Design Patterns to Enable Data Portability between Clouds' Databases

Mahdi Negahi Shirazi
Faculty of Information Technology
Multimedia University
Kualalumpur, Malaysia
negahi@live.com

Ho Chin Kuan
Faculty of Information Technology
Multimedia University
Kualalumpur, Malaysia
ckho@mmu.edu.my

Hossein Dolatabadi
Faculty of Information Technology
Multimedia University
Kualalumpur, Malaysia
Hossein_dolat@yahoo.com

Abstract— **Cloud providers offer their infrastructure, platform and software to customers who found cloud computing as an appropriate solution for their requirements. The companies who want to choose a cloud provider should be noted that after developing their own software in a specific cloud environment, the migration to another cloud environment would be difficult or it might be impossible. Cloud databases are responsible for storing data in a scalable and high available way in cloud environments. This paper provides a solution for enabling portability between column family databases and graph databases as cloud databases by proposing design patterns.**

Keywords- Cloud Computing, Data Portability, NoSQL, Column Family Databases, Graph Databases, Design pattern.

I. INTRODUCTION

Cloud computing is a most recent paradigm in computer science. Enterprises and developers desire to adopt existing software to cloud computing. The development of cloud computing software benefits developers in the way of paying attention to software instead of hardware aspects of deploying software. Cloud computing provides computing anytime and anywhere; in addition, it also omits borders of computing. Users only need an internet connection to access to any cloud computing providers' data centers. The cloud computing is rooted in the advantages of several different technologies including virtualization and multi-core chips which fall under hardware category, Web services, SOA, Web 2.0 as Internet technologies, utility and grid computing as a part of distributed computing, and automatic system managements and data center automations as a part of system managements[1].

Some unique characteristics of cloud computing such as fault-tolerance, high availability and scalability lead to invent the special data management system in cloud computing. For example, BigTable was developed by Google, Cassandra and Hive was developed by Facebook and in the same way SQL Azure was developed by Microsoft. Cloud data management systems offer comprehensive and complete solution that are suitable for cloud's characteristics like scale-out models; in addition, they transparent scale-out resources from applications. Different data management systems have different goals, for example, BigTable and Hbase are usually applicable in analytical applications; in contrast, Cassandra is used for Web management systems[2].

Different cloud vendors based on their strategies introduce different data models in cloud computing environments. Basically there are two paradigms in data modeling of cloud computing. The first one in NoSQL and the second one is a relational database. The first one in NoSQL and the second one is relational database.

NoSQL does not refer to "Never SQL or No to SQL". It means "Not Only SQL". NoSQL is not a specific piece of software, but rather a philosophy. NoSQL is just the concept of using databases other than SQL databases to store your data, and more specifically, databases other than relational databases [3]. The lack of ability of a horizontal scale in traditional database management systems is a major problem. This problem magnifies when traditional database management systems deal with massive data and distributed systems [3, 4].If a SQL database is suffered from the lack of performance in a distributed system, the problem will not be eliminated by just adding a machine to the system. In general, the problem will be removed by replacing a high capacity computer with an old machine for handling a large database. Furthermore, since the elimination of problem needs large computers, the solution does not fit well for the cloud computing architecture. For overcoming this issue NoSQL is applied, which has four categories in NoSQL paradigm. Key-value pair store, column family databases, document-oriented databases and graph databases are four category of NoSQL paradigm.

Portability in a cloud refers to provide an ability to move data and applications from one cloud provider to another. There are many reasons that a specific cloud user decides to migrate to a different cloud. Dissatisfaction of service, better alternative, change business or technology strategy, low cost and failure of provider are the most important reasons that a user migrate to new cloud provider.

A design pattern describes the core of a solution. Solutions are reusable a million times. Each design pattern focuses on a specific problem and it introduces a comprehensive solution for it. Nowadays, software designers use design patterns to improve their quality of attribute such as performance, usability, security, and also; they cause to write code clearly [5].

This paper is organized as follows: Section 2 describes our background study. Section 3 highlights the motivation of the problem to be solved in this project while the proposed solution is presented in Section 4. Finally, we conclude this paper in Section 5.

978-0-7695-4710-7/12 $26.00 © 2012 IEEE
DOI 10.1109/ICCSA.2012.29

II. Related Work

One of techniques that it enables portability is migration data. Migrate in and migrate out can be enable through replication. Using different cloud providers' virtualizations technology is one way to achieve data migration through replication [6].

