http://dx.doi.org/10.5392/IJoC.2016.12.1.060

Concurrency Control Method to Provide Transactional Processing for Cloud Data Management System

Dojin Choi
Department of Information and Communication Engineering
Chungbuk National University, Cheongju Chungbuk, 362-763, Republic of Korea

Seokil Song
Department of Computer Engineering
Korea National University of Transportation, Chungju, Chungbuk, 27469, Republic of Korea

ABSTRACT

As new applications of cloud data management system (CDMS) such as online games, cooperation edit, social network, and so on, are increasing, transaction processing capabilities for CDMS are required. Several transaction processing methods for cloud data management system (CDMS) have been proposed. However, existing transaction processing methods have some problems. Some of them provide limited transaction processing capabilities. Some of them are hard to be integrated with existing CDMSs. In this paper, we proposed a new concurrency control method to support transaction processing capability for CDMS to solve these problems. The proposed method was designed and implemented based on Spark, an in-memory distributed processing framework. It uses RDD (Resilient Distributed Dataset) model to provide fault tolerant to data in the main memory. In our proposed method, database stored in CDMS is loaded to main memory managed by Spark. The loaded data set is then transformed to RDD. In addition, we proposed a multi-version concurrency control method through immutable characteristics of RDD. Finally, we performed experiments to show the feasibility of the proposed method.

Key words*: Transaction, Cloud Data Management, Snapshot Isolation.*

1. INTRODUCTION

Cloud computing has become a prevalent infrastructure for many application domains due to its several virtues such as scalability, fault tolerance, high performance cost, pay-as-you-go and so on. Data management system is one of the applications of cloud computing [1]. Many cloud data management systems have been proposed and widely used. To our knowledge, representative cloud data management systems are BigTable [2], Cassandra [3], HBase [4] and so on.

A CDMS (Cloud Data Management System) provides transparent partitioning and replication with improvement scalability, availability, fault-tolerant through automated load balance and fault recovery. In spite of those features, [1] claims that CDMSs will not replace the traditional RDBMS in the near future. Rather than replaces RDBMS, CDMSs provide another choice for the applications which are suitable to be deployed in the cloud, i.e., large scale data analysis and data management in the web applications. According to [1], those applications do not require transactional processing capabilities.

However, recently, some CDMS applications such as online game, cooperation edit program, social network services, and so on require transactional processing functions. For example, in social network services, when a user A follows a user B, two operations to add B to the following list of A and add A to the followers list of B must be processed atomically. We can find other examples requiring transactional processing easily in online game and cooperation edit service area.

Recently, many methods to process transactions on CDMS have been proposed [5]-[13]. In [5], S3 provides eventual consistency and additional consistency requirements for some applications should be implemented based on it. Some of them provides limited transaction processing on some partition of managed data set. This approach provides very high consistency, but, most methods in this approach use OCC (optimistic concurrency control) methods based on lock techniques which may delay transaction processing time [7]-[10].

Methods proposed in [6], [10], [12] decompose a database kernel into TC (transaction component)s and DC (data component)s. TCs provide concurrency control and fault recovery functions, while DCs manages caches and access

** Corresponding author, Email: sisong@ut.ac.kr*

Manuscript received Feb. 24, 2016; revised Mar. 07, 2016; accepted Mar. 14, 2016

methods. TCs and DCs does not know each other's internal architecture. That is, TCs do not need to know physical storage architecture of DCs, and, also, DCs do not know transaction processing architecture of TCs. Consequently, TCs and DCs can be distributed across nodes on a cluster so as to improve the scalability and the transaction throughput of CDMSs. Even though they show good scalability and transaction throughput, it is very hard to implement them and apply them to CDMSs. Therefore, it is not easy to integrate them to existing CDMSs.

[13] proposed a modified single version OCC for CDMSs. It modifies the verification phase and the validation phase of OCC to decrease abort ratios of transactions. Particularly, the validation phase is processed in distributed manner. This approach, also, has the same problem to those of [11], [12]. We have to redesign and implement of core engine of existing CDMS.

In this paper, we propose a new concurrency control method to support distributed transaction processing for existing CDMSs. The proposed method can be easily integrated with the existing CDMSs without any modification. The proposed method is based on Spark [15], which is in-memory distributed and parallel processing framework, to improve transaction processing throughput.

