International Journal of Scientific Research in Computer Science, Engineering and Information Technology
© 2017 IJSRCSEIT | Volume 2 | Issue 5 | ISSN : 2456-3307

Implications of NoSQL Transaction Model in Cloud Database System

Ashok Kumar P S , Md Ateeq Ur Rahman
Department of CSE, JNTU/ SCET, Hyderabad, Andra Pradesh, India

ABSTRACT

NoSQL cloud database systems are new types of databases that are built across thousands of cloud nodes and are capable of storing and processing Big Data. NoSQL systems have been used in large scale applications that need high availability and efficiency. Consequently, such systems lack support for standard transactions which provide stronger consistency. This Paper proposes a new transactional model which provides NoSQL systems with standard transaction support and strong level of data consistency. The strategy is to supplement current NoSQL architecture with an extra layer that manages transactions. The proposed model is configurable where consistency, availability and efficiency can be adjusted based on application requirements. The Preliminary experiments show that it ensures stronger consistency and maintains good performance.

Keywords : Big Data, NoSQL, ACID, BASE, Couchdb, Mapreduce

I. INTRODUCTION

The concept of Big Data has led to an introduction of a new set of databases used in the cloud computing environment, that deviate from the characteristics of standard databases. The design of these new databases embraces new features and techniques that support parallel processing and replication of data. Data are distributed across multiple nodes and each node is responsible for processing queries directed to its subset of data. Each subset of data managed by a node is called shard. This technique of data storage and processing using multiple nodes improve performance and availability.

The architecture of these new systems, also known as NoSQL (Not Only SQL) databases, is designed to scale across multiple systems. In contrast to traditional relational databases which is built on sound mathematical model, NoSQL databases are designed to solve the problem of Big Data which is characterized by 3Vs (Volume, Variety, Velocity) or 4Vs (Volume, Variety, Velocity, and Value) model. As such, NoSQL systems do not follow standard models or design principles in processing Big Data. Different vendors provide proprietary implementation of NoSQL systems such that they meet their (business) needs. For instance, unlike traditional relational database systems which rely heavily on normalization and referential integrity, NoSQL systems incorporate little or no normalization in the data management.

The primary objective of NoSQL systems is to ensure high efficiency, availability and scalability in storing and processing Big Data. NoSQL systems do not ensure stronger consistency and integrity of data. They therefore do not implement ACID (Atomicity, Consistency, Isolation, Durability) transactions. However, it is important to provide stronger consistency and integrity of data while maintaining appropriate levels of efficiency, availability and scalability.

Table 1. Comparison between SQL and NoSQL

ACID (RDBMS)	BASE (NoSQL)
strong consistency	weak consistency
isolation	last write wins
transaction	program managed
scale-up (limited)	scale-out (unlimited)
robust database	simple database
shared-something (disk, memory, process)	shared-nothing (parallelizable)

II. LITERATURE REVIEW

Various approaches have been proposed to address transaction management in NoSQL databases. However, because of the diverse flavors and kinds of NoSQL databases, there has been no accepted standard approach of managing transactions in NoSQL databases. Deuteronomy is an approach towards transaction processing in NoSQL databases. Deuteronomy separates the transactional component (TC) from the data component (DC). The TC manages transactions and transactions can span multiple DCs. In contrast to the approach proposed in this paper, Deuteronomy makes use of locking mechanism to manage concurrency and ensure consistency. Locking is useful but it has negative effects on the performance of transactions [6].

G-Store introduces a key grouping protocol to group keys for applications that need multi-row transactions. Groups within G-store are dynamic and have a life span. Thus, groups will be deleted after their life span. Transactions are limited to within a group and G-Store cannot provide transactions across groups [7].

Megastore uses entity groups formation similar to G-store. But in Megastore, group formation is static and an entity belongs to a single group throughout the life span of that entity. As such, ACID transactions can only take place within specified groups [4].

COPS (Cluster of Order Preserving Servers), introduces two variables called dependencies and versions to preserve order across keys. A COP is implemented using a distributed key value NoSQL database [8].

CloudTPS is like Deuteronomy, make use of two layers architecture which includes LTM (Local Transaction Manager) and the cloud storage. Transactions are replicated across LTMs to preserve consistency in the presence of failures [9].