Another technique to enable portability is cloud brokers. A Cloud broker provides an abstract service for customers by integrating and federating other cloud providers' resources. This integration causes make resources transparent for customers. Cloud customers deal with only with the broker providers [6]. Figure 1 shows a simple model of cloud broker.

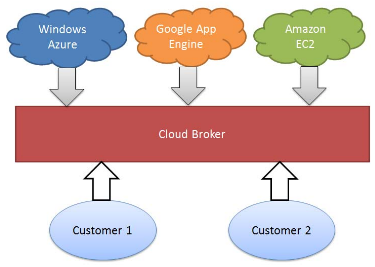

Figure 1. Cloud Broker

Another way that provides cloud portability is an abstract layer that applications are implemented based on it. This abstraction layer provides a comprehensive and complete way with a namespace for programmers to support blobs, table and queue. Moreover because of metadata that is added to data for recognizing the operation and target of data in some case the performance of an operation decrease [7].

Design Patterns provide some advantages for designing SQL queries. The structured Query Language is a declarative programming language whose aim is to manipulating and selecting data in relational databases. Although the SQL query might be very short and simple, it also might be very complex. Hence, design patterns can provide for SQL programming a proven way to how create understandable and reusable SQL queries. The complexity of a SQL query is directly related to relational database schemas. Some complex SQL queries might be difficult to understand. Therefore, the design patterns provide a way to understand these types of SQL queries [8].

Another benefit of design pattern in the database level is contributing to design a suitable database schema. The database schema is a major part of a software design that acts as a platform for an application. The goal of using design pattern in designing tables' schema is to provide an alternative solution for problems. Patterns should address fundamental concerns such as data integrity, query efficiency and evolution. There are several problems in designing tables' schemas. The first design pattern is pivot. The main motivation for this problem is to have huge attribute. Pivoting manages the attribute-value pairs. A solution for dealing with pivoting problem is classified attributes into two categories. The first category is stable. It refers to those attributes that are not frequently modified at schema level. The second one is evolvable that refers to attributes in which changes are predicable. The next solution is generalization and specialization for relational databases. In an object oriented paradigm, by using generalization a set of classes are abstracted in a high level class that it is called parent or supper class; in addition, specification is a process that is the opposite of generalization. The modeling structural inheritance is the goal of generalization and specialization process. The next solution is materialization. In object-oriented world materialization is a relationship between an abstract class and a set of concrete objects of it. In database level the usage of this concept is separating of commonly repeated information as opposed information [9].

III. Motivation

Nowadays, cloud computing is most attractive solution for every company. Cloud computing provides wide range resources for developing application. Instead of focusing on lack of hardware resources, performance, security and update fundamental software like operating system, developers just focus on what they should be designed and implemented.

One of main issue in cloud computing area is data portability between different clouds. Stuck in a specific cloud provider platform is a common problem to developing software for cloud computing. In some reason companies might change their cloud providers or they want to use different cloud simultaneously. Famous cloud providers use different storage architecture; consequently, portability needs huge effort.

NoSQL is a new paradigm in computer science; hence, developers and enterprises previously less pay attention to it. On the other hand, NoSQL databases become more popular since cloud computing was invented because they can scale out easily. Nowadays Google, Twitter and Facebook use the column family database because of handling the huge size of data is sufficiently possible. Moreover, cloud providers support column family databases like Google App Engine. According Neo4j blog, the open source graph database now supported by Microsoft Windows Azure.

IV. Design Patterns

The aim of design patterns is transfer data from column family to graph database. The data portability design pattern can be achieved by providing a way to transfer data from Neo4j to HBase. The proposed design pattern for transferring data from HBase to Neo4j must consider the column family concept. Furthermore, there are several issues in moving data in column family databases to graph databases.

- Column family: there are two possible column family types. The first option is that the column in a specific column family refers to another row. It means they act like a foreign key in the relational database. The second option is that the column gives more information about the row.
- Timestamp: by using timestamp, there are several version of a specific column in a row.

For migrating data from HBase as a column family database to Neorj as a graph database after selecting the table and filtering desired records, the following pattern should be performed:

1. Create a root node (RNode).
2. If a row key node (RKNode) that its value of "RowKey" property is equal to the current row key go to step 5.
3. Create RKNode that it is connected to RNode by an edge, which is named "Root". Then, add a property whose name is "RowKey" and put the current row key on it as the value.
4. Check the column family nodes (CFNoe) for the RKNode is not available. If it is available skip the rest of steps for the current row.
5. For every column family, add a new node (CFNode).
6. Link CFNode to the RKNode. Then add property for the relationship that it connects RKNode to CFNode with the following property: <CF:name_of_column family>
7. Check all columns' name in a column family. If a column's name conform the following pattern: <Column Family Name_ Property Name: Value> Add following property to corresponding edge based on column family's name. < Property Name: Value>
8. For every column in the column family add a node (CNode). The CFNode is connected to the CNode by an edge with following property: < C: column's name >
9. For every version of value for a column create a node (VNode). Link them to the CNode. The relationship's property is based on following pattern: <Timestamp: value>.
10. Examine the value of column. In this situation there are some possibilities that they mention in following. First, if the value of column is just a pure data, so the following property adds on VNode: <Value: Column_Value>. The next possibility is the value refers to a row key in other table or current table. For this situation the existence of RKNode for corresponding row key value should be evaluated. If it is available, recognize the start node of the incoming relationship. If the start node is RNode remove the edge and link the RKNode to the VNode. If the start node is not the RNode, link VNode to RNode If the RKNode is not available follow only step 3 for the row that its row key is on the column. Then, link the new node to the VNode. The VNode property in this possibility follow this pattern based on the value of the column refers to current table or other table: <Refer:CurrentTable> or <Refer: NewTable>.
11. For those RKNode that is connected to VNode repeat from step 4.

For migrating data form Neo4j as a graph database to HBase as a column family database. The following pattern should be travelled.

1. Create a record.
2. Generate a list that contains all nodes that their position is exactly after the root node (first-level nodes).
3. Put the first-level node's unique ID as a row key. If a first-level node does not have unique ID, the application should generate it.
4. If there are more properties in a first-level node, put node's properties in a column family with an optional name.
5. All nodes that are related to a first-level node will be categorized into two lists. The first list refers to those nodes that they have one or more common properties with a first-level node. The second list refers to nodes that there is no common attribute.
6. Each node in the **first list** refers to a new record in the current table. Create a column family based on incoming edge's name. Then, create a column that is conformed to following pattern: <Column Family Name_ Counter: Key of New Record>.
7. For each node in the **second list** outgoing edges have to be examined. There are two options based on the property on the outgoing edges:
   (A): the key of outgoing edge **is "Column"**, so the value of incoming edge is column family and the value of "Column" property is a column in the column family. The next node is recognized and the property of outgoing edge is timestamp of the column. The value of the current node's property should be examined. If the key of the node property **is "Value"**, the value of the property is the content of current column. In addition, if the key of the node property is "Refer", the value of the column is assigned based on the property's key in the next node.
   (B): The key of an outgoing edge **is not "Column"**. Thus, the depth of sub graphs that are connected to the current node must be calculated. Based on the value of the depth choose one the following possibilities:
   (B1): Depth = 1. The next node is a new record in the current table. In the current table a column family will be created base on an outgoing edge's name; in addition, the column must be generated base on outgoing edge's name and attach the number to it (Edge's name_Number). Then, repeat step 1 for next node and put its row key value as the value of the current column.
   (B2): Depth > 1: All edges should be examined. Compare all edges' properties. For all nodes that their edges' properties are alike act based on "7B1". The sub graph that the edges' properties are different must be assumed as a new table. For this case the following

steps should be performed: first a new table should be generated based on outgoing edge property. Then, repeat from step 1 for the sub graph.

8. If an edge has two properties, the one that it is key-value pair is added as a column in the current column family. The column name should be conformed following pattern: <Column Family Name_ Property Name>.

For evaluating two design patterns we will introduce a simple suitable scenario. The graph database really needs a unique data style. Hence, based on Health Infoscape [10] a HBase table schema is designed. MIT Senseable Cities and General Electric create a new method of understanding that it represents human diseases by analyzing 7.2 million of electronic medical report that its name is Health Infoscape. Figure 2 indicates the HealthTablet table. Because each disease relates to a number of conditions and other diseases, the number of columns in "Cause" column family would be changeable. It is significantly considerable that the presentation of data in a column family database as shown blow is totally different from the way that data are stored in real column family databases. Figure 3 indicates the corresponding graph that is generated based on design pattern.

The power of graph databases reveal when data are high interconnectivity. The complexity of data is the main point of using graph databases rather than the size of data. Modeling simple and complex domains is ability of graph databases by powerful abstraction. Hence, graph databases help to model data in special case; in addition, they facilitate analyze data. On the other hand, column family databases provide a smooth way for maintaining data in huge size. Moreover, they are weak in modeling complex data and analyzing data; thus, migrating to new style of data is a suitable solution for getting more information about a specific data. In result, the style of data is most significant considerable to use proposed patterns for migrating data. For applying these patterns to existing data, it is necessary to examine the style of data. If existing data are in a column family database, the relationship among of records must be considered.

V. Conclusion

Portability is the most noticeable issue in cloud computing since providers introduce own infrastructure. Data migration between different clouds is a technique for providing data portability in cloud. This paper provides a formal way for migrating data between HBase as a column family database to Neo4j as graph database. HBase database does not support foreign key, but in some tables' design the designer use a column as a foreign key. Thus, design pattern that transfers data from HBase to Neo4j needs to know these columns. Column family databases are the best solution for storing a very huge amount of data. The capacity of graph databases do not equal of column family databases. In result for migrating data form HBase to Neo4j the amount of data is an issue. For future work clustering algorithm will be applied to second pattern in order to reach the best result because in graph databases relationships and nodes have many properties, so these types of data must be store in a column family or a column. For gaining the best result these data must be categorized in a suitable column family or column.

Row key	Info		Prevalence			Causes		
	Name	Category	Female	Male	Total	Cause1	Cause2	Cause3
d1	*Heartburn*	*Digestive system*	*9.4%*	*9%*	*9.2%*	*d2*		
	1	*1*	*1*	*1*	*1*	*2*		
d2	*Chest Pain*	*Circulatory System*	*6.8%*	*6.8%*	*6.8%*			
	3	*3*	*3*	*3*	*3*			
d4	*Dizziness*	*Nervous System*	*4%*	*2.8%*	*3.5%*			
	5	*5*	*5*	*5*	*5*			

Figure 2. HealthTable in HBase

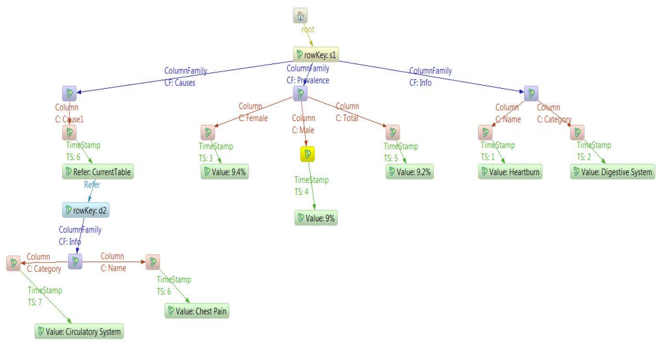

Figure 3. HealthGraph Bases on HealthTable

References

[1] R. Buyya, J. Broberg, and A. G. s. nski, Cloud computing Principles and Paradigms: Wiley Online Library, 2011.

[2] Y. Shi, X. Meng, J. Zhao, X. Hu, B. Liu, and H. Wang, "Benchmarking cloud-based data management systems," 2010, pp. 47-54.

[3] S. Tiwari, Professional NoSQL: Wrox, 2011.

[4] P. Warden, Big Data Glossary: O'Reilly Media, 2011.

[5] M. N. Shirazi, M. Hejazi, and H. Dolatabadi, "Applied Dynamic Chain of Responsibility in Web Application Security," in International Conference on Computer Science and Information Technology, 2011, pp. 279-282, to be published.

[6] IBM. (2010, 30-Oct). Portability In The Cloud. Available: http://www.slideshare.net/bharathshares/portability-in-the-cloud

[7] Z. Hill and M. Humphrey, "CSAL: A Cloud Storage Abstraction Layer to Enable Portable Cloud Applications," in Cloud Computing Technology and Science (CloudCom), 2010 IEEE Second International Conference on, 2010, pp. 504-511.

[8] V. Tropashko and D. Burleson, SQL Design Patterns: Expert Guide to SQL Programming: Rampant TechPress, 2007.

[9] E. Stathopoulou and P. Vassiliadis, "Design Patterns for Relational Databases," ed: Citeseer, 2009.

[10] Carlo Ratti, Eric Baczuk, Dominik Dahlem, Xiaoji Chen, Camille Kubie Manager, and A. Atkinson. (2011, 12-Dec). Health Infoscape. Available: http://senseable.mit.edu/healthinfoscape/

[11] E. Gamma, R. Helm, R. Johnson, and J. Vlissides, "Design patterns: Abstraction and reuse of object-oriented design," ECOOP'93—Object-Oriented Programming, pp. 406-431, 1993.