Moving from Relational Data Storage to Decentralized Structured Storage System

Upaang Saxena, Shelly Sachdeva, and Shivani Batra

Department of Computer Science Engineering
Jaypee Institute of Information Technology, Noida, India
{upaangsaxena,ms.shivani.batra}@gmail.com,
shelly.sachdeva@jiit.ac.in

Abstract. The utmost requirement of any successful application in today's environment is to extract the desired piece of information from its Big Data with a very high speed. When Big Data is managed via traditional approach of relational model, accessing speed is compromised. Moreover, relational data model is not flexible enough to handle big data use cases that contains a mixture of structured, semi-structured, and unstructured data. Thus, there is a requirement for organizing data beyond relational model in a manner which facilitates high availability of any type of data instantly. Current research is a step towards moving relational data storage (PostgreSQL) to decentralized structured storage system (Cassandra), for achieving high availability demand of users for any type of data (structured and unstructured) with zero fault tolerance. For reducing the migration cost, the research focuses on reducing the storage requirement by efficiently compressing the source database before moving it to Cassandra.

Experiment has been conducted to explore the effectiveness of migration from PostgreSQL database to Cassandra. A sample data set varying from 5,000 to 50,000 records has been considered for comparing time taken during selection, insertion, deletion, and searching of records in relational database and Cassandra. The current study found that Cassandra proves to be a better choice for select, insert, and delete operations. The queries involving the join operation in relational database are time consuming and costly. Cassandra proves to be search efficient in such cases, as it stores the nodes together in alphabetical order, and uses split function.

1 Introduction

With growing technology, data and its users are growing exponentially. This exponentially growing data is termed as Big Data. It analyses both structured and unstructured data. Big Data is as important to business and society as the Internet because of the fact that more data leads to more accurate analysis. Thus, data needs to be managed carefully for getting efficient results. Big Data is so large that it is difficult to process using traditional database and software techniques. In most enterprise scenarios, the data is big, moves very fast, and exceeds current processing capacity. Big Data is characterized by five parameters, namely, Volume, Variety, Velocity, Variability and Complexity. 'Volume' refers to the size of the data which determines

W. Chu et al. (Eds.): DNIS 2015, LNCS 8999, pp. 180–194, 2015.

the value and potential of the data under consideration. 'Variety' signifies heterogeneity of Big Data. Big Data consists of not only structured data but also unstructured data. It may constitute images, text, notes, graphs, numbers and dates. 'Velocity' in the context refers to the speed of generation of data or how fast the data is generated and processed to meet the demands and the challenges which lie ahead in the path of growth and development. 'Variability' mentions the inconsistency which can be shown by the data at times, thus hampering the process of being able to handle and manage the data effectively.

Data management becomes very complex process, especially when large volumes of data come from multiple sources. The data needs to be linked, connected and correlated for capturing the information that is conveyed. This situation is termed as the 'Complexity' of Big Data.

The choice of traditional approach (relational model) is assumed to be most promising for storing data due to its power of querying database in an efficient manner rapidly. This assumption proves invalid as data grows in size. Relational databases are not adequate to support large-scale systems due to limitations in their architecture, data model, scalability and performance [1]. This laid down the need for another model which should fulfill the requirement of high availability of data rapidly.

Companies such as Google and Amazon were pioneers to hit problems of scalability and came up with solutions, namely, Big Table [2] and Dynamo [3] respectively. Big Table and Dynamo relax the guarantees provided by the relational data model to achieve higher scalability. Subsequently, a new class of storage systems was proposed named as 'NoSQL' systems. The name first meant 'do not use SQL if you want to scale' and later it was redefined to 'not only SQL' (which means there exist other solutions in addition to SQL-based solutions). NoSQL database named as Cassandra [4] has been proposed by Facebook using the properties of Big Table and Dynamo DB. Cassandra is an open source database and is used to store chats in Facebook [4].

Current research focuses on Cassandra for improving the availability of Big Data. It started with the aim of migrating existing data in PostgreSQL to Cassandra to achieve the benefits of big data analytics. To mitigate overall cost of moving from relational to NoSQL, authors in the current research is performing compression of data before migration using a well-known compression technique, i.e., Snappy Algorithm.

The paper is further divided into following sections. Section 2 highlights the motivation behind the current research and related work. Section 3 proposes a framework for migrating EAV database to Cassandra and provides its implementation details. Section 4 gives the details about experiment performed. Results of the framework implementation are shown in Section 5. Section 6 finally concludes current research and throws a light on future aspects of the work done.