III. SYSTEM ARCHITECHTURE

The below figure shows a general block diagram describing the activities performed by this project. The entire architecture has been implemented in nine modules which we will see in high level design and low level design in later chapters. Three major divisions in this project are-

1. Transactions
2. Operations
3. Data models

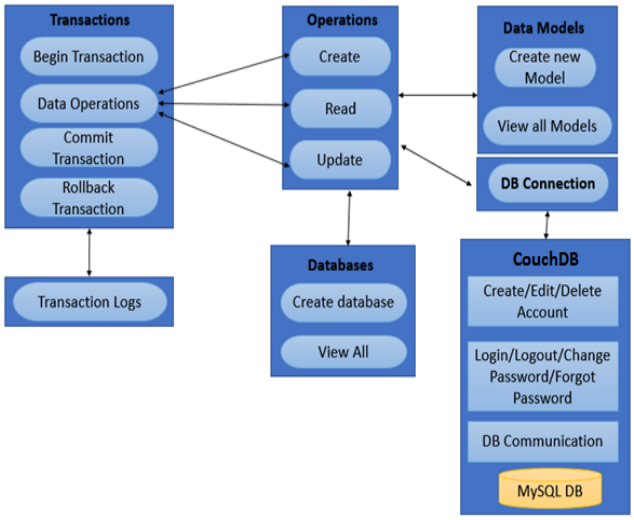


Figure 1. System Architecture of Transaction Protocol

A. Data Access Layer

Data access layer is the one which exposes all the possible operations on the data base to the outside world. It will contain the DAO classes, DAO interfaces, POJOs and Utilities are the internal components. All the other modules of this project will be communicating with the DAO layer for their data access needs

B. Account Operations

Account operations module provides the following functionalities to the end users of our project.

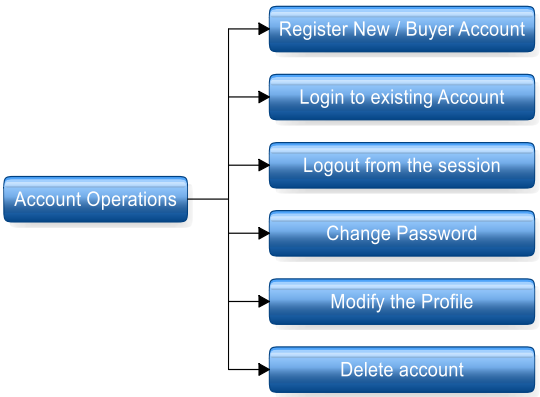


Figure 2. Account operations module

Here, the end user can perform various data operations. The possible data operations include the write access, read access, update, or delete access.

Before the user can perform any of these mentioned data operations, they have to select the database against which the data operations must be performed. Account operations module will be re-using the DAO layer to provide the above functionalities.

C. Connection to Couch DB and Databases

COUCHDB adopts a semi-structured data model and schema-less data base, based on the JSON (JavaScript Object Notation) format. It proposes an original approach, based on structured materialized views, which can be produced from document collections. In COUCH DB views are defined with the MAPREDUCE paradigm,

The end user can create a connection to the CouchDB by specifying the host name and the port number at the moment of instance is installed. The end user can also connect to a remote couch db that is present in a different geographic location by just entering its host's IP address and the port number. The default port number will be 5984.

A MapReduce framework simplifies implementing parallel algorithms:

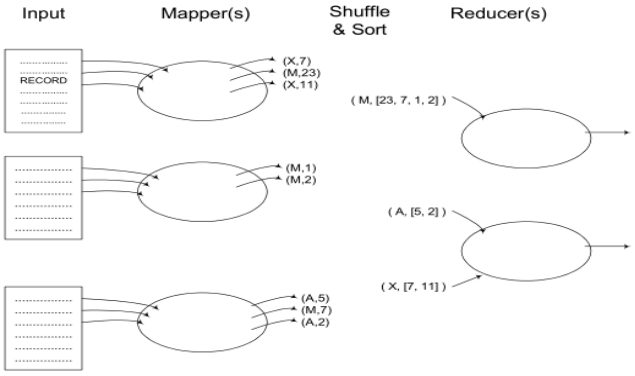


Figure 3. MapReduce design patterns

The user can create a new data base or view the list of all existing databases is using this module. The user can grant the permission on the database to the other users in the transaction layer, i.e. the user can allow other participant user to perform any of the transaction operations on the database is created.

