A Multi-Key Transactions Model for NoSQL Cloud Database Systems

Adewole Ogunyadeka
Department of Computing & Communication Technologies
Oxford Brookes University, Oxford, UK
adewole.ogunyadeka-2013@brookes.ac.uk

Muhammad Younas
Department of Computing & Communication Technologies
Oxford Brookes University, Oxford, UK
m.younas@brookes.ac.uk

Hong Zhu
Department of Computing & Communication Technologies
Oxford Brookes University, Oxford, UK
hzhu@brookes.ac.uk

Arantza Aldea
Department of Computing & Communication Technologies
Oxford Brookes University, Oxford, UK
aaldea@brookes.ac.uk

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 increasingly been used in large scale applications that need high availability and efficiency but with weaker consistency. Consequently, such systems lack support for standard transactions which provide stronger consistency. This paper proposes a new multi-key transactional model which provides NoSQL systems with standard transaction support and stronger 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 proposed model is validated through a prototype system using MongoDB. Preliminary experiments show that it ensures stronger consistency and maintains good performance.**

Keywords—NoSQL databases, cloud, multi-key, transactions, consistency, availability, efficiency

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 [1]. 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 characterised 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.

In this paper we propose a new model that takes into account transactional principle of standard database systems. The objective is to provide consistency and to maintain the ACID properties while taking into consideration the availability and efficiency of NoSQL databases. The proposed model is built on the concept of Multi-Key transactions and is referred to as M-Key transactions model. It aims to overcome the challenges of implementing ACID transactions in NoSQL databases. The proposed M-Key model follows a loosely coupled architecture in order to separate transaction processing from underlying data storage and to ensure transparency and abstraction. In order to implement concurrency, the proposed approach exploits snapshot isolation technique [5]. Snapshot isolation is an optimistic concurrency control technique which allows for higher concurrency.

The potential contributions of the proposed model are summarized as follows.

- The design of a new Multi-Key transaction model for NoSQL systems that maintains ACID properties of transactions in order to ensure stronger consistency.
- Development of a loosely-coupled architecture that separates the transactional logic from underlying data thus ensuring transparency and abstraction.
- Development of a prototype system using real NoSQL system, MongoDB, which is evaluated using the YCSB+T benchmark based on standard Yahoo! Cloud Services Benchmark (YCSB). The results show enhanced consistency and performance.

The paper is structured as follows. Section II describes research issues related to NoSQL systems. Section III reviews related work. Section IV presents the theoretical model of the proposed approach. Section V describes the architecture. Section VI presents the transaction protocol. Section VII

provides implementation details and a proof concept. Section VIII concludes the paper and identifies future research work.

II. NOSQL DATABASE SYSTEMS AND RESEARCH ISSUES

The requirements and characteristics of NoSQL systems force them to deviate from adopting the principles of standard SQL database systems. Though this deviation brings in improvements such as high efficiency, availability and scalability, it also ends up in various issues. These include:

- *The adoption of simple data model with little or no normalization of data:* NoSQL data do not follow standard principles of normalization. But de-normalised data result in inconsistency and lack of integrity among data entities.
- *Lack of support for join operations and the inadequacy for formulating complex but useful queries:* The implication of de-normalizing and flattening out data into a single table implies that there is no support for join operations. Though this simplifies query processing it lacks the power and flexibility of designing useful queries which are available in relational databases.
- *Relaxation of consistency and referential integrity and lack of multi-key (and cross tables) transactions:* As discussed above, ACID transactions are not supported in NoSQL database. This affects the consistency and integrity across replicas of data.

The above issues make it difficult to implement transactions in NoSQL systems. In this paper we present a new model that provides transactional support for NoSQL systems in order to enhance data consistency without severely impacting availability and efficiency.

III. RELATED WORK

Various approaches have been proposed to address transaction management in NoSQL databases. However, because of the diverse flavours and kinds of NoSQL databases, there has been no accepted standard approach of managing transactions in NoSQL databases. Deuteronomy [6] 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.

G-Store [7], 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. Megastore [4], 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.

COPS (Cluster of Order Preserving Servers) [8], introduces two variables called dependencies and versions to preserve order across keys. It is implemented using a distributed key-value NoSQL database. CloudTPS [9], 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.