2 Motivation and Related Work

The data is growing exponentially in the world, and we need a special database to handle it. For reducing space, data must be compressed before storage. Many companies are facing the problem of growing data and various methods have been evolved

and implemented to provide a commendable solution. Key motivations for the current research are as follows:

- Location dependency due to master-slave behavior of traditional SQL systems.
- Manual intervention during failover and failback situations with generally replicated SQL systems.
- Presence of mixture of structured, semi-structured, and unstructured data.
- Latency and transactional response time issues due to dependence on synchronous replication.
- Costly JOIN operations.
- Absence of method for obtaining sequential information in case of sorted data.
- Low fault tolerance of SQL databases.
- Dynamically allocation of variable length data in database.
- Unavailability of variable schema.

Due to master-slave behavior (centralized storage), availability of data is affected a lot. For improving this, we should move towards distributed storage approach. Ensuing distributed approach helps in various ways as discussed below:

- The use of distributed Database Management System (DBMS) guarantees ZERO downtime (as same data is replicated over multiple nodes and failure of a single node does not impact overall data availability). Whereas, in centralized DBMS, failure of an instance may bring down all the dependent downstream applications. The recovery of centralized DBMS is difficult task, especially when this downtime is not planned.
- Distributed model of DBMS helps in better load balancing across different nodes. Every node is responsible for providing/processing the request for a pre-defined subset of data, which can be configured in central (or master) node, which decides which node will be responsible for providing/ processing which set of data. Whereas, in centralized database there is no secondary instance or replicas available, all the requests need to be processed by central instance. Thus, on a busy day user queries can overwhelm the centralized system, bringing down the responsiveness of the server drastically.
- One of the main advantages of using distributed database is scalability. For increasing the size of database, simply adding one physical instance will work for enhancing the capabilities of database. Whereas in centralized database, it is limited by memory or processing speed of the server where centralized database is mounted on and it cannot be expanded after a certain limit.
- Disaster recovery is easy and reliable in distributed DBMS. Suppose an application is relying on a database which has been mounted on four physical instances (i.e., data has been replicated on four nodes). If one of the nodes fails, recovery will comprise of following steps: (i) Identifying the failure of a node, (ii) Letting the master node (responsible for allocating requests among different nodes), (iii) Removing the failed node from cluster, (iv) Bringing one fresh node / repaired node to the cluster, and replicating the current data from other nodes to this node.

All this happens without impacting applications and users that are sending requests to database. There is no failure at any given point of time for users and external applications. This type of recovery mechanism is not possible in case of centralized DBMS.

Wang G. et. al. reveals the secret of NoSQL in [5]. CAP theorem [6], BASE theorem [7] and Eventual Consistency theorem [8] construct the foundation stone of NoSQL. It is often used to describe a class of non-relational databases that scale horizontally to very large data sets, but in general do not make ACID guarantees. NoSQL data stores vary ideally in their offerings. Consistency means that all copies of data in the system appear same to the outside observer at all times. Availability means that the system as a whole continues to operate inspite of node failure [9]. For example, the hard drive in a server may fail. Partition-tolerance requires that the system continue to operate inspite of arbitrary message loss. Such an event may be caused by a crashed router or broken network link which prevents communication between groups of nodes. Depending on the intended usage, the user of Cassandra can opt for Availability + Partition tolerance or Consistency + Partition tolerance.

To meet the needs of reliability and scalability as described above, Facebook has developed Cassandra [4]. It is one of NoSQL databases used by Twitter, and Facebook. Cassandra is an open source database management system. It is a distributed storage system for managing very large amounts of structured data spread out across many commodity servers, while providing highly available service with no single point of failure [4]. It guarantees the availability of database with Zero downtime at the cost of data redundancy. It aims to run on the top of an infrastructure of hundreds of nodes (possibly spread across different data centers). At this scale, small and large components fail continuously. The way Cassandra manages the persistent state in the face of these failures drives the reliability and scalability of the software systems relying on this service. It resembles database in many ways and shares many design and implementation strategies. However, it does not support a full relational data model, but provides clients with a simple data model that supports dynamic control over data layout and format.

DataStax Corporation examines the why's and how's of migrating from Oracle's MySQL to Cassandra technology [1]. Database migrations in particular can be resource intensive. Thus, IT professionals should ensure that they are taking the right decision before making such a move. Because of rapid expansion of big data applications, various IT organizations are migrating away from Oracle's MySQL or planning to do so. These companies either have existing systems transforming into big data systems, or they are planning new applications that are big data in nature and need something 'more' than MySQL for their database platform. To grasp the advantages of distributed environment, authors in the current research aims at providing a framework for transferring data stored in Relational data storage to Decentralized structured storage system.

3 Moving from Relational Data Storage to Decentralized Structured Storage System

This section gives the details of architecture of decentralized structured storage system (Cassandra), followed by its comparison with other Relational Database Management Systems (RDBMS). It discusses moving from relational data storage (PostgreSQL database) to decentralized structured storage system (Cassandra).