In our proposed method, database stored in a CDMS is loaded to main memory managed by Spark. The loaded data set is transformed to a RDD (Resilient Distributed Data Set). RDD is a data model for fast fault recovery to continue transaction processing in main memory even though system failures are occurred. RDD model has immutable characteristics so when a part of RDD should be modified, a new version of RDD is created. There are two types of operations for RDDs.

One is transformation operation to create new RDD from an old RDD. For example, a filter operation filters an old RDD with a condition and produces a new RDD that satisfies the condition. Action operation return a value after running a computation on the RDD. We use transformation operations to produce the new version of a data and manage the versions of the data to enable multi-version concurrency control.

This paper is organized as follows. In Section 2, we summary existing transaction processing methods for CDMSs. In Section 3, we propose a new transaction processing system based on Spark. Then, we show experimental results in Section 4, and finally conclude in Section 5.

2. RELATED WORK

In this section, we describe the transaction processing techniques of existing CDMSs and serializable snapshot isolation (SSI) techniques. Actually, our proposed method is based on Deuteronomy and SSI. Therefore, we describe the existing transaction processing techniques in brief and Deuteronomy and SSI in detail.

2.1 Existing Transaction Processing Methods for CDMSs

Simple Storage Service (S3) is Amazon's highly available cloud storage solution. S3 is used as the disk for database. It uses key-value data model and keys are referred to as records. In S3, updates are not necessarily applied in the same order as they were initiated. The only guarantee that S3 gives is that updates will eventually become visible to all clients and that the changes persist. This property is called eventual consistency. If an application has additional consistency requirements, then such additional consistency guarantees must be implemented on top of S3 as part of the application [5].

[5] presents various protocols in order to store, read, and update objects and indexes using S3. It preserves the scalability and availability of S3 and achieves the same level of consistency as a database system. It follows the distributed systems' approach, thereby preserving scalability and availability and maximizing the level of consistency that can be achieved under this constraint.

ElasTraS [7] is a data store that is designed to be a light-weight data store that supports only a subset of the operations supported by traditional database systems. ElasTraS is analogous to partitioned databases which are common in enterprise systems, while adding features and components critical towards elasticity of the data store. It uses proven database techniques to process concurrency control, isolation, and recovery, while using design principles of scalable systems such as Bigtable to overcome the limitations of distributed database systems.

Generally, existing key value stores only guarantee the atomicity of a transaction on single keys since majority of current web applications require that. Many other applications such as online multi-player casino, collaborative applications need multi-key accesses. The reduced consistency guarantees and single key access granularity supported by the key-value stores often places huge burden on the application programmers.

G-Store [8] which is a scalable data management system for cloud provides transactional multi key access guarantees. It proposed key group abstraction that defines a granule of on-demand transactional access over dynamic, non-overlapping groups of keys using a key-value store as an underlying substrate.

The key grouping protocol uses the key group abstraction to transfer ownership for all keys in a group to a single node which then efficiently executes the operations on the key group. It is suitable for applications that require transactional access to groups of keys that are transient in nature, but live long enough to amortize the cost of group formation. The number of keys in a group should be small enough to be stored in a single node. Considering the size and capacity of present commodity hardware, groups with thousands to hundreds of thousands of keys can be efficiently supported.

In [10], storage requirements of today's interactive online applications are introduced as scalability, rapid development, responsiveness (low latency), durability and consistency, and fault tolerant. Megastore [10] is a storage system developed to meet the storage requirements of today's interactive online services. It blends the scalability of a NoSQL data store with a traditional RDBMS (Relational Database Management System). It uses synchronous replication to achieve high availability and a consistent view of the data.

It partitions the data store and replicate each partition separately, providing full ACID (Atomicity, Consistency, Isolation, Durability) semantics within partitions, but only limited consistency guarantees across them. It provides traditional database features, such as secondary indexes, but only those features that can scale within user-tolerable latency limits, and only with the semantics that its partitioning scheme can support. In [10], Megastore only guarantees serializability among transactions accessing the same partitioned group by allowing update operations on a group serially, while aborting and restarting other concurrent updates.