JavaScript Object Notation is a text based data interchange format. JSON is built on two structures:

1. Obect: A collection of name and value pairs
2. Array: An ordered list of values

JSON is a simple text format initially designed for serializing Javascript objects. Javascript is a scripting language (distinct from Java) which is intensively used in Web browsers for "dynamic HTML" applications. Javascript function can access and modify the DOM tree of the document displayed by a browser, i.e. Any change made to this document, is instantaneously reflected in the browser window, so that Java script enables the creation of rich, interactive client-side applications.

The basic construct of JSON is a key-value pair of the form "key": value. Here is a first example, where the value is a character string:

Eg:

Table 2
PSEUDO CODE OF JSON

```
Kannada_Movie_ making
{
"title": "Bhangarada Manusya",
"year": "1990",
"summary": "Be Self Employer,
"country": "India",
"state": "karnataka",
"language": "kannada",

"director": {
"first_name": "puttanna",
"last_name": "kanagal",

"actors": [{
"first_name": "Raj",
"last_name": "kumar",
"role": "Hero"},
{
"first_name": "Bharathi",
"last_name": "V",
"role": "Heroinn"}]
}
```

D. Transactions

Before the end user can perform the transaction; he/she will have to select the database against which the transaction has to be executed. The user is going to select either data base is created by the user or the database have granted the access by other users. Each

and every single operation in the transaction session will be logged in the local mysql table and will be available to view in the GUI. The end user can either rollback or commit the transaction after all the data operations have been performed.

IV. EXPERIMENTAL RESULTS

The proposed model transactional logic is implemented as a prototype system using CouchDB for NoSQL transactions. This proposed model is implemented by java programming language.

Running one client, the time taking for one transaction to complete is about 0.2 seconds. With one client, experiments show that the system can handle between 30-40 transactions/second. With respect to correctness, the system showed the correctness for every transaction,

i.e. for each read transactions it take just 0.04 seconds. This experiment shows that the proposed system maintains good level of performance, while ensuring stronger consistency of the data in NoSQL databases.

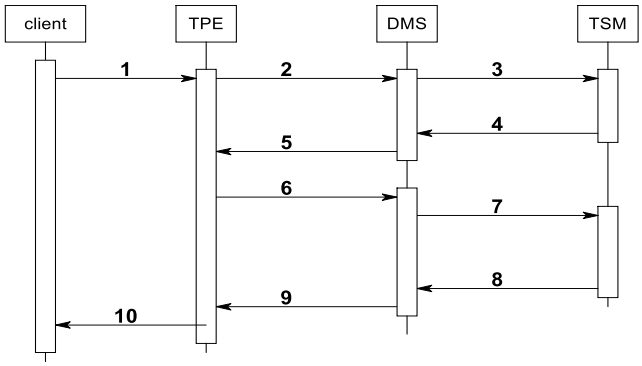


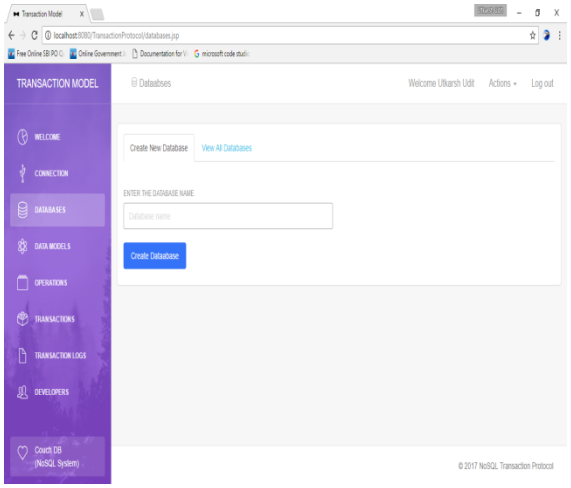
Figure 3. Interaction of System components

Where,

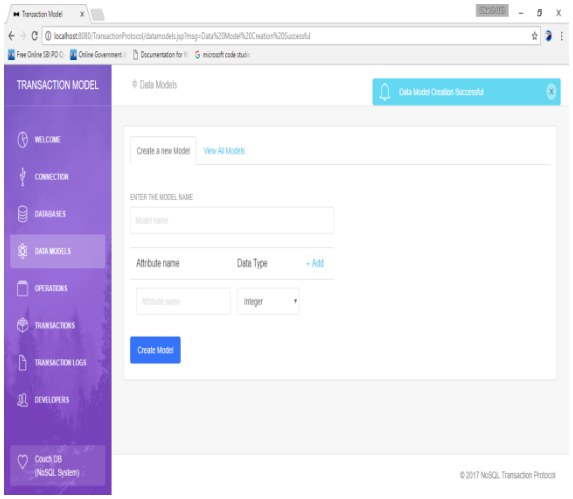
1. Transaction Request
2. Commit
3. Service
4. Ack.
5. Rollback
6. Request
7. Service
8. Ack.
9. Transaction saved
10. Ack.