3.1 Architecture of Cassandra

Cassandra is essentially a hybrid between a key-value and a column-oriented (or tabular) database. The main focus of Cassandra architecture is that the hardware failures exist and data integrity/durability should be ensured at all given times. Cassandra uses peer-to-peer distributed systems across homogenous nodes to replicate the data in all nodes in a cluster. Whenever there is a data change, each node has a sequence of commit log to execute for ensuring data consistency and all nodes have set of data at all the time. Cassandra is a row-oriented database and allows only authorized/authenticated user to connect to any node in the cluster, it uses CQL [10] (Similar to SQL) for querying data from nodes. Whenever there is a read/write request to any node, node acts as a master node for processing/executing that request and the same node decides where this request should be processed (depending on the way configuration is done for all nodes in cluster). Terminology used in Cassandra is as follows.

1. **Node:** Where the data is stored, a node can be thought of as a single physical instance where Cassandra database is mounted. It is the basic component of a distributed DBMS.
2. **Data Centers:** Data center is a collection of nodes, many nodes group together based on the data they contain and form data center
3. **Cluster:** A cluster is a group of one or more data centers.
4. **Table:** Table is a collection of ordered columns fetched by row. A row consists of columns and has a primary key. The remaining columns apart from primary key can have separate indexes and tables can be dropped, modified or inserted without impacting or interrupting updates. Example of the table formed in Cassandra is shown in Table 1.

Fig. 1 gives a snapshot of architecture of Cassandra. Architecture of Cassandra constitutes of building components, which are described as follows:

Gossip
It is a peer-to-peer communication protocol that shares location and other details of the node to other nodes in the cluster. This information is stored in each and every node in the cluster which can be used whenever any node powers out.

Table 1. Example of a table in Cassandra

Row key 1	Name	Email	Work phone	Mobile phone
	ABC	abc@pqr.com	001-123-234	912345678
Row Key 2	Name	Email	Work Phone	
	DEF	def@xyz.com	001-124-432	
Row Key 3	Name	Email		
	GHI	ghi@pqr.com		

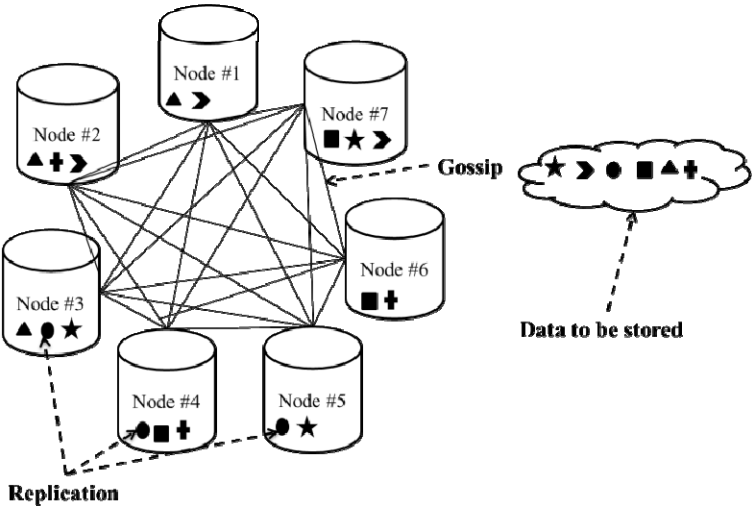

Fig. 1. Architecture of Cassandra

Partitioner

Partitioning is the 'heart' of Cassandra's architecture and used to partition the data across different nodes. Each node is responsible for processing different subset of data as assigned by partitioner. Partitioner is the hash function to compute the token of partition key. It decides which node to look at or to direct any request to (as partition key is the part of primary key in each row of data).

Replication Factor

It defines the number of replications each cluster will have. For example, replication factor of '3' means data will be replicated across nodes in the same cluster 3 times.

Replication Strategy

Replication factor determines number of copies each data will have, whereas, replication strategy determines the nodes where data has to be replicated to best suit the data availability and fault-tolerance.

Snitch

Snitch defines group of machines into data centers, which in turn is utilized by replication strategy to form the replicas and distribute them across clusters, snitch makes use of a dynamic snitch layer which choose the best replicas based on performance monitoring.

Considering Cassandra data model, data is placed in a two dimensional space within each column family. To retrieve data in a column family, users need two keys: row name and column name. In that sense, both the relational model and Cassandra are similar, although there are several crucial differences (listed below).

