencies before the data can be used for query: (i) we run the inconsistency detection program to detect and extract all the inconsistent coding among different datasets; (ii) we manually harmonize these inconsistency coding into uniform codings and maintain the mappings between them in a CSV file; (iii) we run another program to update the harmonized coding in corresponding tables stored in different databases. All the three query systems take similar steps to perform data harmonization.

Results

In this section, we first present the results for data loading and harmonization of the eight NSRR datasets, and then we present the comparative evaluation of the three patient cohort query systems using MySQL, MongoDB and Cassandra, respectively. All these evaluations were conducted on a computer with Intel Core i5/2.9 GHz processor and 8 GB RAM.

Data loading and harmonization

We integrated a total of 39, 342 patient records from eight NSRR sleep datasets into MySQL, MongoDB and Cassandra, respectively. Table 2 shows the numbers of tables needed for all three systems. MySQL required twenty tables due to the limitation on the table column count, while MongoDB only required one and Cassandra required eight.

We detected coding inconsistencies for 43 query concepts within eight datasets. These coding inconsistencies were harmonized into uniform codings. Take the heterogeneous codings for gender as an example, the harmonized coding is: 1-male and 2-female. For those datasets which are not consistent with this coding, the harmonization was performed to update the source data with the harmonized coding.

Database system	Number of tables
MySQL	20
MongoDB	1
Cassandra	8

Table 2: Numbers of tables needed for each database system to load the eight datasets.

Comparison of relational and NoSQL databases

We performed the comparison between SQL and NoSQL databases in terms of the data loading, data harmonization and query performance. For data loading, we compared the time spent on importing data into MySQL, MongoDB and Cassandra, respectively. For data harmonization, we compared the detected number of concepts with coding inconsistency, detection time and harmonization time. For query performance, we designed several sets of patient cohort queries that are composed of a single query concept or multiple query concepts to compare the query time. In the followings, each reported time was obtained by performing the corresponding operation five times and taking the average time.

Data loading: Table 3 shows the time taken for importing each dataset into the three database systems. It took MongoDB a total of 419.2 seconds, MySQL 337.0 seconds and Cassandra 330.9 seconds, to load 39, 342 records in the eight datasets. MongoDB took more time than MySQL and Cassandra for data loading.

Figure 3 visually demonstrates the loading time of eight datasets using MySQL, MongoDB and Cassandra, respectively.

Data harmonization: Although utilizing different databases, the first two steps for data harmonization were identical in three systems. We were able to detect coding inconsistency for the same number (43) of concepts within eight datasets in five seconds. Table 4 shows the time taken to perform data harmonization in each system. It took all the three systems over 6 h to complete the harmonization. The runtime complexities were similar since all these databases need to traverse all the records and update the corresponding column names, values (MySQL, Cassandra) or key-values (MongoDB). Cassandra required the least time to harmonize the data as it provides the best performance on the write operation.

Datasets	MySQL	MongoDB	Cassandra
SHHS	165.2s	207.7s	159.8s
CHAT	22.2s	29.3s	25.6s
CFS	22.2s	35.7s	29.8s
HEARTBEAT	1.9s	2.5s	2.2s
SOF	4.2s	4.5s	3.7s
MrOS	35.4s	39.1s	28.1s
HCHS	45.1s	56.9s	45.2s
MESA	40.8s	43.5s	36.5s
Total	337.0s	419.2s	330.9s

Table 3: Time to load eight datasets into MySQL, MongoDB and Cassandra, respectively.

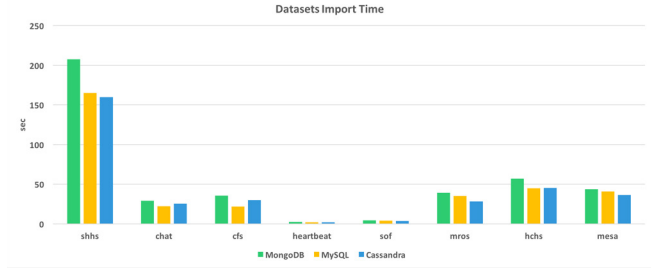

Figure 3: Data loading time comparison.

Query performance: To evaluate the query performance of the SQL-and NoSQL-based systems, we conducted experiments on performing patient cohort queries across the eight datasets. Each cohort query consists of one or more query concepts. Three sets of cohort queries were used. The first set of queries involved only one concept, while the second set and the third set involved two and four concepts, respectively.