V. RESULT ANALYSIS

Creation of Databases:

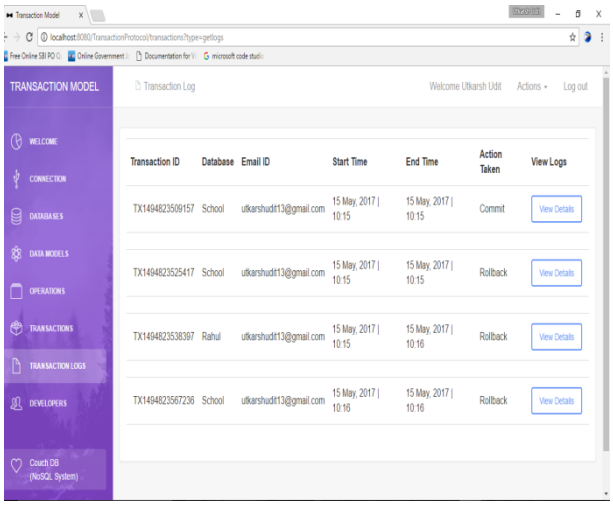


Creation of Data Models:



Transaction logs:

Every transaction will be stored in the transaction logs of the application as shown below:



VI. CONCLUSION

We proposed a new model, called M-Key transaction model, for NoSQL database systems. It provides NoSQL databases with standard ACID transactions support that ensures consistency of data. The project described the design of the proposed model and the architecture within which it is implemented. As a proof of concept the proposed approach is implemented using real Couch DB database system.

VII. REFERENCES

[1]. D. DeWitt and J. Gray, "Parallel Database Systems: The Future of High Performance Database Systems," Commun. ACM, vol. 35(6), Jun. 1992.

[2]. F. Chang, J. Dean, S. Ghemawat, W. C. Hsieh, D. a. Wallach, M. Burrows, T. Chandra, A. Fikes, and R. E. Gruber, "Bigtable: A distributed storage system for structured data," 7th Symp. Oper. Syst. Des. Implement. (OSDI '06), Nov. 6-8, Seattle, USA, 2006.

[3]. A. Silberstein, A. Silberstein, B. F. Cooper, B. F. Cooper, U. Srivastava, U. Srivastava, E. Vee, E. Vee, R. Yerneni, R. Yerneni, R. Ramakrishnan, and R. Ramakrishnan, "PNUTS: Yahoo!'s Hosted Data Serving PLatform," Proc. 2008 ACM SIGMOD Int..

[4]. J. Baker, C. Bond, J. Corbett, and J. Furman, "Megastore: Providing Scalable, Highly Available Storage for Interactive Services.," Proc. Of the Conference on Innovative Data system Research (CIDR 2011), 2011.

[5]. H. Berenson, P. Bernstein, J. Gray, J. Melton, E. O'Neil, and P. O'Neil, "A Critique of ANSI SQL Isolation Levels", 2007.

[6]. J. J. Levandoski, "Deuteronomy_: Transaction Support for Cloud Data," Conf. on Innov. Data Systems Research (CIDR), California, USA.vol. 48, 2011.

[7]. S. Das and A. El Abbadi, "G-Store_: A Scalable Data Store for Transactional Multi key Access in the Cloud," In: Proc. of the 1st ACM symposium on Cloud computing. Indianapolis, USA, ACM, 2010.

[8]. W. Lloyd, M. J. Freedman, M. Kaminsky, and D. G. Andersen, "Scalable Causal Consistency for Wide-Area Storage with COPS. In: Proc. of the 23rd ACM Symposium on Operating Systems Principles. Cascais, Portugal. 2011.

[9]. Z. Wei, G. Pierre, and C. H. Chi, "CloudTPS: Scalable transactions for web applications in the cloud," IEEE Trans. Serv. Comput., vol. 5, 2012.

[10]. A. Dey, A. Fekete, R. Nambiar, and U. Rohm, "YCSB+T: Benchmarking web-scale transactional databases," Proc. - Int. Conf. Data Eng., 2014.