1. Relational columns are homogeneous across all rows in the table. A clear vertical relationship usually exists between data items; which is not the case with Cassandra columns. This is the reason Cassandra stores the column name with each data item (column).
2. In relational model 2D data space is complete. Each point in the 2D space should have at least the null value stored there. This is not the case with Cassandra, and it can have rows containing only a few items, while other rows can have millions of items.
3. In relational model the schema is predefined and cannot be changed at runtime, whereas in Cassandra users can change the schema at runtime.
4. Cassandra always stores data by sorting columns based on their names. This makes it easier to search for data through a column using slice queries. However, it is harder to search for data through a row unless we use an order-preserving partitioner.
5. Another crucial difference is that column names in RDMBS represent metadata about data, but never data. In Cassandra, the names of columns can include data. Consequently, Cassandra rows can have millions of columns, while a relational model usually has tens of columns.
6. Using a well-defined immutable schema, relational models support sophisticated queries that include JOINs, and aggregations. In relational model, users can define the schema without worrying about queries. Cassandra does not support JOINs and most SQL search methods. Therefore, schema has to be catered to the queries required by the application.

After a rigorous survey, authors presents Table 2 which compares Cassandra database, with various existing relational databases. These databases are differentiated on various parameters, such as architecture, data model, structure of queries, enterprise search, enterprise analytics, memory, security, data independence, usage and recovery. The analysis highlights the importance of Cassandra. To gain the functionalities provided by Cassandra, the current study experimented with sample data to migrate from PostgreSQL database to Cassandra.

Table 2. Comparison of Cassandra with relational database management systems

Product Capability	**DataStax Enterprise Cassandra**	**Oracle RDBMS**	**Oracle MySQL**	**Microsoft SQL Server**
Core Architecture	Masterless (no single point of failure)	Master-slave (single points of failure)	Master-slave (single points of failure)	Master-slave (single points of failure)
High Availability	Always-on continuous availability	General replication with master-slave	General read-only scale out replication; simple master-master	SQL Server replication, clustering and mirroring
Data Model	Dynamic; structured and unstructured data	Legacy RDBMS; Structured data	Legacy RDBMS; Structured data	Legacy RDBMS; Structured data
Scalability Model	Big data/Linear scale performance	Oracle RAC or Exadata	Manual sharding with MySQL	Manual sharding, general partitioning
Multi-Data Center Support	Multi-directional, multi-cloud availability	Nothing specific	Nothing specific	Nothing specific
Security	Full security support	Full security support	Full security support	Full security support
Enterprise Search	Full Solr integration	Handled via Oracle search	Full-text indexes only	Full-text indexes only
Enterprise Analytics	Integrated analytics with workload isolation with MapReduce, and Hive	Analytic functions in Oracle RDBMS via SQL MapReduce	Some analytic functions, no Hadoop support	Basic analytic functions
Database Option	Built-in in-memory option	Columnar in-memory option	MySQL cluster	Coming in-memory option
Enterprise Management & Monitoring	DataStax OpsCenter & automated management services	Oracle Enterprise Manager	MySQL Enterprise Monitor	SQL Server Enterprise Studio
Operations	No join operation, but use various other methods to show results similar to join	Performs search and insert operation (as well as various other costly operations like join)	Performs all the operations	Various operations are performed
Data Independence	Data on different data centers are independent and separable	Data depends on data types defined in the schema	Data depends on data types defined in the schema	Data depends on data types defined in the schema
Usage	Users using very large database (where not possible to store data on a SQL database)	Users of medium scale business	General users (where numbers of users are less)	General users (where numbers of users are less)
Recovery and atomicity	Remembers deletes, but full recovery is manual- using node tool	Doesn't remember delete and no chance of recovery	No provision of data recovery is present	Data can only be recovered, when log file is maintained

3.2 Moving from Relational Data Storage to Decentralized Structured Storage

Current study proposes a solution to the problem of managing growing data on various applications by migrating PostgreSQL database (centralized approach) to Cassandra (distributed approach). The whole process of migrating PostgreSQL database to Cassandra is divided in three layers as shown in Fig. 2. Firstly, authors are applying Google Snappy algorithm on the data to reduce the size of data stored in the database which directly impacts the cost of migration. After the size is reduced, an intermediate database (MySQL) is used to transfer whole database on a NoSQL database (Cassandra).

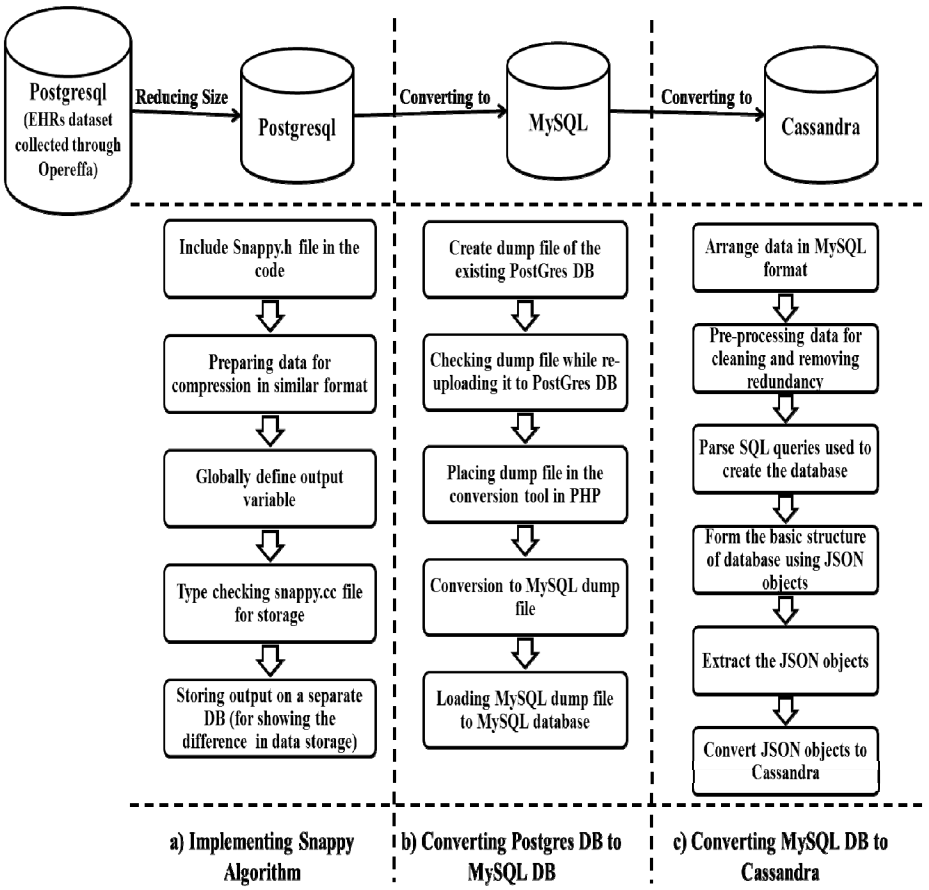

Fig. 2. Migrating from Relational data storage to Decentralized structured storage

3.3 Compressing Database

Snappy (previously known as Zippy) is a fast data compression and decompression library written in C++ by Google based on ideas from LZ77 [1, 11]. It does not aim for maximum compression, or compatibility with any other compression library; instead, it aims for very high speeds and reasonable compression. Compression speed is 250 MB/s and decompression speed is 500 MB/s using a single core of a Core i7 processor running in 64-bit mode. The compression ratio is 20–100% lower than gzip [12]. Snappy is widely used in Google projects like BigTable, MapReduce and in compression data in Google's internal RPCsystems. It can be used in open-source projects like Cassandra, Hadoop, LevelDB, RocksDB, Lucene[9]. Decompression is tested to detect any errors in the compressed stream. Snappy does not use inline assembler and is portable [13]. Fig. 3 describes the complete Google Snappy algorithm working and is delivering a final product – Snappy test file, which will take a database as an input and will compress the data to almost 80% of original data and will deliver a final product as a compressed database. For the implementation of Snappy algorithm, we need to include this 'snappyfile.h' file in our program, and it will compress the data, which is the input in the code. Snappy test file is the file used for testing of Snappy algorithm, and we need it to check, whether the compressing algorithm is working fine or not. For delivering the end product, we need to include snappy public file in the code, which defines all the functions necessary to compress the data. Apart from this, we also need 'snappy.h' file and 'snappy.cc' file, which is the header file of Snappy algorithm, and is provided open source by Google. Fig. 2(a) defines the complete process of application of Google Snappy algorithm on data.

3.4 PostgreSQL to Cassandra

The data stored in PostgreSQL database is converted into its dump files. The idea behind this dump method is to generate a text file with SQL commands. Consequently, this is fed back to the server, which will recreate the database in the same state as it was at the time of the dump. Authors used 'pg_dump' command to create the dump file of the existing database (in PostgreSQL), which we need to convert. This dump file is converted to a MySQL dump file, which can further be loaded into SQL database. This converts PostgreSQL Database to MySQL database. Fig. 2(b) describes the process of conversion of data from a PostgreSQL Database to MySQL database, considering that we should not get any loss of data during transfer.