IV. THE PROPOSED APPROACH: THEORETICAL MODEL

A NoSQL transaction $\mathcal{NST}$ is defined as the execution of a (cloud) application which comprises different operations that provide transitions between consistent states of the shared data. In other words, $\mathcal{NST}$ is a sequence of operations which are executed in a way such that all of them are successfully completed or none at all. $\mathcal{NST}$ is required to follow the ACID (atomicity, consistency, isolation, durability) properties.

$\mathcal{NST}$ is a multi-key transaction as it involves more than one data key item. Most NoSQL systems, do not perform multi-key operations.

Definition 1: A $\mathcal{NST}$ can be formally defined as a tuple, $\mathcal{NST} = (\mathcal{OP}, \mathcal{PaO})$, where $\mathcal{OP}$ is a set of operations, $\mathcal{OP} = \{\mathcal{OP}_i \mid i = 1...n\}$, and $\mathcal{PaO}$ is a partial ordering of the operations which determines their order of execution. For instance, $\mathcal{OP}_i > \mathcal{OP}_j$ represents that $\mathcal{OP}_i$ is executed before $\mathcal{OP}_j$.

In the proposed model, $\mathcal{OP}_i^r[\mathcal{DE}]$ represents a read operation of NST; meaning that $\mathcal{NST}$ reads data entity, $\mathcal{DE}$, from a NoSQL database. Similarly, $\mathcal{OP}_i^w[\mathcal{DE}]$ represents a write operation of NST; meaning that $\mathcal{NST}$ writes (updates) a data entity, $\mathcal{DE}$, to a NoSQL database.

The above read and write operations ($\mathcal{OP}_i^r[\mathcal{DE}]$ and $\mathcal{OP}_i^w[\mathcal{DE}]$) are used to model the CRUD (Create, Read, Update and Delete) operations. In the proposed model, $\mathcal{OP}_i^r[\mathcal{DE}]$ is simply to read data without any modification to the data. $\mathcal{OP}_i^w[\mathcal{DE}]$ is to write data meaning that data can be modified through Create, Update or Delete operation.

In addition, to data read/write operations, $\mathcal{NST}$ is also associated with (control) operations, begin or start, commit and abort. These are explained as follow.

Begin or start operation: The execution of each $\mathcal{NST}$ must be marked through a begin or start operation. That is, $\mathcal{NST}$ should begin first before any of its operations ($\mathcal{OP} = \{\mathcal{OP}_i \mid i = 1...n\} \in \mathcal{NST}$) can be executed.

Commit and Abort operations: If $\mathcal{NST}$ is successfully executed then it terminates with a 'commit' operation. If $\mathcal{NST}$ cannot be successfully executed then it terminates with an abort operation.

$\mathcal{NST}$ can be of type seq ($\mathcal{Begin} \mid \mathcal{OP}_i \mid \mathcal{Cmt} \mid \mathcal{Abt}$) but with the condition that either $\mathcal{Cmt}$ (commit) or abort ($\mathcal{Abt}$) occurs only once within the sequence.

A $\mathcal{NST}$ comprises different read/write operations but it can have either one commit or abort operation. This is denoted as:

- $\mathcal{NST}_i = \{\mathcal{Begin}\} \cup \{\mathcal{OP}_1^r[\mathcal{DE}], \ldots, \mathcal{OP}_n^r[\mathcal{DE}]\} \cup \{\mathcal{OP}_1^w[\mathcal{DE}], \ldots, \mathcal{OP}_n^w[\mathcal{DE}]\} \cup \{\mathcal{Cmt}_i, \mathcal{Abt}_i\}$

  Where $\mathcal{DE}$ is Data Entity.

- $\mathcal{C}mt_i \in \mathcal{NST}_i$ iff $\mathcal{A}bt_i \notin \mathcal{NST}_i$
  i.e., If $\mathcal{NST}_i$ is committed then abort operation cannot be executed.
- Assume a (control) operation, ct_i, is $\mathcal{C}mt_i$ or $\mathcal{A}bt_i$ (transaction commits or aborts), then for any read/write operation $OP_i[DE] \in \mathcal{NST}_i$, $OP_i[DE] < ct_i$. In other words, commit or abort operation must be after the read/write operation.
- If $OP_i^r[DE]$, $OP_i^w[DE] \in \mathcal{NST}_i$, then such read/write operations should be ordered either as $OP_i^r[DE] < OP_i^w[DE]$ or $OP_i^w[DE] < OP_i^r[DE]$. That is, data entity, DE, should be read and written in a proper order.