Note that due to the limit of the table column count in MySQL, data elements exceeding the limit need to be split into multiple tables. In addition, there might be multiple data elements corresponding to the same query concept. For instance, in the SHHS dataset, there are three data elements mapped to the query concept Hypertension as follows.

- htnderv_s1: Hypertension Status based on 2nd and 3rd blood pressure readings or being treated with HTN meds;
- srhype: Self-Reported Hypertension; and
- htnderv_s2: Derived Hypertension classification (based on blood pressure measurements, history of HTN dx and medication use).

Such related data elements may be stored within the same table or across multiple tables. Therefore, a query concept may involve data elements within the same table or across multiple tables in the MySQL-based query system. We refer to query concepts involving data elements across multiple tables as cross-table query concepts.

Table 5 presents the time taken for each query using MySQL-based system. The highlighted time indicates that the corresponding query involves cross-table query concepts in the corresponding dataset. For example, in the SHHS dataset, "Age", "Asthma", "Hypertension" and "Time awake after sleep onset" are the cross-table query concepts.

System	Harmonization time
MySQL-based	6h 53m 53s
MongoDB-based	7h 9m 47s
Cassandra-based	6h 25m 15s

Table 4: Harmonization time for three systems.

As can be seen from Table 5, when querying Age in the CFS dataset, the query time was relatively short, since Age (Tables 5-7) was a within-table query concept. Even when querying two or more concepts at the same time, as long as they were from the same table, the query times were almost less than 0.1 seconds.

For the SHHS dataset, querying within-table concept Gender only took 0.03 seconds. However, when executing "AND" logic queries that contain two concepts involving different tables in MySQL, the query took more than 3 seconds. The situation could get even worse if the query consisted of multiple cross-table concepts. For instance, four query concepts "Asthma", "Gender", "Hypertension" and "Time awake after sleep onset" took about 12 seconds to complete. These illustrate that the MySQL-based system encountered a dramatic query time increase when query cross-table concepts. The major reason for such increment is that when performing such queries, the traditional relational database needs to perform costly JOIN operations.

Tables 6 and 7 show the query time taken for the MongoDB-based and Cassandra-based systems, respectively. There is no highlighted time in these two tables, since no data split operations were needed for these two NoSQL databases. For the SHHS dataset, both MongoDB and Cassandra achieved better performance when querying MySQL cross-table concepts (see the highlighted times in Table 5); however, for single-table concepts, the performance varied. For the CHAT dataset, all the queries were the cross-table concepts in MySQL, the performance of MongoDB and Cassandra were sometimes better than that of MySQL, while sometimes worse. This may be because the CHAT dataset contains a small number of patient records (917, see Table 1), in which case MySQL was efficient in performing the JOIN operation on data across tables.

Figure 4 shows the average time taken for each query using three different database systems. We can see that both MongoDB and Cassandra achieved consistently faster query performance compared to MySQL. MongoDB demonstrated the best query performance. MySQL performance was highly dependent on the query concepts.

Statistical evaluation of average query time: To evaluate the statistical significance of the differences in the average query times. We conducted

Query concept					MySQL				
	SHHS	CHAT	CFS	HEARTBEAT	SOF	MrOS	HCHS	MESA	Average
Age	**3.10s**	**1.53s**	0.019s	0.56s	NA	NA	0.04s	NA	1.04s
Gender	0.03s	**0.06s**	0.006s	0.02s	0.04s	0.21s	**5.21s**	0.18s	0.72s
Asthma	**3.63s**	**0.06s**	0.013s	0.009s	NA	**1.23s**	0.039s	NA	0.83s
Hypertension	**3.33s**	**0.04s**	0.011s	0.006s	NA	**1.64s**	0.04s	NA	0.84s
Time awake after sleep onset	**3.59s**	**0.19s**	0.12s	NA	NA	NA	NA	0.009s	0.97s
Weight	0.10s	**0.05s**	0.009s	0.02s	NA	**1.14s**	NA	NA	0.26s
Gender and weight	0.05s	**0.06s**	0.007s	**0.03s**	NA	**1.29s**	NA	NA	0.29s
Asthma and gender	**6.51s**	**0.05s**	0.01s	0.013s	NA	**1.46s**	**5.50s**	NA	2.25s
Asthma and hypertension	**6.18s**	**0.12s**	0.028s	0.007s	NA	**2.15s**	0.07s	NA	1.43s
Hypertension and time awake after sleep onset	**5.27s**	**0.12s**	0.052s	NA	NA	NA	NA	NA	1.81s
Asthma and gender and hypertension and time awake after sleep onset	**12.90s**	**0.31s**	0.04s	NA	NA	NA	NA	NA	4.42s
Asthma and weight and hypertension and time awake after sleep onset	**10.21s**	**0.21s**	0.029s	NA	NA	NA	NA	NA	3.48s
NA means unavailable information and Bold numbers indicate that corresponding query concept(s) involve data elements from multiple table									

Table 5: Cohort query time for the MySQL-based system.

Query concept	MongoDB								
	SHHS	CHAT	CFS	HEARTBEAT	SOF	MrOS	HCHS	MESA	Average
Age	0.15s	0.06s	0.05s	0.05s	NA	NA	0.15s	NA	0.092s
Gender	0.06s	0.06s	0.05s	0.05s	0.04s	0.06s	0.11s	0.05s	0.06s
Asthma	0.45s	0.05s	0.06s	0.06s	NA	0.08s	0.14s	NA	0.14s
Hypertension	0.31s	0.05s	0.07s	0.07s	NA	0.08s	0.14s	NA	0.12s
Time awake after sleep onset	0.36s	0.11s	0.13s	NA	NA	NA	NA	0.12s	0.18s
Weight	0.10s	0.13s	0.04s	0.05s	NA	0.05s	NA	NA	0.074s
Gender and weight	0.15s	0.04s	0.06s	0.05s	NA	0.06s	NA	NA	0.07s
Asthma and gender	0.31s	0.08s	0.07s	0.05s	NA	0.08s	0.12s	NA	0.118s
Asthma and hypertension	0.50s	0.05s	0.06s	0.07s	NA	0.08s	0.11s	NA	0.145s
Hypertension and time awake after sleep onset	0.60s	0.60s	0.11s	NA	NA	NA	NA	NA	0.44s
Asthma and gender and hypertension and time awake after sleep onset	0.61s	0.68s	0.12s	NA	NA	NA	NA	NA	0.47s
Asthma and weight and hypertension and time awake after sleep onset	0.51s	0.63s	0.11s	NA	NA	NA	NA	NA	0.42s
NA means unavailable information									

Table 6: Cohort query time for the MongoDB-based system.

Query concept	Cassandra								
	SHHS	CHAT	CFS	HEARTBEAT	SOF	MrOS	HCHS	MESA	Average
Age	0.92s	0.11s	0.05s	0.11s	NA	NA	0.82s	NA	0.402s
Gender	0.16s	0.06	0.04s	0.10s	0.06s	0.11s	0.92s	0.06s	0.19s
Asthma	0.95s	0.07s	0.08s	0.13s	NA	0.05s	0.89s	NA	0.36s
Hypertension	0.82s	0.19s	0.09s	0.15s	NA	0.07s	0.81s	NA	0.355s
Time awake after sleep onset	0.89s	0.22s	0.07s	NA	NA	NA	NA	0.26s	0.36s
Weight	0.39s	0.19s	0.09s	0.12s	NA	0.11s	NA	NA	0.18s
Gender and weight	0.55s	0.10s	0.11s	0.09s	NA	0.13s	1.11s	NA	0.35s
Asthma and gender	0.83s	0.15s	0.14s	0.12s	NA	0.14s	1.21s	NA	0.43s
Asthma and hypertension	1.01s	0.12s	0.13s	0.16s	NA	0.11s	0.12s	NA	0.275s
Hypertension and time awake after sleep onset	1.32s	0.11s	0.19s	NA	NA	NA	NA	NA	0.54s
Asthma and gender and hypertension and time awake after sleep onset	1.22s	1.11s	0.22s	NA	NA	NA	NA	NA	0.85s
Asthma and weight and hypertension and time awake after sleep onset	1.04s	1.21s	0.25s	NA	NA	NA	NA	NA	0.83s
NA means unavailable information									

Table 7: Cohort query time for the Cassandra-based system.

Comparative pair	t-value	p-value
MySQL and MongoDB	3.5785	0.001676
MySQL and Cassandra	2.93414	0.007678

Table 8: T-test result for two independent means using average query time.

t-test using two independent means with 0.05 significance level and two-tailed hypothesis. If the p-value is less than 0.05, then query performances are considered significantly different. As shown in Table 8, we can see the p-values are less than 0.05 for MySQL vs. MongoDB and MySQL vs. Cassandra. This indicates that the two NoSQL-based systems achieved a significantly better query performance than the MySQL-based system did.

Scalability: To evaluate the scalability of the SQL and NoSQL-based system, we conducted experiments on performing patient cohort queries across SHHS datasets with different scales. The rationale to use the SHHS dataset for scalability evaluation was in two folds: (i) it contained the largest number of data records among these eight datasets; (ii) it contained data elements mapping to both within-table and cross-table query concepts.

We scaled up the SHHS dataset by duplicating the original data records by three, five and ten times, which are denoted as SHHS × 3, SHHS × 5 and SHHS × 10 respectively. Note that these duplicated data also had unique identifiers starting from the last identifier of the original data record. The cohort queries were identical with those that were previously used for evaluating the query performance.

Table 9 shows the time taken for each query in different scales using the MySQL-based system. Each highlighted time indicates that the corresponding query involved cross-table query concepts.

As we can see from Table 9, when querying "Gender" for these scaled datasets, the query times were short, since "Gender" was a within-table query concept. Even for a query with two or more concepts, the query time remained short if these concepts were within-table (e.g., concepts "Gender" and "Weight"). However, when performing cross-table queries, the query times increased dramatically along with the scales.

Query concept	MySQL			
	SHHS	SHHSx3	SHHSx5	SHHSx10
Age	**3.10s**	**31.76s**	**87.16s**	**318.56s**
Gender	0.03s	0.08s	0.56s	1.44s
Asthma	**3.63s**	**33.17s**	**84.23s**	**312.09s**
Hypertension	**3.33s**	**32.14s**	**86.11s**	**306.06s**
Time awake after sleep onset	**3.59s**	**30.92s**	**81.42s**	**312.8s**
Weight	0.10s	0.21s	0.49s	1.02s
Gender and weight	0.05s	0.56s	1.47s	0.03s
Asthma and gender	**6.51s**	**57.05s**	**154.01s**	**585.13s**
Asthma and hypertension	**6.18s**	**50.12s**	**140.02s**	**581.43s**
Hypertension and time awake after sleep onset	**5.27s**	**50.71s**	**135.52s**	**580.53s**
Asthma and gender and hypertension and time awake after sleep onset	**12.90s**	**95.31s**	**258.04s**	**917.92**
Asthma and weight and hypertension and time awake after sleep onset	**10.21s**	**96.21s**	**252.79s**	**924.91s**

Table 9: Cohort query time for the MySQL-based system.

Query concept	MongoDB			
	SHHS	SHHSx3	SHHSx5	SHHSx10
Age	0.15s	0.12s	0.15s	0.25s
Gender	0.06s	0.06s	0.08s	0.10s
Asthma	0.45s	0.45s	0.56s	0.66s
Hypertension	0.31s	0.35s	0.47s	0.67s
Time awake after sleep onset	0.36s	0.29s	0.46s	0.56s
Weight	0.10s	0.13s	0.14s	0.21s
Gender and weight	0.15s	0.14s	0.16s	0.25s
Asthma and gender	0.31s	0.38s	0.47s	0.65s
Asthma and hypertension	0.50s	0.55s	0.56s	0.67s
Hypertension and time awake after sleep onset	0.60s	0.62s	0.66s	0.77s
Asthma and gender and hypertension and time awake after sleep onset	0.61s	0.68s	0.76s	0.86s
Asthma and weight and hypertension and time awake after sleep onset	0.51s	0.63s	0.65s	0.91s

Table 10: Cohort query time for the MongoDB-based system.

Query concept	Cassandra			
	SHHS	SHHSx3	SHHSx5	SHHSx10
Age	0.16s	0.17	0.24s	0.30s
Gender	0.95s	0.97s	1.08s	1.23s
Asthma	0.82s	0.81s	1.09s	1.25s
Hypertension	0.89s	1.02s	1.27s	1.45s
Time awake after sleep onset	0.39s	0.49s	0.54s	0.82s
Weight	0.39s	0.49s	0.54s	0.82s
Gender and weight	0.55s	0.61s	0.71s	0.96s
Asthma and gender	0.83s	0.95s	1.04s	1.12s
Asthma and hypertension	1.01s	1.12s	1.13s	1.36s
Hypertension and time awake after sleep onset	1.32s	1.34s	1.37s	1.51s
Asthma and gender and hypertension and time awake after sleep onset	1.22s	1.25s	1.34s	1.66s
Asthma and weight and hypertension and time awake after sleep onset	1.04s	1.21s	1.25s	1.65s

Table 11: Cohort query time for the Cassandra-based system.

For instance, when querying Age, the query times were 3.10s, 31.76s, 87.1s and 318.56s for SHHS, SHHS × 3, SHHS × 5 and SHHS × 10, respectively. The query time for concept Age was over 5 min when the number of data records was ten times larger. The situation could get even worse for queries consisting of multiple cross-table concepts. For instance, it would take 917 seconds to query four concepts "Asthma", "Gender", "Hypertension" and "Time awake after sleep onset". These illustrates that the MySQL-based system did not provide a decent scalability for high-dimensional data in our case.

Tables 10 and 11 present the query times taken for the MongoDB-based and Cassandra-based systems. For these NoSQL-based systems, there was no need to split tables for a single dataset. We can see from

the tables, both MongoDB based and Cassandra-based system achieved tremendously better performance when querying MySQL cross-table concepts.

To better demonstrate the scalability of these three systems, Figures 5, 6 and 7 show the query times of different scaled SHHS datasets for each query. In these figures, Q1 to Q12 are corresponding to the queries in Table 9 from top to bottom. We can see that the increment of query time along with the size of datasets for both MongoDB-based and Cassandra-based system was small. These NoSQL-based systems demonstrated a better scalability in terms of query performance compared to MySQL-based system.

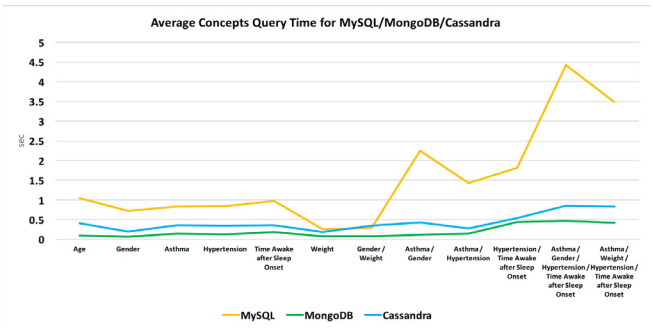

Figure 4: Average query time for each query using MySQL, MongoDB and Cassandra.

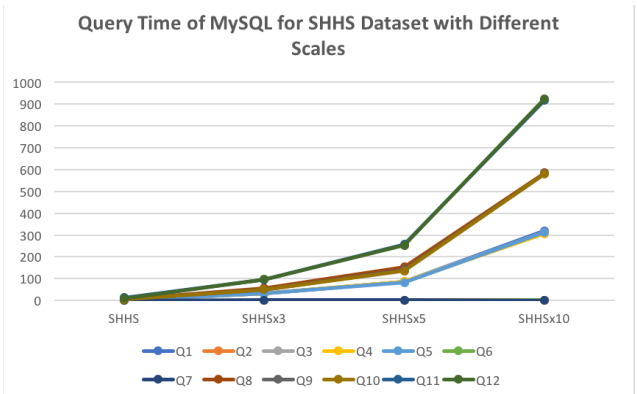

Figure 5: Query time of MySQL for SHHS Dataset with Different Scales.

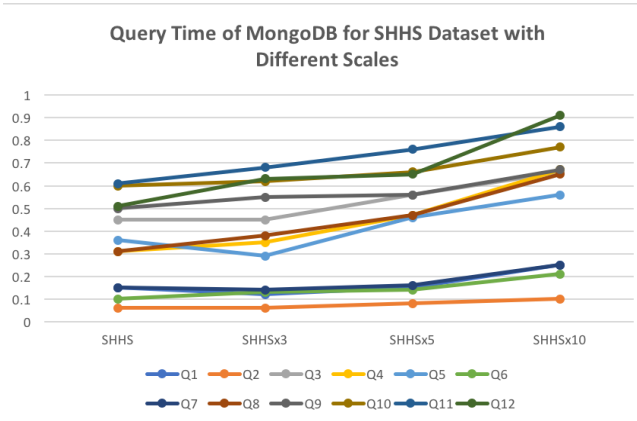

Figure 6: Query time of MongoDB for SHHS Dataset with Different Scales.

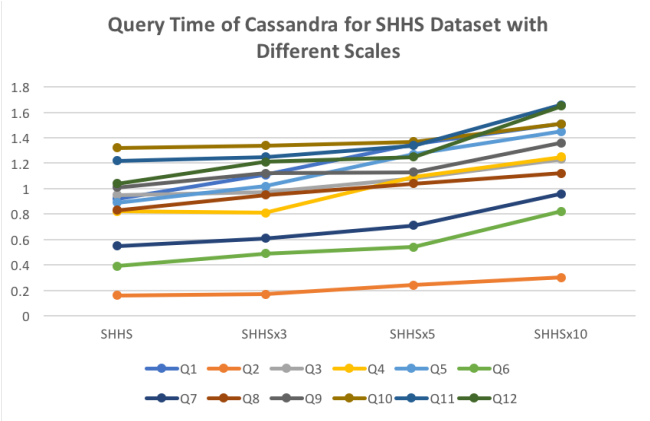

Figure 7: Query time of Cassandra for SHHS Dataset with Different Scales.

Discussion

Distinction with related work

Weber et al. [6] have developed a prototype Shared Health Research Information Network (SHRINE) based on i2b2 for the federated query of clinical data repositories. However, the i2b2/SHRINE system deals with uniform data across different i2b2 instances, where these instances share the same data structure. In this paper, we mainly focused on the heterogeneous and high-dimensional data across disparate datasets, where these datasets have different data structures.

Another related work is the MongoDB-based cohort query tool for clinical repositories [31], where the tool can be used to query a single data source. In this paper, we deal with multiple data sources and explored another NoSQL-based approach.

Limitations

A limitation of this work is that the sizes of the NSRR datasets are limited in the number of patient records (39, 342 records). Although it was shown that the NoSQL-based systems outperformed the SQL-based system on the NSRR datasets, it would be interesting to see how they perform when the number of patient records gets extremely large and to compare the actual storage required by different databases. Another limitation is that we only explored two NoSQL database systems to facilitate the patient cohort queries across disparate sources. Compared with these two, how other NoSQL databases perform still needs further investigation.

Conclusion

In this work, we developed two NoSQL-based patient cohort identification systems, in comparison to a SQL-based system, to evaluate their performance on supporting high-dimensional and heterogeneous data sources in NSRR. Utilizing NoSQL databases, we overcame the limitation of maximum table column count in traditional relational databases. We successfully integrated eight NSRR cross-cohort datasets into NoSQL databases, which largely enhanced the query performance compared to the MySQL-based system, while maintained similar performance for data loading and harmonization. This study indicates

that NoSQL-based systems offer a promising approach for developing patient cohort query systems across heterogeneous data sources.

Acknowledgments

The authors thank the University of Kentucky Center for Clinical and Translational Science (Clinical and Translational Science Award UL1TR001998), as well as National Sleep Research Resource (NHLBI R24HL114473) for supporting this work.

References

1. Shivade C, Raghavan P, Fosler-Lussier E, Embi PJ, Elhadad N, et al. (2013) A review of approaches to identifying patient phenotype cohorts using electronic health records. J Am Med Inform Assoc 21: 221-230.
2. Ni Y, Kennebeck S, Dexheimer JW, McAneney CM, Tang H, et al. (2014) Automated clinical trial eligibility prescreening: Increasing the efficiency of patient identification for clinical trials in the emergency department. J Am Med Inform Assoc 22: 166-178.
3. Kurian AW, Mitani A, Desai M, Yu PP, Seto T, et al (2014) Breast cancer treatment across health care systems: Linking electronic medical records and state registry data to enable outcomes research. Cancer 120: 103-111.
4. Mathias JS, Gossett D, Baker DW (2012) Use of electronic health record data to evaluate overuse of cervical cancer screening. J Am Med Inform Assoc 19: e96-101.
5. Deng Y, Denecke K (2016) Patient records retrieval system for integrated care in treatment of cervical spine defect. Springer 1: 10-25.
6. Weber GM, Murphy SN, McMurry AJ, MacFadden D, Nigrin DJ, et al. (2009) The Shared Health Research Information Network (SHRINE): A prototype federated query tool for clinical data repositories. J Am Med Inform Assoc 16: 624-630.
7. Zhang GQ, Siegler T, Saxman P, Sandberg N, Mueller R, et al. (2010) VISAGE: A query interface for clinical research. AMIA Jt Summits Transl Sci Proc 2010: 76.
8. Bache R, Miles S, Taweel A (2013) An adaptable architecture for patient cohort identification from diverse data sources. J Am Med Inform Assoc 20: 327-333.
9. Zhang GQ, Cui L, Lhatoo S, Schuele SU, Sahoo SS (2014) MEDCIS: Multimodality epilepsy data capture and integration system. AMIA Annu Symp Proc 2014: 1248-1257.
10. Goodwin TR, Harabagiu SM (2016) Multi-modal patient cohort identification from EEG report and signal data. AMIA Annu Symp Proc 2016: 1794-1803.
11. Dean DA, Goldberger AL, Mueller R, Kim M, Rueschman M, et al. (2016) Scaling up scientific discovery in sleep medicine: The National Sleep Research Resource. Sleep 39: 1151-1164.
12. https://sleepdata.org
13. https://sleepdata.org/datasets/shhs
14. Quan SF, Howard BV, Iber C, Kiley JP, Nieto FJ, et al. (1997) The sleep heart health study: Design, rationale and methods. Sleep 12: 1077-1085.
15. Redline S, Sanders MH, Lind BK, Quan SF, Iber C, et al. (1998) Methods for obtaining and analyzing unattended polysomnography data for a multicenter study. Sleep 7: 759-767.
16. https://sleepdata.org/datasets/chat
17. Redline S, Amin R, Beebe D, Chervin RD, Garetz SL, et al. (2011) The Childhood Adenotonsillectomy Trial (CHAT): Rationale, design and challenges of a randomized controlled trial evaluating a standard surgical procedure in a pediatric population. Sleep 11: 1509-1517.
18. Marcus CL, Moore RH, Rosen CL, Giordani B, Garetz SL, et al. (2013). A randomized trial of adenotonsillectomy for childhood sleep apnea. New England J Med 25: 2366-2376.
19. https://sleepdata.org/datasets/cfs
20. Redline S, Tishler PV, Tosteson TD, Williamson J, Kump K, et al. (1995) The familial aggregation of obstructive sleep apnea. Am J Respir Crit Care Med 1: 682-687.
21. Redline S, Tishler PV, Schluchter M, Aylor J, Clark K, et al. (1999) Risk factors for sleep-disordered breathing in children. Associations with obesity, race and respiratory problems. Am J Respir Crit Care Med 1: 1527-1532.
22. https://sleepdata.org/datasets/heartbeat
23. https://sleepdata.org/datasets/sof
24. https://sleepdata.org/datasets/mros
25. https://sleepdata.org/datasets/hchs
26. https://sleepdata.org/datasets/mesa
27. https://dev.mysql.com/doc/refman/5.7/en/column-count-limit.html
28. Kaur K, Rani R (2013) Modeling and querying data in NoSQL databases. IEEE 2013: 17.
29. Schulz WL, Nelson BG, Felker DK, Durant TJ, Torres R (2016) Evaluation of relational and NoSQL database architectures to manage genomic annotations. J Biomed Inform 64: 288-295.
30. Goli-Malekabadi Z, Sargolzaei-Javan M, Akbari MK (2016) An effective model for store and retrieve big health data in cloud computing. Comput Methods Programs Biomed 132: 75-82.
31. Tao S, Cui L, Wu X, Zhang GQ (2017) Facilitating cohort discovery by enhancing ontology exploration, query management and query sharing for large clinical data repositories. AMIA Ann Symposiu Proceed 1: 2-4.
32. Chodorow K (2013) MongoDB-The definitive Guide.
33. Lakshman A, Malik P (2010) Cassandra: A decentralized structured storage system. Operat Sys Rev 44: 3540.
34. https://github.com/datastax/cpp-driver
35. https://www.x-search.net/
36. https://github.com/mongodb/mongo-ruby-driver
37. http://www.datastax.com/doc-source/pdf/dse31.pdf