Cassandra is a database with variable schema. It is column oriented, and gives us the flexibility to store data of different types on a same database. This gives us an advantage of storing data together. Thus, we considered moving this MySQL database to NoSQL database (Cassandra). Our work is implementation of the architecture described by Phani Krishna in [14]. To move the database from a relational model to Cassandra, authors are using ETL tools. They firstly extract the data, and transform the data by cleansing and enriching it to a processed data. Then, the data is further parsed and converted to JSON objects. Then with the help of JSON files [15], it converts the database to Cassandra. Fig. 2(c) shows the way data is moved from MySQL

database to Cassandra database after parsing the queries and obtaining the JSON objects from the SQL query. Fig. 4 presents a snapshot of JSON object creator from MySQL database. These JSON objects are further converted to Cassandra and we get our desired result.

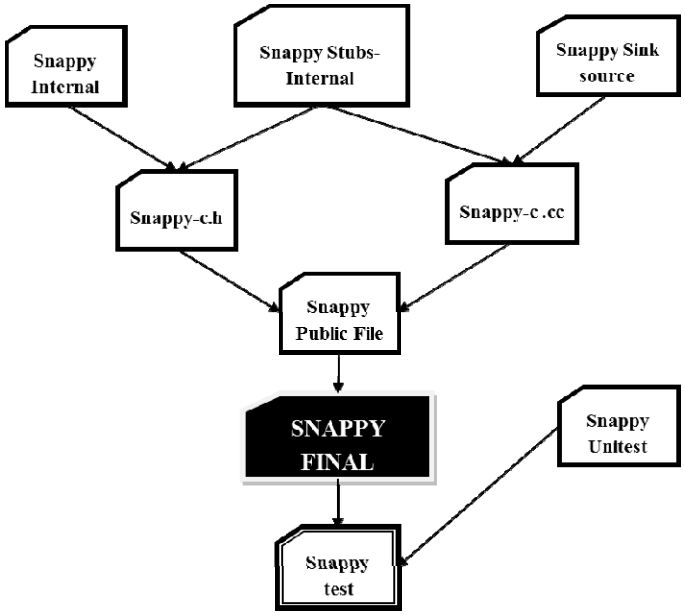

Fig. 3. File map of GOOGLE Snappy Algorithm

Fig. 4. JSON objects Creator from a MySQL database

4 Experiments

For experimentation authors are using datasets of standardized Electronic Healthcare Records (EHRs) database [16-17]. Speedy access of information is highly demanded by the users now a days. There are millions of health organizations and billions of users. In such scenario, data must be stored in such a way which guarantees instant access. Availability is very critical for healthcare domain as unavailability of right information at right time may even result in loss of patient life. Other than availability; standardization, sparseness and volatility are also very critical for EHRs [18]. Several standard development organizations (openEHR, CEN, ISO and HL7) [19-23] are working to provide a standard which can be adopted by every heath organization to achieve globalization. To deal with sparseness and volatility Entity Attribute Value (EAV) model is preferred over relational model [18]. For implementing the framework proposed in previous section, we considered the database, which contains standardized EHRs data. EHRs data was synthesized by the authors using a clinical application named Opereffa [24]. Opereffa stands for openEHR REFerence Framework and Application. We explored Opereffa for our research purpose and found that it stores EHR data in a standard format (openEHR) in a single generic table (based on EAV model) using PostgreSQL. Fig. 5 provides a snapshot of the data collected through Opereffa by the authors. Database shown in Fig. 5 follows EAV model approach where "context_id" specifies the entity, "archetype path" resemble attribute column of EAV model and "value_string", "value_int" and "value_double" are analogical to the value part of EAV model. As EAV approach is followed in Opereffa, the stored dataset will be free from sparse entries. The dataset stored through Opereffa is available at a single computer where Opereffa is running. Everyone who wants to access the data needs to communicate to the single point of contact. Due to this centralized storage, availability is affected a lot. To improve on availability, we should move towards the distributed storage approach (Cassandra) by implementing the framework proposed in previous section. Sample datasets varying from 5000 to 50,000 have been used for conducting experiments for this study.

	id [PK] bigint	context_id character var	archetype_n character var	archetype_p character var	name character var	value_string character var	value_int bigint	value_double double preci	session_id character var	instance_ind integer	field_created timestamp(0	archetype_cr timestamp(0	tolven_conte character var	value_at_pat character var	deleted boolean
1	19007	abc	openEHR-EHR	/data[at000	magnitude			123	0616e9c9-00	0	2014-10-28	2014-10-28			FALSE
2	19006	abc	openEHR-EHR	/data[at000	unit	/min			0616e9c9-00	0	2014-10-28	2014-10-28			FALSE
3	19005	abc	openEHR-EHR	/data[at000	value	at0006			0616e9c9-00	0	2014-10-28	2014-10-28			FALSE
4	19004	abc	openEHR-EHR	/data[at000	value	avb			0616e9c9-00	0	2014-10-28	2014-10-28			FALSE
5	19003	abc	openEHR-EHR	/data[at000	value	at0017			0616e9c9-00	0	2014-10-28	2014-10-28			FALSE
6	19002	abc	openEHR-EHR	/data[at000	value	at0012			0616e9c9-00	0	2014-10-28	2014-10-28			FALSE
7	19001	abc	openEHR-EHR	/data[at000	magnitude			123	383f858e-be	0	2014-10-28	2014-10-28			FALSE

Fig. 5. Snapshot of PostgreSQL database

5 Results

Authors have implemented Google Snappy algorithm, and have tested the data compression rate of the end product to figure out the compression rate of the algorithm.

The results obtained are shown in Table 3. To test the difference in time execution different queries have been executed by authors on standard based clinical database and the corresponding Cassandra database. To account for scalability, varying size of datasets has been considered to collect comparative results. Queries are executed considering four main parameters of any query application (insert, delete, search and select). The corresponding times taken by various queries are shown graphically in Fig. 6.

Table 3. Compression results after applying Google Snappy Algorithm

Original Data	Compressed Data
149 KB	121 KB
872 KB	536 KB
1.3 MB	807 KB

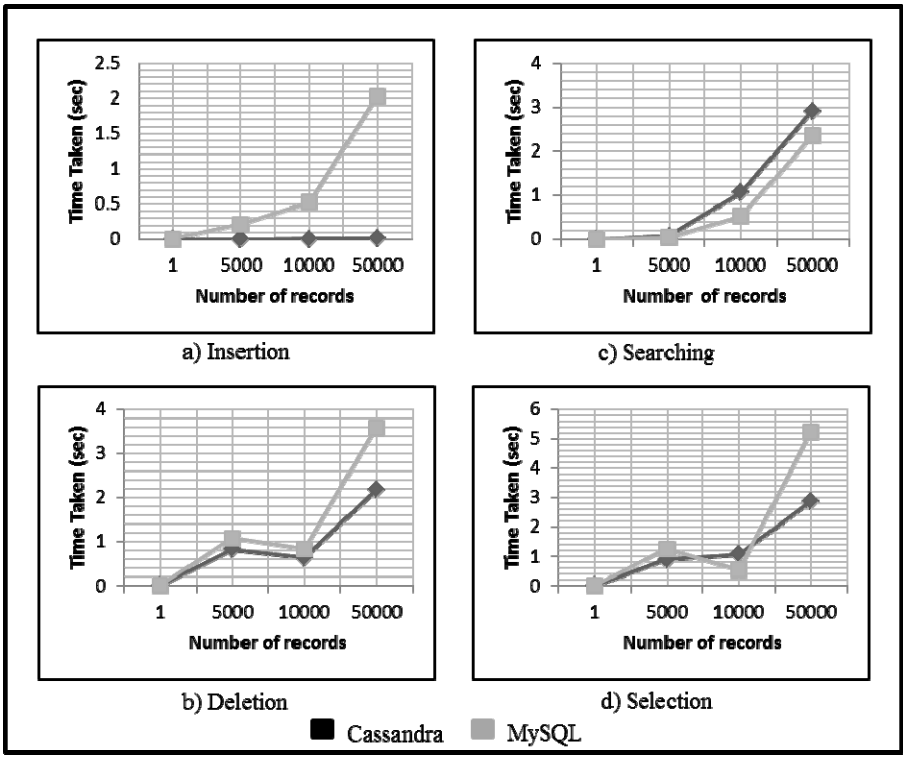

Fig. 6. Comparison of time taken in Cassandra and MySQL

Results obtained in Fig. 6 shows the benefits achieved in terms of time taken for executing queries on MySQL database and Cassandra. Time is calculated in MySQL, using *time()* function and in Cassandra, using *TRACE ON* command [25], which enables user to trace the amount of time an operation needs, in all its steps.

6 Conclusions and Future Scope

To account for the scalability, there is need of shifting to an approach different from relational model which can guarantee high availability with zero fault tolerance, one such model is Cassandra. Current research is a step towards providing an integrated solution for migrating data from relational data storage to decentralized structured storage system. The study implements a framework migrating from PostgreSQL to Cassandra. To minimize the cost of migration, authors in the current research implements Google snappy algorithm before migration. Table 2 is presented to give a clear differentiation of Cassandra from various existing RDBMS. An experimental comparative analysis is done to account for the benefits achieved after migrating to Cassandra from PostgreSQL in terms of time taken to execute a query (insert/ delete/ search/ select). Experiments are performed on different sizes of datasets (considering the scalability of data) ranging from 1 instance to 50,000 instances of standardized EHRs (based on openEHR standard).

In future, work can be done for providing security to all the data nodes and to provide Gossip Protocol complete information of all the nodes. A tool can also be built to change a PostgreSQL data directly to Cassandra, so that we can remove the use of MySQL database, and can get the desired result. This work is in an initial effort to move standard based clinical data to Cassandra. The researchers may apply more NoSQL databases such as MongoDB and Redis to perform a comparative analysis of all the databases. Current research can benefit other arenas related to Big Data such as meteorology, genomics, connectomics, complex physics simulations, biological and environmental research, internet search, finance and business informatics.

References

1. DataStax Corporation, White paper: Why Migrate from MySQl database to Cassandra and How? International Journal of Computer Trends and Technology 3(2) (2012)
2. Chang, F., Dean, J., Ghemawat, S., Hsieh, W.C., Wallach, D.A., Burrows, M., Chandra, T., Fikes, A., Gruber, R.E.: Bigtable: A distributed storage system for structured data. In: Proceedings of the 7th Conference on USENIX Symposium on Operating Systems Design and Implementation, vol. 7, pp. 205–218 (2006)
3. Candia, G.D., Hastorun, D., Jampani, M., Kakulapati, G., Pilchin, A., Sivasubramanian, S., Vosshall, P., Vogels, W.: Dynamo: amazonOs highly available key-value store. In: Proceedings of twenty first ACM SIGOPS symposium on Operating systems principles, pp. 205–220 (2007)
4. Lakhsman, A., Malik., P.: Cassandra - A Decentralized Structured Storage System. In: International Conference on Computing, Engineering and Information (2012)
5. Wang, G., Tang, J.: The NoSQL Principles and Basic Application of Cassandra Model. In: International Conference on Computer Science & Service System, China (2012)
6. Gilbert, S., Lynch, N.: Brewer's conjecture and the feasibility of consistent, available, partition-tolerant web services. In: vol. 33(2), pp. 51–59. Massachusetts Institute of Technology,ACM SIGACT News Homepage archiv, Cambridge (2002)
7. Theorem, B.: Practical Partition-Based Theorem Proving for Large Knowledge Bases. In: MacCartney, B., McIlraith, S., Amir, E., Uribe, T.E. (eds.) 18th Int'l Joint Conference on Artificial Intelligence, IJCAI 2003 (2003)

8. Bailis, P., Ghodsi, A.: Eventual Consistency, 'Eventual Consistency Today: Limitations, Extensions, and Beyond. ACM, UC Berkeley (2013), doi:1542-7730/13/0300
9. Featherston, D.: Cassandra: Principles and Application. In: International Conference on Computing, Engineering and Information, University of Illinois at Urbana-Champaign
10. CQL, https://cassandra.apache.org/doc/cql/CQL.html
11. PostgreSQL White Paper: How to increase performance, scalability and security within a Session Management architecture- (2005)
12. Network Defense- White Paper: Current open issues in NoSql database
13. Google Code, https://code.google.com/
14. Phani Krishna Kollapur Gandla, Migration of Relational Data structure to Cassandra (No SQL) Data structure, http://www.codeproject.com/Articles/279947/Migration-of-Relational-Data-structure-to-Cassandr
15. What is JSON, http://www.json.org
16. Beale, T., Heard, S.: The openEHR architecture: Architecture overview. In: The openEHR release 1.0.2, openEHR Foundation (2008)
17. Duftschmid, G., Wrba, T., Rinner, C.: Extraction of standardized archetyped data from Electronic Health Record Systems based on the Entity-Attribute-Value Model. International Journal of Medical Informatics 79(8), 585–597 (2010)
18. Batra, S., Sachdeva, S., Mehndiratta, P., Parashar, H.J.: Mining standardized semantic interoperable electronic healthcare records. In: Pham, T.D., Ichikawa, K., Oyama-Higa, M., Coomans, D., Jiang, X. (eds.) ACBIT 2013. CCIS, vol. 404, pp. 179–193. Springer, Heidelberg (2014)
19. OpenEHR Community (accessed 10, 2013), http://www.openehr.org/
20. CEN - European Committee for Standardization: Standards (accessed May, 09), http://www.cen.eu/CEN/Sectors/TechnicalCommitteesWorkshops/CENTechnicalCommittees/Pages/Standards.aspx?param=6232&title=CEN/TC+251
21. : ISO 13606-1.: Health informatics: Electronic health record communication. Part 1: RM, 1st edn (2008)
22. ISO 13606-2.: Health informatics: Electronic health record communication. Part 2: Archetype interchange specification, vol. 1 (2008)
23. HL7. Health level 7 (First accessed 10/13), http://www.hl7.org
24. Opereffa, http://opereffa.chime.ucl.ac.uk/introduction.jsf
25. Cassandra, Tracing on Feature, http://www.datastax.com/documentation/cql/3.0/cql/cql_reference/tracing_r.html