V. SYSTEM ARCHITECTURE

The proposed M-Key transactions model is to be implemented in a loosely coupled architecture. It follows loosely-coupled architectural style in order to separate the implementation of transactional logic from the underlying data and to ensure transparency and abstraction. Figure 1 diagrammatically represents the proposed architecture at the higher level of abstraction. It comprises three main components: Transaction Processing Engine, Data Management Store, and Times Stamp Manager.

The fuller implementation of the proposed architecture is still under development. But the main functions of each of these components are briefly described as follows.

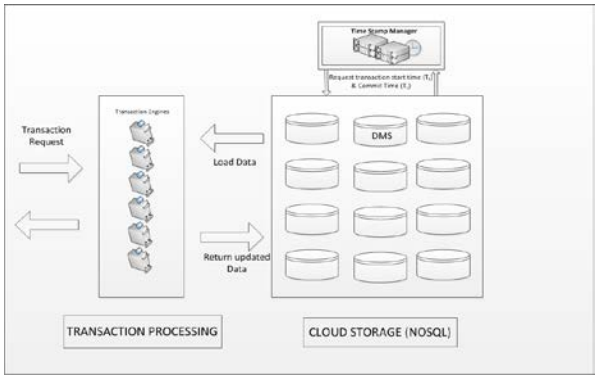

Fig 1. The Proposed System Architecture

Transaction Processing Engine (TPE): The TPE is responsible for implementing M-Key transactions in the proposed system. The main functions which are to be performed by the TPE include:

- Receiving transactional requests from clients and managing such transactions
- Storing of schema information (as NoSQL systems do not provide facilities for schema information)
- Defining relationships between different entities of data
- Provide support for *join operations* (as NoSQL do not support such operations)

Data Management Store (DMS): This component represents the actual NoSQL system such as MongoDB. DMS stores all (Big) data persistently. This component is highly scalable in order to meet Big Data storage requirements. Further, it replicates data in terms of different replicas in order to ensure improved efficiency, high availability and fault tolerance. Replication is the common approach across all NoSQL systems. In the proposed system, the DMS layer will implement snapshot isolation protocol in cooperation with the Time stamp manager (TSM) in order to provide concurrency of transactions.

Times Stamp Manager (TSM): The TSM is to manage the ordering and scheduling of transactions in the proposed system. It interacts with DSM and TPE in order to schedule the execution of the different operations of a transaction. The objective is to maintain consistency of data when concurrently accessed by different transactions. The proposed concurrency technique is to implement Snapshot Isolation which is non-blocking and provides higher concurrency and high efficiency in transactions processing.

VI. TRANSACTION PROTOCOL

This section presents the proposed transaction protocol in order to illustrate the different steps involved in transaction processing. Figure 2 depicts the flow of requests which are communicated between the client, TPE, DMS and TSM. Client represents user's cloud application that submits transactions to the proposed system.

The different steps involved in the protocol are explained as follows. Note that these steps must adhere to the specification (definitions and constraints) specified in Section IV.

1. A client initiates a request to start a new NoSQL transaction ($\mathcal{NST}$).
2. TPE receives client's request and generates an ID for the $\mathcal{NST}$ which is to be executed on NoSQL data. TPE then sends the $\mathcal{NST}$'s ID and related information to the DMS.
3. DMS receives the $\mathcal{NST}$'s ID and related information in order to know which data entities are to be accessed (read/updated) by the $\mathcal{NST}$. DMS then sends the $\mathcal{NST}$'s ID and related information about data entities to the TSM in order to ensure scheduling of $\mathcal{NST}$ and other transactions.
4. TSM saves the information about $\mathcal{NST}$ and it responds with a start-time of a transaction. This time serves as a start time-stamp, which is to determine the order of execution and also the commitment of the $\mathcal{NST}$.

