Data Integration from Heterogeneous Databases using Improved Tries Structure

Phyo Thandar Thant, Thinn Thu Naing
Software Technology Department, University of Computer Studies, Yangon
pphyothandarthant@gmail.com, thinnthu@gmail.com

Abstract

Cloud Computing is one of the converging technology trends in Information Technology which can pave the way to optimized computing solution. Several research issues exist in these converging technologies. Among them, data integration from cloud based heterogeneous relational and NoSQL databases is a challenging issue in cloud based data storage. Moreover, it is also needed to handle SQL and NoSQL data in a single application without knowing the user where and which type of database the needed data resides. In this paper, we proposed data integration system from heterogeneous databases using Improved Tries Data Structure technology which makes improvements to Normal Tries space requirements and structure size of tries. As a result, the system can give several advantages such as providing data transparency to the users , eliminating some barriers in bridging the relational and NoSQL databases in cloud data storage.

Keywords— NoSQL, Tries, Arithmetic Coding, Data Integration, Cloud Computing

1. Introduction

Cloud usage is in various areas such as application hosting, content delivery, ecommerce, and web hosting. Widespread usage of cloud computing technologies leads to data handling problem with heterogeneous data sources. Data handling problem can be solved with the data integration technology. A distributed database can be defined as a database logically integrated but physically distributed on several machines that can communicate through a network infrastructure. Several database experts argued that Relational Database Management System can no longer handle all the data management issues encountered by many current applications. The main advantage of a relational database is data accessibility and ease in retrieving the necessary information. When the situation requires a large set of data, relational databases lose their power. Several systems have already emerged to propose an alternative to RDBMS and many of them are categorized under the term NoSQL, also called ‘Not only SQL’.

These NoSQL databases are very popular and successful in many Big Data application domains. However, they will not displace traditional relational stores. Both NoSQL and SQL systems will continue to find and fulfill complementary data management needs. Therefore, there might be scenarios that require coordinated fashion across both relational and NoSQL databases.

To my knowledge, there exists no data integration system that can handle data from both SQL and NoSQL databases. The integrated information of the past information integration systems cannot be expressed by a standardized data from, but a system defined data format, which seriously impacts on the exchange between various systems, and the achieving process is complex, the cost is higher, difficult to get a wide range of applications. When XML technology emerges, XML data format becomes the standardization description of all kinds of irregular information and rule information possible. In this case, the system intends to implement a type of data integration for heterogeneous data sources using Improved Tries Algorithm for memory efficiency.

This paper intends to present data integration system for heterogeneous databases i.e., relational and NoSQL databases in order to provide all the benefits of these databases. The rest of paper is organized as follows: section 2 presents motivation of this work, section 3 explains why we should we move to NoSQL, section 4 explains the problem definition for the need of data integration system , section 5 shows the contributions of the system, section 6 lists related work, section 7 describes the theory Background, section 8 gives information about the proposed Data integration System and section 9 provides conclusion.

2. Movement to NoSQL

Relational databases are the de facto standard in data storage. They have offered a good mix of flexibility, performance, scalability and compatibility in managing generic data. They also provide simplicity of development through strict consistency which takes a lot of responsibility from application developers and that has created an “one size fits all” attitude. However, in reality, one size doesn’t fit all as relational databases have particular problems to address:

(1) Scalability

Relational databases can only scale well inside the boundaries of a single server. They only provide vertical scalability and do not provide horizontal scalability which is very important feature in distributed computing environment.

(2) Availability

Relational databases have the property to be always in a consistent state. That means refusing new write operations until the current write operation is finished affecting the availability of application to some extent.

(3) Fault-tolerance

Relational databases treat hardware failures as exceptions and special hardware is required to achieve fault tolerance.

NoSQL definition by Stefan Edlich is “Next Generation Databases mostly addressing being non relational, distributed, open-source and horizontal scalable. Its intention has been modern web-scale databases. Nowadays, many companies have found existing data storage solutions inadequate and developed their own implementations. Even more businesses have adopted new open source non-relational data stores. Although the data models of the new data stores are not even nearly as rich as the relational model, it is more efficient with regard to money. So, problems of relational databases (availability, scalability and fault tolerance) are more important than strict consistency in business applications.

Moreover, the term commodity hardware is very common in today’s distributed computing environment. These days, large systems don’t require special hardware. These systems are designed to be used with clusters of commodity hardware. And horizontal scalability is achieved by adding nodes to the clusters and sharing load between clusters and the main advantage is economic. NoSQL databases enables horizontal scaling that are very important for modern distributed database environment as well as in cloud computing. Because of the above reasons, people consider to use NoSQL databases in their applications.

2. Problem Definition for the Need of Data Integration between SQL and NoSQL Data

Nowadays, developers start considering alternative types of database systems for their data storage instead of relational storage. However, both of these databases have advantages of its own. Some software products require data storage where a part of the data is ideally stored in NoSQL database whereas the rest of the data is perfectly relational and thus well suited for a traditional SQL database. In this case, a data integration system that can handle both sql and nosql queries is needed for these types of software products , providing data transparency to the users.

Moreover, in o Tries original tries algorithm, there is a need to limit the depth of the tree structure, i.e., trie structure size may not be acceptable when the key set is very large. Its searching speed is not so efficient compared to binary search algorithm.

3. Related Work

John Roijackers [4] from Netherlands presented the problems arise when a single software product requires data storage where a part of the data is ideally stored in a NoSQL database, whereas the rest of the data is perfectly relational and thus well-suited for a traditional SQL database. As different parts of the data are well-suited for different types of databases, choosing one type of data storage always implies that a part of the data is stored in a less appropriate way. According to his contribution, he considered where the relational part of the data is stored in an SQL database, while the non-relational data is stored in NoSQL . A recent trend towards the use of non-relational NoSQL databases raises the question where to store application data when part of it is perfectly relational. Dividing data over separate SQL and NoSQL databases implies manual work to manage multiple data sources. We bridge this gap between SQL and NoSQL via an abstraction layer which transformed the NoSQL data to a triple format and incorporate these triples in the SQL database as a virtual relation. Via a series of self joins the original NoSQL data can be reconstructed from this triple relation.

[5] bridged the gap between SQL and NoSQL with the help of opensource Apache DerbyDB and Cassandra . As Derby is a relational database, it uses Structured query language for querying the data and and it provide ACID guarantees. So, the operations with the database can be grouped together and treated as a single unit (atomicity) either all the operations in this single unit (transaction) are performed or none is (consistency) also, independent sets of database transaction are performed so that they don’t conflict with each other (isolation) and it also guarantees that the database is safe against unexpected termination (durability). For the NoSQL portion, Cassandra is used.

Ahuja[2] proposed a virtual Database framework Object Oriented Mediator Database System (OOMDS) that enables the centralized global object oriented database, a virtually integrated huge database that will hide the heterogeneity of various cloud databases. OOMDS is a distributed mediator system that uses a object oriented data model and has a relationally complete object oriented query language, OOMDSQL. Through its distributed object oriented multi-database facilities, many autonomous and distributed OOMDS peers can interoperate. In OOMDS mediator system , groups of distributed mediator peers are used to integrate data from different sources. Each mediator in a group has DBMS facilities for query compilation and exchange of data and meta data with other mediator peers. The mediator peers are autonomous without any central schema. A special mediator, the central name server keeps track of what mediator peers are members of a group. The central name servers can be queried for the location of mediator peers in a group. Meta queries to each mediator peer can be posted to investigate the structure of its schema. The objective of OOMDS is to hide the heterogeneity of various cloud databases and to provide a consistent access to the end users.

For the improvements in tries, [9] proposed an alternative approach using a new trie structure called structure shared trie (SS- trie). The main idea of SSTrie is to reduce unused space using shared common structure and bit compression. They proposed three techniques (1) path compression, (2) structure sharing (3) node compression. Path compression is performed by skipping internal nodes with only one child while searching because there is only on way in the path. This increase retrieval speed and reduce trie structure size. Path compression binary trie is also known as PAT. Structure sharing separates trie into 2 parts: (1) shared part and (2) unshared part. It is used to store frequently used structure in trie as in fig (1) using structure id (0 or 1). Node compression is to compress four high bits and four low bits of node transitions to bit vectors. There may be overhead as well as slow bit manipulation. [8] proposes burst trie which requires no more memory than a binary tree but as fast as a trie. It also maintains strings in sorted or near sorted order. According to their experiments, performance of burst trie depends on the distribution of strings.

4. Theory Background

This section gives some theoretical background of the paper. Firstly, introduction of relational databases, NoSQL databases, their nature and types are presented. After that, CAP theorem, Consistent Hashing Techniques are discussed.

4.1 Relational Databases

A database is a means of storing information in such a way that information can be retrieved from it. A relational database is a database constituting a set of relations. In relational database, a relation , R is a set of tuples of the same schema. A tuple , t of tuple schema τ where

$\tau = \{ (A_1,D_1), (A_2,D_2),\ldots(A_n,D_n)\}$ is a set of pairs of the form ,

$t = \{ (A_1,v_1), (A_2,v_2),\ldots(A_n,v_n)\}$

such that $\forall\ v_i \square D_i$

In simplest terms, a relational database is one that presents information in tables with rows and columns. A table is referred to as a relation in the sense that it is a collection of objects of the same type (rows). Data in a table can be related according to common keys or concepts, and the ability to retrieve related data from a table is the basis for the term relational database.

More accurately, the relational model is based on predicate logic and set theory. The mathematics behind the model make this manipulation feasible.

4.2 NoSQL

NoSQL databases, the next generation distributed databases are a cost effective alternative to relational databases. They have been successfully used by Google, Amazon, Twitter, Facebook etc. to achieve massive parallelism, unlimited scalability and high availability.

The most widely usage of NoSQL databases is in the cloud computing environment. Many cloud service providers exist and multiple NoSQL products are available. The most popular service providers are Google and Amazon. With the rise of the large Internet systems, with their huge amount of data and requests, new solutions for the distribution of databases were needed.

This is why within the last years alternative data management systems, so-called NoSQL DMS (or NoSQL data stores), have been created and are becoming more important. There are several types of NoSQL databases such as:

Key-value stores

- is a system that stores values indexed for retrieval by keys.
- systems can hold structured or unstructured data.

Column-oriented databases

- column-oriented databases contain one extendable column of closely related data.
- Facebook created the high-performance Cassandra to help power its website.

Graph Databases

- data is represented as graphs
- best suited for representing data with a high, yet flexible number of interconnections, for example, social relations or geographic data.

Document-based stores

- store and organize data as collections of documents, rather than as structured tables with uniform sized fields for each record.
- users can add any number of fields of any length to a document.

4.3 Data Integration

Data integration shields the heterogeneity of the various heterogeneous data sources and carries out unified operation to different data sources through heterogeneous data integration system. The data integration problem itself is very broad; there are several methods and architectures currently in use today. Of these, Data Warehousing (common data storage) and FDBS's (uniform data access) have become two of the most popular types. A brief overview and comprehensive example of each method will be provided to distinguish its place in the paradigm of data integration. Integrated heterogeneous data is unified for the users. Data integration is intended to carry out using RDF data format in this system.

4.4 Database Virtualization

Databases of many kinds exist in terms of their associated data model differences and vendor differences. Regarding differences among data models, each has different data representation, and unique associated manipulation. Some typical examples include the table type of relational databases (RDB), NoSQL databases, XML-representation type of XML databases (XMLDB), and object-oriented databases (OODB). To have a virtualization feature, there are several ways to manage distributed databases of similar types, the distributed databases of different types, and provide location transparency for users, such that they notice no differences of database structure or location and become able to use databases of all kinds in a flexible fashion. For virtualization of ubiquitous databases in our study, we will virtualize different databases (MySQL and MongoDB) using Tries data structure.

4.5 Tries Tree Data Structure

A trie is an ordered multi-way data structure to store strings over an alphabet. Unlike a binary search tree, no node in the tree stores key associated with that node; its position in the tree shows what key it is associated with. Each node contains an array of pointers, one pointer for each character in the alphabet and all the descendents of a node have a common prefix of the string associated with that node. Root is associated with empty string and values are normally not associated with every node, only with leaves. It is a data structure that allows strings with similar character prefixes to use the same prefix data and store only the tails as separate data. One character of the string is stored at each level of the tree, with the first character of the string stored at the root. Each node has an array of 27 pointers to its branches, one for each of the 26 alphabet characters and one for blank (“ ”). The following figure shows the example trie data structure**.**

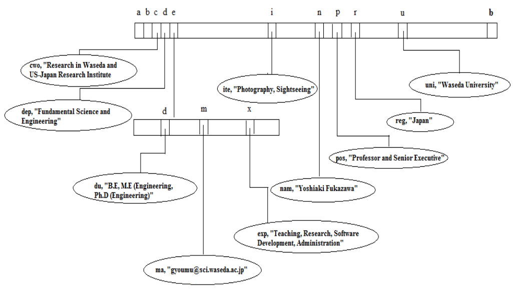

a b c d e
i
n p r
u
b
cwo, "Research in Waseda and US-Japan Research Institute
ite, "Photography, Sightseeing"
uni, "Waseda University"
dep, "Fundamental Science and Engineering"
d
m
x
reg, "Japan"
pos, "Professor and Senior Executive"
du, "B.E, M.E (Engineering, Ph.D (Engineering)"
nam, "Yoshiaki Fukazawa"
exp, "Teaching, Research, Software Development, Administration"
ma, "gyoumu@sci.waseda.ac.jp"

Figure.1: Example Tries

Original Tries algorithm have some limitations on space and time complexity. So, the system makes two improvements in original Tries algorithm:

1. Improving the searching speed of tries algorithm by developing effective key creation methodology.
2. Improving the spacing requirement of tries algorithm by using arithmetic coding approach which is optimal entropy encoding mechanism.

Tries is a good data structure technology and it is suitable for string manipulation and processing. Moreover, in tries, for inserting a word of length 'k' we need (k * 26) comparisons. By Applying the Big O notation it becomes O(k) which will be again O(1). Thus, insert operations are performed in constant time irrespective of the length of the input string. Same holds true for the search operation as well. The search operation exactly performs the way the insert does and its order is (k*26) = O(1), constant time.

4.6 Arithmetic Coding

Arithmetic coding represents a input symbol string as a small interval in [0, 1). It assigns one codeword to entire input stream. Reads input stream symbol by symbol, appending more bits to the code word each time. Code word is a number, representing a segmental subsection based on the symbols’ properties. It also encodes symbols using a non-integer number of bits which leads to very good results (entropy wise). Several steps are necessary for arithmetic coding process. These steps are:

(1) Counting Characters’ frequencies
(2) Constructing Range Table
(3) Encoding
(4) Decoding

Counting Characters’ Frequencies

In this step, we compute the frequency count of each character in the input string and these are stored in TreeMap in order to get the sorted character frequencies. The following shows the example String and the corresponding frequency result of the string.

Example:

Input String: “BILL GATES”

Map: {E=1, T=1, =1, G=1, A=1, S=1, B=1, L=2, I=1}

Range Table Construction

Range table construction is very important in arithmetic coding. All the encoding and decoding processes are depend on range table construction. It consists of three portions. The first column specifies relative frequency, the second column is the lower range and the third column is the upper range. The relative frequency (RF) is calculated as:

Relative Frequency , $RF_i = \frac{F_i}{L}$, i Map

where

F_i = frequencies of character i

L = input string length

The lower and upper range values are repeatedly calculated using the following equations till the end of Map is reached;

range = high - low ;
read character from Map

high = low + range×high_range(c) ;

low = low + range×low_range(c) ;

The following is the example range table values for the above input string:

{ =[0.1, 0.0, 0.1], A=[0.1, 0.1, 0.2], B=[0.1, 0.2, 0.3], E=[0.1, 0.3, 0.4], G=[0.1, 0.4, 0.5], I=[0.1, 0.5, 0.6], L=[0.2, 0.6, 0.8], S=[0.1, 0.8, 0.9], T=[0.1, 0.9, 1.0]}

Encoding

Encoding process accepts the input String and returns the encoded Double value of the corresponding result. As the string is encoded into a double value, it saves significant amount of memory. Firstly, encoding process converts the input string to a character array and using the range table values from Range Table and encodes the input string to a floating point value which only requires a certain amount of memory, so the memory usage of the input string is greatly reduced.

Decoding

Decoding process accepts the encoded double value and perform the decoding process. As the arithmetic coding approach is a lossless data compression approach, it will return the original input string without losing any data. Moreover, it only gives the encoded value of specified length regardless of the input length. So, the memory usage of any string length is reduced significantly.

5. Proposed data integration system (DIS)

Data Retrieval System for Relational and NoSQL databases are very important in today's information technology environment. DIS is intended to solve the problem of handling both types of SQL and NoSQL data in current technological area. Users need not know which part of data resides in relational (SQL) database and which part in non relational (NoSQL) database providing data transparency. There are three layers in HQPS:

(1) Data Source Layer

(2) Combine and Extract Layer

(3) Presentation Layer

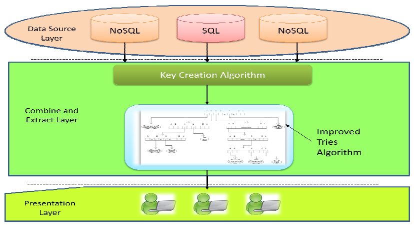
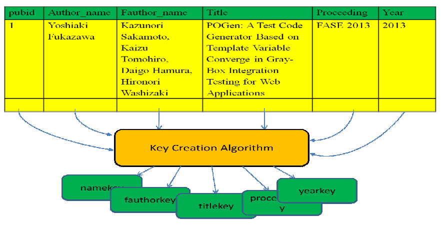


Figure 2: System Overview

In Data Source Layer, data are stored in heterogeneous data sources i.e., SQL and NoSQL databases. When the user want to retrieve data from RNDIS, the user have to give an input query to the Combine and Extract Layer via the Presentation Layer. The Combine and Extract Layer performs necessary operations concerning with the heterogeneous data retrieval process and the integrated results will be given to the users via the Presentation Layer.

5.1 Key Creation Process in Proposed Data Integration System (DIS)

In this process, keys for relational data and NoSQL data are created in order to store the data in tries structure. Metadata information from relational and NoSQL databases are used for key creation. Key creation for NoSQL data is also performed using the column information of JSON object.



pubid	Author_name	Fauthor_name	Title	Proceeding	Year
1	Yoshiaki Fukazawa	Kazunori Sakamoto, Kaizu Tomohiro, Daigo Hamura, Hironori Washizaki	POGen: A Test Code Generator Based on Template Variable Converge in Gray-Box Integration Testing for Web Applications	FASE 2013	2013

Figure 3: Key Creation for Relational Data

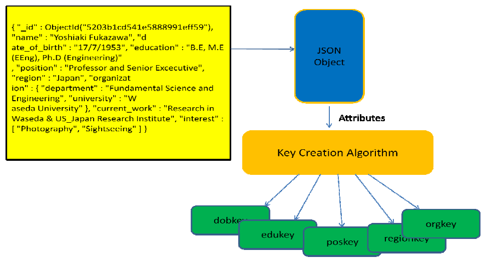

```
{ "_id" : ObjectId("5203b1cd541e5888991eff59"),
"name" : "Yoshiaki Fukazawa", "d
ate_of_birth" : "17/7/1953", "education" : "B.E, M.E
(EEng), Ph.D (Engineering)"
, "position" : "Professor and Senior Excecutive",
"region" : "Japan", "organizat
ion" : { "department" : "Fundamental Science and
Engineering", "university" : "W
aseda University" }, "current_work" : "Research in
Waseda & US_Japan Research Institute", "interest" :
[ "Photography", "Sightseeing" ] }
```

Figure 4: Key Creation for NoSQL Data

5.2 Data Integration with Improved Tries Algorithm

```
Algorithm: IntegrationwithTriesandAE

Step 1 : Accept Input Query
Step 2 : Getting Connectivity to Relational and
         NoSQL Databases
Step 3: Generate and Execute SQL and NoSQL
         queries
Step 4: Insertion Relational Data to Improved Tries
         While (rs.next())
             1. Generate key for resultset data
             2. Encode the resultset data
             3. Put the key and coded value pair to the
                tries
Step 5: Insertion NoSQL data to Improved Tries
             1. Generate key for cursordoc object data
             2. Encode the cursordoc object data
             3. Put the key and coded value pair to the
                tries
Step 6: Retrieve and Return the integrated data from
         tries using their associated keys
```

5.3 Case Study Implementation

In case study implementation, heterogeneous data from relational database and NoSQL database are integrated with Improved Tries Data Structure. Application domain is specified to be academic domain. Biographic and publication data of some professors from 7 universities from Asia region are collected for heterogeneous data storage in this system. And MySQL database will be used as relational data storeage and MongoDB is used for NoSQL data storage. Data collection for the system from heterogeneous sources is shown in Figure 5.

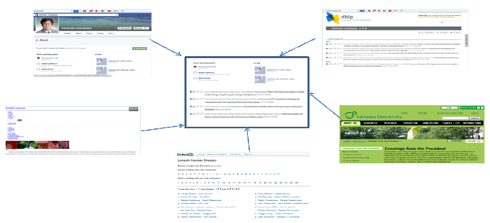


Figure 5: Data Collection Sources

MySQL

MySQL is available for free, is easy to install and hardly takes any disk space or computer memory at all. Moreover, it is a commercial grade database available for free and used throughout the internet. It locks the whole table whenever one user is trying to write to that table, not just the row(s) the user is trying to update or insert. It makes a great database which is mostly for reading, with sporadic writing. On the plus side, it is one of the fastest databases around, for the very reason that transactions, subqueries and row-level locking were intentionally left out.

In case study, publication data from MySQL are retrieved via the resultset which contain all the records concerning with the given query will be returned. Resultset data is read using rs.next() method and all data are inserted into Improved Tries Data Structure in order to continue data integration process.

MongoDB

MongoDB is a GPL open source document store written in C++ and supported by 10gen. It has some similarities to CouchDB: it provides indexes on collections, it is lockless, and it provides a document query mechanism. It is designed with latest NoSQL Theory and it is cost effective. It can represent several relational tables in only one JSON document as shown in the following figure because JSON supports nested key value structure. The following figure shows the sample relational data from NoSQL .

In case study, biographic data are retrieved from NoSQL data store, especially MongoDB in this case, JSON object is returned and data are read from that object and continue data integration process with the help of improved tries. The following figure shows the data integration of relational and NoSQL databases using the improved tries structure.

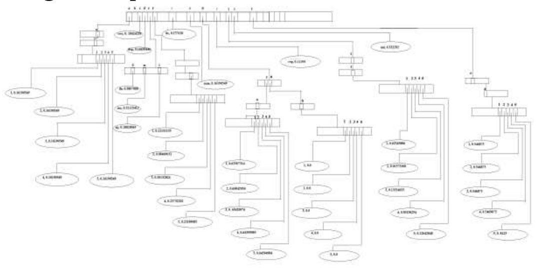


Figure 6: Data Integration using improved tries structure

5.4 Performance Evaluation

This section evaluates the processing memory usage of data integration of scenario 1 in different situations: (1) data integration with trie data structure and (2) data integration with improved trie data structure. The processing memory usage is calculated as follows:

1) Getting the total amount of memory that JVM is available:
   Runtime runtime = Runtime.getRuntime();
2) Do garbage Collection
   runtime.gc();
3) Getting how much memory is being used by your application:

runtime.totalMemory() - runtime.freeMemory();

And according to the evaluation result, we can clearly see that, data integration using improved tries has the desired effect. It reduces the amount of memory usage significantly.

Comparison of Memory Usage for scenario 1

Scenario 1:

Input : Yoshiaki Fukazawa

Output : 20 records from relational and 1 JSON object from NoSQL

Cases	Processing Memory Usage (bytes)	Processing Time (milli seconds)
With Trie	3836976	1617
With Trie and AE	3583296	2883

The figures for memory usage and processing time of scenario 1 are shown below.

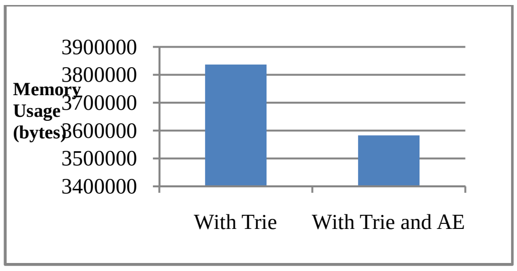


Figure 7: Processing Memory Usage for scenario 1

The bar chart shows the processing memory usage of scenario1 in different situations such as without trie, with trie and the proposed method (with trie and AE). We can clearly see that our approach has better memory usage than other situations.

Comparison of Processing Time for scenario 1

The following figure shows the processing time of scenario1. Because of the complexity of improvements of original Tries, this approach has tradeoff in processing time.

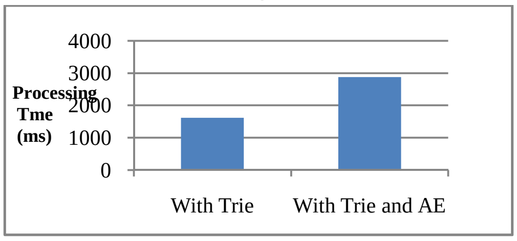


Figure 8: Processing Time for scenario 1

Comparison of Memory Usage for scenario 2

Scenario 2:

Input : Yasuo Matsuyama

Output: 22 records from relational and 1 JSON object from NoSQL

Cases	Processing Memory Usage (bytes)	Processing Time (milli seconds)
With Trie	3844128	1399
With Trie and AE	3425136	3484

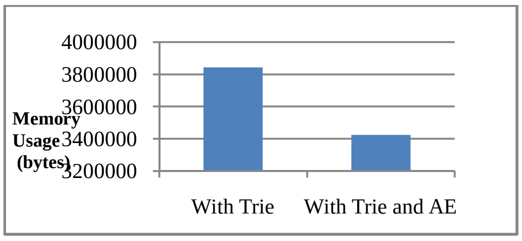


Figure 9: Processing Memory Usage for scenario 2

The bar chart shows the processing memory usage of scenario 2 in different situations. Like the scenario 1, this approach also has better memory usage than other situations.

Comparison of Processing Time for scenario 2

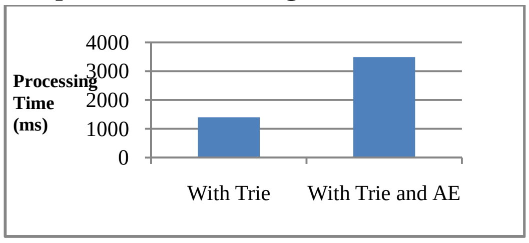


Figure 10: Processing Time for scenario 2

According to the results of the scenarios, the proposed approach has the better memory usage than other two situations but it has processing tradeoffs than with trie.

6. Conclusion

The reason why NoSQL has been so popular the last few years is mainly because relational databases don't scale out very well in a distributed system. However, NoSQL still have some limitations and they can't replace relational databases completely. So , there is a need to integrate data from heterogeneous databases and heterogeneous data handling between different data sources becomes a major challenging issue in distributed computing environment. This system implements hybrid query processing system for heterogeneous data sources using Improved Tries Data Structure and Arithmetic encoding mechanisms as a way of data integration between heterogeneous relational and NoSQL databases. And according to the evaluation results, the system reduces the memory usage significantly during data integration. However, it is also clear that there is running time tradeoff in the proposed data integration system because of the complexity of tries improvements.

References

[1] A.Bennett and C Bayrak "Bridging The Data Integration Gap: From Theory to Implementation May 2011.

[2] A.Ahuja, A.Kumar, R.Singh, "An Approach for Virtualization and Integration of Heterogeneous Cloud Databases", October 2012.

[3] D. Tomaszuk, "Document-oriented triplestore based on RDF/JSON" 2010K. Elissa, "Title of paper if known," unpublished.

[4] J. Roijackers, G. H. L. Fletcher, "Bridging the Gap between SQL and NoSQL"

[5] L. Ferreira, "Briding the gap between SQL and NoSQL" State of the Art Reports 2011.

[6] O.cure, r.Hecht, C. L. Duc, M.Lamolle, "Data Integration over NoSQL Stores using Access Path Based Mappings. 2010.

[7] O. Sutinen, "NoSQL – Factors Supporting the Adoption of Non-Relational Databases", December 2010.

[8] S. Heinz, J. Zobel, H.E. Williams, "Burst Tries: A Fast, Efficient Data Structure for String Keys", ACM Transactions on Information Systems, Volume 20, Issue 2, April 2002, pp 192-223

[9] T. Tanhermhong, T. Theeramunkong, W. Chinan "A Structure-Shared Trie Compression Method", *Proceeding* of SIGFIDET Workshop on Data Description, Access and Control ACM:207-217, 2001.