Generally, traditional DBMS consists of query processing component and DB kernel. DB kernel provides tightly integrated code such as access methods, data caching and persistence, concurrency control and recovery. [6] splits the DB kernel into transaction component (TC) and data component (DC). Transaction component (TC) performs concurrency control and recovery functions, and has a logical level knowledge about keys and records. It does not know physical structures such as pages, buffers and so on, physical structure. Data component (TC) have access methods and cache management functions. It provides atomic logical operations and does not know how they are grouped in user transactions.

Unbundling transaction approach has some advantages as follows. In the multi-core architectures, unbundled transaction approach may run TC and DC on separate cores. It easy to extend DBMS functions. For example, to provide new access method, it only needs to change DC. Also, it is suitable for cloud data management system with transactional support. TC coordinates transactions across distributed collection of DCs without 2PC. It is possible to add TC to data management system that already supports atomic operations on data.

[6] proposed a partition lock method that have the range as resource like keys, pages, and tables. The TC can request a lock on a range resource from the DC, and each table is partitioned to a set of range resources. The boundary of the range resource may be not keys in the table. It means that the TC can know these boundaries without the need to lock at the data. The TC maps the range in where clause into the set of partition resources on which the TC can request locks. The TC requests a lock on each partition before accessing its records. This lock acts as a cover lock in which the TC can talk safely with the DC to discover the records inside the locked range resource. When multiple partitions are required to cover the range, the partitions are locked one at a time, as records are accessed sequentially within the range. The basic idea of Deuteronomy [11] system is from unbundling transaction approach [6]. It seems that Deuteronomy system is a complete system including upgraded unbundling transaction approach. In [12], a dynamic range lock method for [6] is proposed.

MaaT [13] introduces the new design of OCC (Optimistic Concurrency Control) for transactions in CDMSs. It provides unlimited transactional processing capabilities. It re-designs the existing OCC to eliminate locks during two phase commit for distributed transactions. Without using locks in executing distributed transactions, it also can reduce the abort rate of OCC incurred by deadlock avoidance mechanisms.

2.2 Serializable Snapshot Isolation (SSI)

Snapshot isolation (SI) based transaction execution model is a multi-version based approach utilizing the optimistic concurrency control concepts [16]. A transaction T1 executing with SI takes snapshot of committed data at start of T1 called start timestamp, always reads/modifies data in its own snapshot and updates of concurrent transactions are not visible to T1. T1 is allowed to commit only when another Ti running concurrently has not already written the data item that T1 intends to write. Generally, this commit protocol is called as first committer wins (FCW). SI has many benefits so it is widely used in many systems.

Reads of transactions with SI are never blocked even though concurrent transactions that update same data item. The concurrency of SI is similar to that of read committed isolation level. However, SI has critical drawback. It is vulnerable to anomalies such as write skew anomaly and read-only transactional anomaly. That is, SI breaks serializability in some cases. In [17] proposed serializable snapshot isolation (SSI) that solve the write skew anomaly. It uses lock techniques to automatically detect and prevent snapshot isolation anomalies at runtime for arbitrary applications.

3. PROPOSED MULTI-VERSION CONCURRENCY CONTROL METHOD FOR CDMS

The multi-version concurrency control method proposed in this paper is based on the RDD model of Spark. A RDD is immutable so when it is modified the new version of the RDD is created. The proposed concurrency control supports the SI (Snapshot Isolation) [16] when processing transactions by using this characteristic. The SI, which is one of transaction isolation level, update, allows a transaction to read committed data items always, by updating data items after creating the

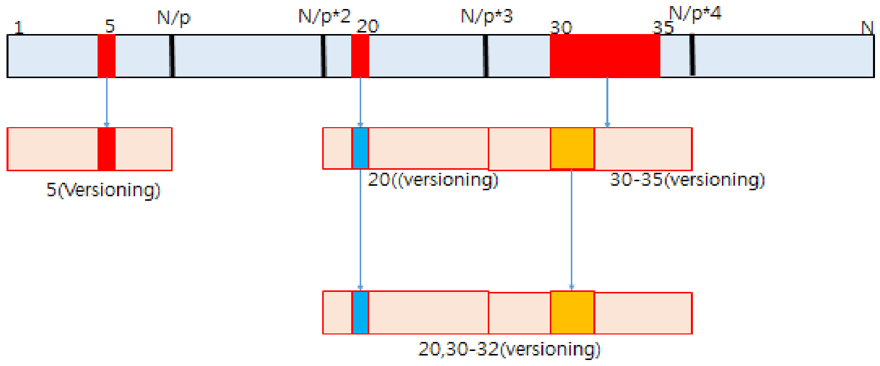
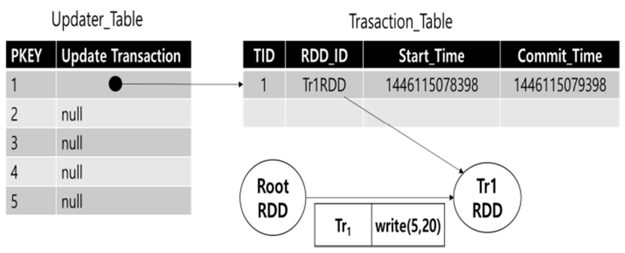


Fig. 1 Example of Version Management

snapshot of data items so as to improve the concurrency of read consistency.

In our proposed concurrency control method, a database in a CDMS is loaded into distributed main memory as a RDD managed by Spark. Then all transactions read and write data items of the RDD in distributed main memory. Writes operations of transactions create new versions of the RDD and the RDD versions are managed by our concurrency control method. We logically split a RDD to a number of partitions. Each partition contains the same number of data items. When a transaction write data items, we apply the map transformation operation to create a new version of the RDD which contains the partitions including the data items. Fig. 1 shows the versioning method of the proposed concurrency control method. In the figure, the key range of database is from 1 to N and the size of partition is p. Therefore, the key range of the ith partition is from N/p*(i-1) + 1 to N/p*i. When a data item whose key is 5 is to be updated, the first partition is containing 5 is filtered and mapped to the new version RDD of the partition. When a transaction updates data items whose keys are 20, 30, 31, 32, two partitions including those keys are filtered and mapped to the new version RDDs.

We maintain Update_table and Transaction_table to manage multiple version RDDs as shown in the Fig. 2. PKEY is for partition identifiers and update transaction is for transaction identifiers which most recently updates partition 1. Transaction_table stores transactions that creates new RDDs for partitions. TID is for transaction identifiers and RDD_ID is for RDD identifiers. In the figure, Tr1 transaction create a new RDD (Tr1RDD) to update a data item whose key is 5. Then, other transactions can access the Tr1RDD by referencing Update_table and Transaction_Table quickly.


Updater_Table

PKEY	Update Transaction
1	●
2	null
3	null
4	null
5	null

Trasaction_Table

TID	RDD_ID	Start_Time	Commit_Time
1	Tr1RDD	1446115078398	1446115079398

Fig. 2 Example of Transaction Processing

4. PERFORMANCE EVALUATION

We perform experiments to show the scalability of the proposed concurrency control method. Server cluster used in our experiments consists of 4 nodes, and each node is equipped with Intel Xeon E5-2620 CPU (2 Ghz, 4 cores) and 8GB DDR3 main memory. The number of transactions used in the experiments are varied from 3,000 to 30,000. Transactions are generated by a number of threads, and each thread runs 100 transactions. The number of threads are varied from 300 to 3,000.

We implement the proposed method based on Spark by using RDD to maintain multi-versions of data items. To perform experiments, we also implement a transaction generator. The transaction generator consists of a number of threads which execute two kinds of transactions repeatedly. One kind of transaction performs only read operations, while the other includes read and write operations. Keys of each transaction is selected randomly.

Table 1 Experimental parameters

Parameters	Values
Number of nodes	8
OS	Ubuntu 14.04
CPU	Xeon 2Ghz x 4 core
RAM	8GB
Number of transactions	3,000 ~ 30,000

In this experiment, we show the scalability of the proposed concurrency control method based on Spark. Therefore, we measure the number of transactions committed per a second. Fig. 3 shows the transaction throughput with varying the number of transactions. As shown in the figure, transaction throughput increases as the number of transactions increases. However, when the number of threads are more than 100 (1000 transactions), the throughput slightly decreases.

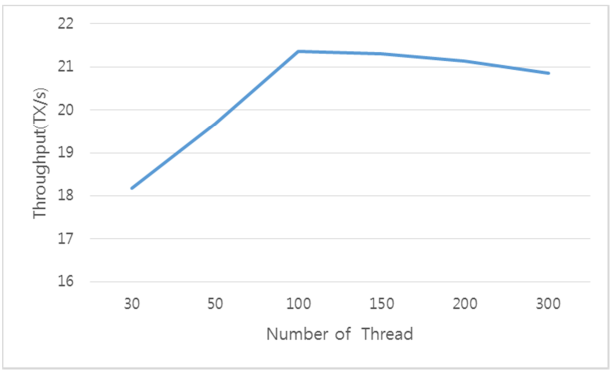


Fig. 3 Transaction throughput

5. CONCLUSION

In this paper, we designed and implemented the existing OCC based on Spark by using the immutable characteristics of RDD. The main contribution of the proposed method is that it can be easily integrated with the existing CDMSs without any modification. All transactions are processed on main memory managed by Spark. We performed experiments on 8 nodes cluster to show that the proposed method is scalable to the number of transactions.

6. ACKNOWLEDGEMENT

This was supported by Korea National University of Transportation in 2015, and also, this research was supported by Basic Science Research Program through the National Research Foundation of Korea(NRF) funded by the Ministry of Education (NRF-2014R1A1A2059342)

REFERENCES

[1] Y. Shi, X. Meng, J. Zhao, X. Hu, B. Liu, and H. Wang, "Benchmarking Cloud-based Data Management Systems," Proc. CloudDB '10, 2010, pp. 47-54.

[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," ACM Transactions on Computer Systems (TOCS), vol. 26, no. 2, 2008, p. 4.

[3] A. Lakshman and P. Malik, "Cassandra: A Decentralized Structured Storage System," ACM SIGOPS Operating Systems Review, vol. 44, no. 2, 2010, pp. 35-40.

[4] A. Khetrapal and V. Ganesh, *HBase and Hypertable for Large Scale Distributed Storage Systems*, Dept. of Computer Science, Purdue University, 2006.

[5] M. Brantner, D. Florescu, D. Graf, D. Kossmann, and T. Kraska, "Building a Database on S3," Proc. ACM SIGMOD '08, 2008, pp. 251-264.

[6] D. Lomet and M. F. Mokbel, "Locking Key Ranges with Unbundled Transaction Services," Proc. VLDB Endowment, 2009, pp. 265-276.

[7] S. Das, D. Agrawal, and A. E. Abbadi, "ElasTraS: An Elastic Transactional Data Store in the Cloud," Proc. USENIX HotCloud Workshop, San Diego, 2009, pp. 131-142.

[8] S. Das, D. Agrawal, and A. E. Abbadi, "G-store: A Scalable Data Store for Transactional Multi Key Access in the Cloud," Proc. 1st ACM Symposium on Cloud Computing, 2010, pp. 163-174.

[9] Z. Wei, G. Pierre, and C. H. Chi, "Scalable Transactions for Web Applications in the Cloud," Proc. Euro-Par 2009 Parallel Processing, 2009, pp. 442-453.

[10] J. Baker, C. Bond, J. C. Corbett, J. J. Furman, A. Khorlin, J. Larson, and V. Yushprakh, "Megastore: Providing Scalable, Highly Available Storage for Interactive Services," Proc. CIDR, 2011, pp. 223-234.

[11] J. J. Levandoski, D. B. Lomet, M. F. Mokbel, and K. Zhao, "Deuteronomy: Transaction Support for Cloud Data," Proc. CIDR, 2011, pp. 123-133.

[12] T. Kim and S. Song, "Dynamic Partition Lock Method to Reduce Transaction Abort Rates in Cloud Data Management Systems," Cluster Computing Journal, vol. 18, no. 1, 2014, pp. 233-242.

[13] H. A. Mahmoud, V. Arora, F. Nawab, D. Agrawal, and A. E. Abbadi, "MaaT: Effective and Scalable Coordination of Distributed Transactions in the Cloud," