2013 International Conference on Advances in ICT for Emerging Regions (ICTer): 234 - 239

A Federated Approach on Heterogeneous NoSQL Data Stores

H. M. L. Dharmasiri [1] , M. D. J. S. Goonetillake [2]
University of ColomboSchool of Computing No 35, Reid Avenue, Colombo 7, Sri Lanka
[1]modithadharmasiri@gmail.com
[2]jsg@ucsc.cmb.ac.lk

***Abstract*—NoSQL has now become a topic of major interest. It addresses the problems of scalability and performance of the traditional RDBMS (Relational Database Management System) with high volumes of data.**

Today there are over hundred NoSQL implementations mainly focused around four data models with a lot of heterogeneity between them. Therefore integrating several NoSQL data stores to work together in situations such as changing the underlying data store becomes a hectic task and migrating the data from one system to another is also not feasible due to the high volumes of data involved. The objective of this research is to address the heterogeneity of NoSQL data stores through database federation which has been around since the early 80's which has proven to be successful in integrating heterogeneous data storages.

In this research we have successfully implemented a NoSQL federation with Cassandra, MongoDB and CouchDB proving that NoSQL federation is feasible with a certain degree of overhead.

***Index Terms*—NoSQL, Data store, Database Federation**

I. Introduction

DATA storage plays a major role in any kind of a computing application regardless of its content. Database Management Systems (DBMS) have been there for decades and have evolved a lot in becoming the highly complicated and effective manner of data storage today.

In today's world, Relational Database Management Systems (RDBMS) have become the pinnacle of data storage systems with extensive development and improvements throughout the years, with SQL becoming the de-facto standard. The industry, and the hardware and software capabilities grew with time and the need for larger volumes of data storage became a necessity. Databases are not at the gigabyte level any more and large data stores have terabytes or even petabytes of data. Hence the term 'Big Data' and NoSQL came to being.

NoSQL which stands for Not only SQL is a broad set of data stores which differ from traditional relational database management systems in many aspects. Classical RDBMS require fixed table structures which is not necessary in NoSQL. They avoid costly operations such as joins.

There are more than hundred implementations of NoSQL data stores as of now [1]. These systems create a vast heterogencity among them. Due to this reason working with several NoSQL systems become difficult [2]. Corporate mergers are common now and merging parties want to benefit from their data. Changing from one system to another is also costly as there are large volumes of data involved. Changing the complete application to cater the next NoSQL system is not that feasible. As more and more solutions move towards NoSQL these issues become more common.

In this paper we have successfully applied an already existing concept of RDBMS specially designed to address the heterogeneity issues, to NOSQL which is federation.

II. Related Work

A. Database Federation

By early 80s uses of computer systems were a common task in the corporate world. Almost all the systems were independent from each other even inside the same business. For an example employee's HR records were handled separately from their salary details. This turned out to be really inefficient and a waste of time and resources as well as there were no consistency among the systems.

Therefore Database federation was suggested and first discussed by McLeod and Heimbigner as one which "define[s] the architecture and interconnect[s] databases that minimize central authority yet support partial sharing and coordination among database systems"[3], collection of components to unite loosely coupled federation in order to share and exchange information" and "an organization model based on equal, autonomous databases, with sharing controlled by explicit interfaces."[3]

A federated database management system can be considered as a virtual DBMS. With the help of data abstraction it enables users and applications to store and retrieve data from several non-contiguous databases with only a single query from its uniform interface. Federation also provides data distribution and some scalability as well.[4]

The databases in the federation supports both local and federation operations. Therefore the existing applications that already use the component databases are not affected. The federated operations involve global operations which manipulate several component DBs. The coupling of the FDBMS whether tightly or loosely depends on the level of integration of the component DBs[3]. Eventhough you can have multiple federated schemas, it is advisable to have a single federated schema as it helps to maintain the uniformity in interpretation of the integrated data.

B. NoSQL

NoSQL is said to have the meaning "Not only SQL" which represents alternate storage systems from the traditional

978-1-4799-1276-6/13/$31.00 ©2013 IEEE

RDBMSs,it is more like an umbrella term used for data stores that don't follow traditional RDBMS principles.

NoSQL systems are normally used for large sets of data which is accessed and manipulated in a web scale. Most of these systems came from the industry perspective rather than the research community. Google has been a pioneer in researching these alternative storage systems in order to handle the massive amounts of data involved in their applications . They have used them for some time and then published them in a series of papers.[5][6][7]. Amazon released their own alternate data storage solution "Dynamo" which they use in their environment[8].

1) NoSQL Heterogeneity: At the time of writing there are 100+ NoSQL implementations. [9] These can be categorized in the way they store and manipulate data. This classification of NoSQL Systems can be seen in Figure 1.

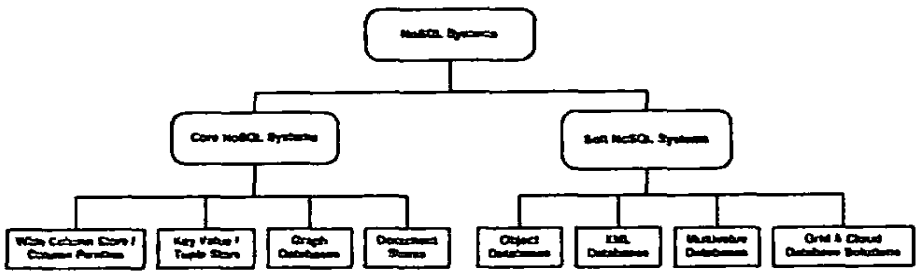

Fig. 1. NoSQL Hetogeneiety

Apart from the classification mentioned in Figure 1, different implementations of the data models create a vast and complex heterogeneity among NoSQL solutions.

III. Design

There are certain aspects that need to be considered from NoSQL point of view regarding database federation.

A. Heterogeneity

The main reason behind implementing a database federation is to cater the heterogeneity among the component databases.

- Different data models are being used by different NoSQL data-stores.
- Even with the same data model different implementations have variations.
- Query language differences
- Different features of CAP theorem being supported.
- Different consistency models.

In our federation we have addressed different data models, different implementations and query language differences.

B. Level of federation

We implemented a tightly coupled single federation solution because a tightly coupled federation is easy to handle as there isn't much user involvement in managing the federation. If it was loosely coupled there could be problems in data integrity. The rules for the tightly coupled federation is predefined, therefore the end user modifications will not affect the integrity. Single federation is the initial step. So it is safe to assume that it is possible to have multiple federations if we manage to have a single federation.

C. Schemas

The implemented NoSQL Federation is built considering different NoSQL data-stores having the same schema. This is done as an initial step trying to have a federation with all the data stores having the same schema. This can be used in a situation such as moving from one data store to another. The lessons learnt through this will help to identify how it can be further enhanced to support different schemas.

As there is a particular schema that we use in the federation there won't be any schema conflicts, but naming, domain, precision, meta-data and data conflicts can arise. To overcome this issue we are introducing a data dictionary or a set of meta-data about the data-stores as well as the data stored within the database. It is a one time configuration which needs to be done at the beginning.

Figure 2 illustrates the overall design architecture and the components of the federated NoSQL system.

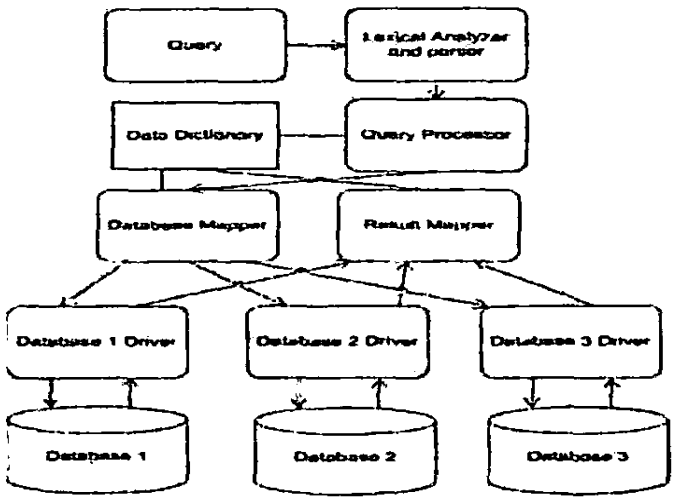

Fig. 2. Architecture of a Federated NoSQL System

As shown in Figure 2 the federated NoSQL system consists of 8 main components and each of the components play a major role in supporting the federated architecture.

D. Query

In the traditional relational database model, SQL has become the standard query interface for the database management systems with minor variations among the different vendor implementations.

The NoSQL databases on the other hand do not have a standard query interface. Different data models require different query interfaces. Even in the same model there are differences between the query interface these range from SQL like query language, REST API to a completely new query language specific to that particular NoSQL Database management system.

In our federated solution we are addressing this issue by implementing a common query interface to access the federated database solution. Introducing a completely new query language is not feasible as the users will not be interested

in learning a new query interface. Therefore we decided to have a query interface as a modification to the standard SQL(a subset of SQL). By having a SQL like query language the users will feel more comfortable in using it rather than getting to know different syntaxes in order to access different NoSQL database management systems. There are some modifications to the standard SQL but it is not of a major scale, reducing the time to get familiarized with the system and accessing several databases at once with ease. In the initial stage we have implemented three major functionalities in the federated system.

- **Insert**:Insertion is the main part of a database management system whether it is federated or not. The insertion will happen only to a single database management system which is part of the federation depending on query.
- **Delete**:Deleting a certain entry from a database is as important as inserting it. The system will delete the specified element from the relevant database.
- **Search** :Searching or querying the database with desired parameters is a major functionality of a database management system. These would be simple queries supported by the NoSQL databases and once the query is made, the federated system will get the results from all the different NoSQL databases participating in the federation
- **Lexical Analyzer and Parser**: In order to process the queries they need to be analyzed and parsed. Each of the query strings have to go through a lexical analyzer which identifies the string tokens and a stream of tokens is created. These tokens are identified by the predefined grammar rules implemented using the regular expressions. The lexical analyzer ignores white-spaces and comments and sends the stream of tokens to the parser for further analysis if there are no errors. The parser then validates the query using the grammar rules, analyzes the syntactical correctness of the input query. After checking the correctness it generates the parse tree. Once the parsing is successful the query information is passed on to the query processor in order to continue the execution of the query.
- **Query Processor**: The query processor takes the output of the parser as the input and analyzes it and clearly distinguishes the query information. It identifies necessary information such as the type of query, databases being involved and search parameters. According to the information it creates a special data structure containing the details of the query. This data structure is then sent in to the database mapper which executes the query on the relevant database.
  The lexical analyzer and parser together with the query processor constructs a filtering processor and the transformation processor in federation context.
- **Database Mapper**: Once the query is identified the databases must be selected from the federation which gets involved in the query. In the case of an insertion the query itself would indicate to which database the data must be inserted into. After that the database mapper will map the user given query to the query in that database in order to communicate with that NoSQL data store. All the NoSQL data stores must participate in a select query. Therefore the database mapper should map the user given query into the relevant database query of that NoSQL data store in order to get the desired results. The deletion process a search needs to be done on all the databases to identify the data that needs to be deleted. After that the deletion process can be executed.
  Once the query mapping is done to the respective NoSQL data stores the generated queries are executed via the database drivers to get the result.
- **Database Driver**: Different NoSQL data stores use different methods, drivers and APIs to query and interact with the data store. Some of these solutions are based on persistent APIs which is not appropriate for a federated solution as it restricts the functionality to a pre-defined schema. Therefore in order to have a NoSQL data store to participate in a federation it needs to have an API or driver which allows the most flexible and interactive access to the underlying database. Most of the NoSQL data stores have several database drivers to them which most of them are done by 3rd parties. It is important to choose the best solution out of them in order to have a successful federation.
  The database mapper and the database driver work together as a construction processor for the federation.
- **Database**: The federating NoSQL data sources are the most important component of the system. There will be different NoSQL data stores with support to different data models and several different data stores supporting the same data model as well. We have chosen data stores having the same schema.
- **Result Mapper**: Different NoSQL data stores use could use different methods to output the query results in the federation results are finally mapped into JSON (Javascript Object Notation) as a common data format. The result mapper maps the end result consisting of data from all the data stores to JSON when giving query results.
- **Data Dictionary**: Due to the intense heterogeneity among the different data stores there should be a mechanism to compensate it. In our federated system we are introducing a data dictionary which keeps metadata to overcome this. With the help of the data dictionary naming, domain, precision, meta-data and data conflicts can be eliminated. The data dictionary is a XML file which needs to be created at the initial deployment of the federation. This contains the details about the schema that is being used in the federation. That schema is what the end users see.

IV. Implementation

As NoSQL federation haven't been explored thus far our main goal is to explore the possibility of having a NoSQL federation. We used 3 major NoSQL data-stores for the federation, namely CouchDB, MongoDB and Cassandra. The reason behind choosing these data-stores are

- They can be considered as matured NoSQL solutions

978-1-4799-1276-6/13/$31.00 ©2013 IEEE

available so far because although there are 100+ implementations most of them are at the infant level.

- Mostly being used by the industry and in lots of applications. Unlike some of the solutions where they are only used in a particular scenarios.
- Most of the NoSQL solutions are community based. Therefore proper documentation is not available in most of the systems. Having a good community support is essential in working with these NoSQL data-stores.
- Availability of stable client libraries.

Among these NoSQL data-stores CouchDB and MongoDB are document stores and Cassandra is a column family. Different libraries were used to access these data-stores. The problem behind this task is as most of the NoSQL systems are open-source, community based there are several libraries implemented to access these NoSQL systems. So it is very difficult to choose which one suits best for the task. Most of these access libraries are persistent APIs which does not suit our task. In those persistent APIs the database persistent classes needs to be implemented beforehand. This affects our federation's portability as each time the federation is deployed the classes need to be created with each of the schema. So what we needed was a raw data access libraries without the addition of persistence.

We used the default java client for MongoDB, couchdb4j for CouchDB and Astyanax for Cassandra as the client libraries to access the federation.

V. Evaluation

We have evaluated our solution under implementability, runtime, performance and usability.

As the test scenario we have created a sample schema (customers with id, first name, last name, age, phone number and email) with 1 000 000 records evenly distributed among the three data-stores that we use. We used a simple customer schema as the schema for the federation. We used 3 node clusters for each of the data stores and a single machine running the federation for our evaluation.

A. Implementability analysis

Our main objective is to discover the possibility of implementing a federated NoSQL solution. In our attempt we succeeded in having a federated environment with CouchDB, MongoDB and Cassandra. Key-value stores were not used as it does not support schemas and instead have kv pairs.

With our implemented NoSQL federated solution we have successfully managed to interact with all 3 NoSQL data-stores. The implementation supports the very basic facilities such as insertion and simple querying of the underlying data-stores. The objective of our federation is not to replace the original interface of the underlying data-stores, but to provide support in integrating results from several different ones.

After the execution of queries in the Listing 1 the federation successfully inserted and queried the data accordingly.

```
INSERT INTO customer.<datastore_name> (cust_id ,
    fname ,lname , age , phone , email)
    VALUES
    (123456,'John', 'De Silva', 27, 0782457892,
     aaaa@bcd.com);

SELECT * FROM customer where age >= 49 ;
```

Listing 1. Insert and search queries of the Federation

In order to add a NoSQL data store to the federation several factors need to be considered. It is important to have a generic access method to the underlying NoSQL data-store. Most of the NoSQL data-stores available at this time are designed for specific tasks. This requires the end user to have a persistent API which only allows you to manipulate that particular schema. If you are to change the schema you have to regenerate all the data model files according to the new schema. This is quite all right because there's hardly any need to change the schema of a data-store. Although we have also used a particular schema for the federation we allow it to be easily configurable through a single configuration file. Therefore using the federated solution in different scenarios only involve changing that configuration.

Another important factor that we come across is the availability of a reliable and stable client to access the database with the support to operations that we need on our federation. We encountered this problem with our CouchDB client. Initially we used JzBoy but soon figured out that it only support simple equal queries but not range queries that we intended to implement. After that we moved on to CouchDb4j which had the features that we wanted. Unfortunately that also gave performance issues when it came to large volumes of results. It is hard to find good client interface in most of the cases because most of the clients are community based and lack proper documentation. Therefore most of the time it involves trial and error with testing each of the clients deciding which suits you best. Working with a well established client library would take a lot of effort off the implementation process.

B. Runtime analysis

It is important to know the performance aspects of the implemented federation solution. We have created a 1 million entries of sample data and distributed them among the 3 NoSQL data-stores in the test environment. The running time is calculated for insert and search queries.

After having the federated system up and running with sample data we tested the run time taken for an insert query that goes through the federation. The factors need to be considered here is the amount of time taken for each of the insert queries to successfully execute through the federated system. We have tested the system with 50 insert queries for each of the data-stores with a total of 150 insertions and got the mean value for an insertion query for each of the data-stores. After that we compared the time taken through the federation and without the federation.

Figure 3 illustrates the time comparison of insert queries on each of the underlying NoSQL data stores. It is clearly visible that there is a slight increment of the runtime of the query when it is done through the federation. This is because the federation introduces an extra layer for the query interface which involves parsing the query and redirecting it

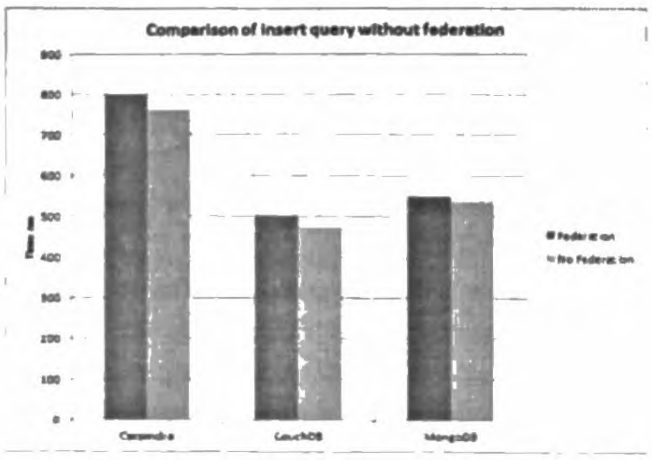

Fig. 3. Comparison of insert query with and without the Federaion

to the correct data-store. The extra time taken is negligible percentage of the overall time. The extra time is almost consistent irrespective of the underlying data store.

The next important feature we have implemented in the federated solution is the search operation. To test the performance of the federation we have tested it with different range queries which would result different number of search results. With the format of

```
SELECT * FROM customer where age > = X ;
```

Where the value of X was changed to get different number of results. Each query was executed 25 times and the mean value was taken which is shown in Figure 4.

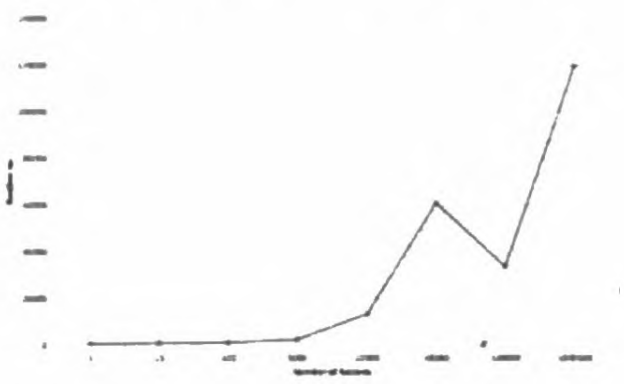

Fig. 4. Runtime for Search queries

As seen on Figure 4, the time taken to execute the result increases as the number of records returned by the query increase. There is a decrement of run time when the record size increases from 48280 to 100000. This is because the couchdb4j library which we used to access couchDB failed to give results when the number of results exceeded certain number. Therefore we had to use only MongoDB and Cassandra for the rest of the tests. Then the runtime decreased dramatically indicating that couchdb4j was the bottleneck of our federated solution. This shows the importance of having a good stable library to manipulate and query NoSQL datastores.

C. Performance analysis

It is important to compare our Federation system with other alternatives. Unfortunately there are no federation solutions available for NoSQL at the moment. Therefore we compared our Federation system against the approach of changing the end user application code to access the different NoSQL datastores. The time and effort taken in terms of lines of code are then compared between the two approaches.

We compared the lines of code involved in accessing all 3 underlying data stores within the same application with and without the federated approach in the end user's perspective. For this the number of lines are shown are Astynax library for Cassandra, couchdb4j for CouchDB and the MongoDB driver provided for MongoDB for java environment. Therefore these values of lines of code might change for different access methods and the running environment.

The federated system use significantly lower number of lines compared to other systems. The main reason for this is that when accessing the data-stores the configurations must also be set in the code. But in our federated system the configuration is one time at the initial set-up process. If there is any need for a change such as changing the schema, the simple change of the configuration file is sufficient and the change will be reflected on the federation immediately.

We then compared the runtime of different data-stores and the federation system to return a specific number of results for a range query. Figure 5 illustrates the results obtained.

It is clear that the runtime increases as the number of results retuned by the query increase described in Figure 5. It seems like MongoDB has better performance compared to the other systems. The federation has taken more time because it involves another layer to analyse the query as well as to map the end result into JSON, the common data model. The reason for CouchDB's poor performance could be the poor performance of couchdb4j library that we used. Cassandra has also not performed well compared to MongoDB. The reason behind this is the underlying indexing mechanism. As we are using a range query, the indexing mechanism of MongoDB allows it to quickly return the results. But Cassandra only have hash indexes and it does not have a b-tree index property in order to support range queries. This brings in another important factor, choosing the right NoSQL data store for the right job.

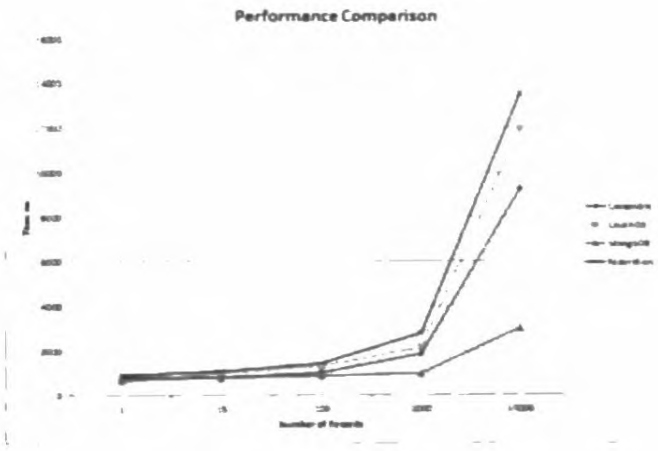

Fig. 5. Performance comparison

We observed the time taken by the different phases of the federation and came up with an equation for the federation's performance. Figure 6 sums up the federation time performance into an equation. Here the time taken for the lexical analyzer and parser to map the query T_{lex} and the query

978-1-4799-1276-6/13/$31.00 ©2013 IEEE

mappings (T_{Mapcas},$T_{Mapcouch}$,$T_{Mapmongo}$) considerably less amount of time compared to the others. The time taken for the queries (T_{Qcas},T_{Qcouch},T_{Qmongo}) depends on the type of query, number of results in the query. As we have observed the type of query has a effect on the time for the database to execute that particular query. The higher the number of records involved in the result and more time it takes to the data-store to query them. The communication overhead also depends on the number of records involved in the result the more the records the mote time it takes to deliver. The next important factor is the time taken for the result mapper to map the result in to the common JSON format. This solely depends on the number of records returned by each of the data stores and how long a single record takes to convert. We take the maximum time taken by the component DB as the data stores execute the queries in parallel. Therefore the end result depends on which data store takes the most time to deliver.

$$T = T_{lex} + MAX(T_{Cas} + T_{Casmap} - T_{Mongo})$$

$$T_{Cas} = T_{Mapcas} + T_{Qcas} + n_{Cas}T_{Casmap}$$

$$T_{Couch} = T_{Mapcouch} + T_{Qcouch} + n_{Couch}T_{Couchmap}$$

$$T_{Mongo} = T_{Mapmongo} + T_{Qmongo} + n_{Mongo}T_{Mongomap}$$

T = *Total Time taken to get the record*

T_{lex} = *Time taken for lexical analyzer to analyze the query*

T_{Cas} = *Time Taken by Cassandra for the query*

T_{Couch} = *Time Taken by CouchDB for the query*

T_{Mongo} = *Time Taken by MongoDB for the query*

T_{Mapcas} = *Time for mapping parse tree to Cassandra query*

T_{Qcas} = *Time taken for querying Cassandra*

n_{Cas} = *Number of records returned from Cassandra*

T_{Casmap} = *Time taken to convert a single Cassandra record to JSON*

Fig. 6. Time equation of the federated NoSQL system

VI. Conclusion

Database federation have been around since the early 80's there haven't been any exploration done in federating NoSQL data stores. So there are vast areas to be explored and improved. The main goal of this research was to examine the feasibility of having a federated NoSQL solution. This research has managed to achieve this task by implementing a federated system between MongoDB, CouchDB and Cassandra. The choice of the NoSQL systems contain two document stores and a column family.

It's not only the data model that affects when it comes to a federation of NoSQL. There are query model differences between different NoSQL implementations of the same data model. Furthermore the different consistency methods in different NoSQL data stores also have an effect on the federation. But in this research an assumption was made that the federation is used instead of accessing multiple data stores from the end user application itself. Therefore the consistency differences can be ignored.

To have support for the federation it is necessary to have a solid, stable NoSQL data stores which have proper client interface which allows to access the data store in a generic manner. This research introduces a SQL-like query interface to the federation. This creates a much more user-friendly interaction for the users who are familiar with the popular SQL syntax. The federation must hide the different data models that the underlying component data stores use. The implemented solution uses JSON as the common data model mainly due to the wide use of JSON at the moment.

When it come to the performance aspect of the federation, the introduction of the federation creates an overhead due to query mapping and mainly data conversion. This becomes a considerable fact when the output result size increases. Therefore it's a choice between performance and the ease of use. If performance prevails the federation can be used as a temporary solution which can be deployed easily.

As final words this research has managed to successfully implement a federated NoSQL system with three different NoSQL solutions. With the tests and evaluation it can be concluded that NoSQL federation is feasible and it will take off a lot of time and effort taken in large scale data migrations. But you have to choose carefully on which component data stores to use as these choices have a great impact on the end performance which is quite essential in NoSQL systems.

VII. Future Work

We have successfully implemented a federation platform supporting basic features in our research. There are lot of improvements as well as additions to this from here onwards.

We have used existing client libraries to access the NoSQL component DBs. Accessing the DBs directly would remove the client overhead as well as overcome the lack of features of different clients. Since we have only used a single schema in our federation our next step would be introducing different schemas to this federation and manipulating them.

References

[1] R. Cattell, "Scalable SQL and NoSQL data stores," *ACM SIGMOD Record*, vol. 39, no. 4, pp. 12–27, 2011.

[2] M. Stonebraker, "Sql databases v. nosql databases," *Communications of the ACM*, vol. 53, no. 4, pp. 10–11, 2010.

[3] D. McLeod and D. Heimbigner, "A federated architecture for database systems," *Proceedings of the May 19-22, 1980, national computer conference on - AFIPS '80*, p. 283, 1980.

[4] A. P. Sheth and J. A. Larson, "Federated database systems for managing distributed, heterogeneous, and autonomous databases," *ACM Computing Surveys*, vol. 22, no. 3, pp. 183–236, Sep. 1990.

[5] S. Ghemawat, H. Gobioff, and S. Leung, "The google file system," in *ACM SIGOPS Operating Systems Review*, vol. 37, no. 5. ACM, 2003, pp. 29–43.

[6] J. Dean and S. Ghemawat, "Mapreduce: Simplified data processing on large clusters," *Communications of the ACM*, vol. 51, no. 1, pp. 107–113, 2008.

[7] F. Chang, J. Dean, S. Ghemawat, W. Hsieh, D. Wallach, M. Burrows, T. Chandra, A. Fikes, and R. Gruber, "Bigtable: A distributed storage system for structured data," *ACM Transactions on Computer Systems (TOCS)*, vol. 26, no. 2, p. 4, 2008.

[8] G. DeCandia, D. Hastorun, M. Jampani, G. Kakulapati, A. Lakshman, A. Pilchin, S. Sivasubramanian, P. Vosshall, and W. Vogels, "Dynamo: amazon's highly available key-value store," *SIGOPS Oper. Syst. Rev.*, vol. 41, pp. 205–220, Oct. 2007.

[9] P. S. Edlich, "Nosql databases," 2010. [Online]. Available: http://nosql-database.org/