   As in Section IV, if $OP_i^r[DE]$, $OP_i^w[DE] \in NST_i$, then these read/write operations should be ordered either as $OP_i^r[DE] < OP_i^w[DE]$ or $OP_i^w[DE] < OP_i^r[DE]$. That is, data entity, DE, should be read and written in a proper order following the time-stamp information.

5. Based on the above, DMS releases the required data entities to the TPE where $\mathcal{NST}$ is actually taken place. Note that the proposed architecture separates transaction processing from the actual NoSQL database system in order to ensure abstraction and transparency.

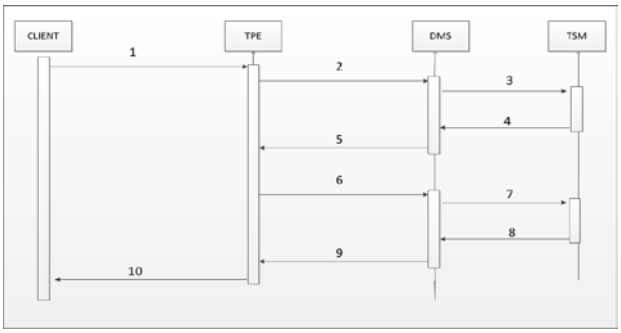

Figure 2. Interaction of System's Components

6. Once $\mathcal{NST}$ is completed, TPE sends the updates (made to data entities) to the DMS. This means that if $\mathcal{NST}$ updates a data entity (modify, delete) then DMS has to reflect this in the data store in order to ensure that data is consistent.
7. The DMS contacts TSM to request a commit timestamp. TSM checks if another transaction has updated data after its start timestamp of the requesting transaction. If this happens, then the $\mathcal{NST}$ aborts and sends the information to the client through the TPE. Otherwise, it continues.
8. The TSM responds to the DMS with a commit timestamp. The DMS then stores the data in the data store.
9. The DMS responds with a commit message to the TPE. This means that $\mathcal{NST}$ is successfully committed using the commit operation, $\mathcal{C}mt_i$.

VII. PROOF OF CONCEPT

A. Evaluation Benchmark

One of the major issues in evaluating NoSQL database systems is that there is no standard benchmark yet. According to our research, Yahoo! Cloud Services Benchmark (YCSB) is the most commonly used cloud services benchmark in order to evaluate the performance of NoSQL systems. However, YCSB benchmark does not support transactions. Instead it is limited to evaluating single NoSQL operation rather than group of operations as in transactions. For the evaluation of the proposed approach, we therefore adopt the YCSB+T benchmark which is developed for web-scale transactional systems [10]. Basically, the YCSB+T benchmark is an extension of the YCSB benchmark and it remains the only benchmark for cloud systems with support for transactions.

Our preliminary experiments take into account the following elements in the evaluation.

(i) Transactional overhead: This is to evaluate the overhead caused by the proposed M-Key transactions model in introducing ACID transactions into NoSQL database systems. As described above, majority of the NoSQL do not support ACID transactions

(ii) Consistency: This is to show how the M-Key transactions model guarantees stronger consistency in NoSQL database systems while maintaining acceptable level of performance.

We use the closed-economy workload (as in [10]) to evaluate the (i) Number of transactions per second, and (ii) Consistency (or correctness) of transactions.

B. Experimental Setup and Results

The proposed model is implemented as a prototype system using the NoSQL MongoDB which does not support multi-key transactions. In the proposed model, transactional logic is implemented using Python language. The implementation is carried out on a 16GB RAM Windows PC system.

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 multi-key transactions/second. With respect to correctness, the system showed 100% correctness for every transaction. Read transactions take just 0.04 seconds for each read. These experiments show that the proposed system maintains good level of performance while ensuring stronger consistency of the data in NoSQL databases.

VIII. 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 paper 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 NoSQL database system of the MongoDB. Evaluation is carried out through the YCSB+T benchmark. Preliminary experiments show promising results in terms of ensuring consistency and performance.

Future work includes full implementation of the proposed system and with detailed experimentation.

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. Conf. Manag. data - SIGMOD '08, 2008.

[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, "Don ' t Settle for Eventual : 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.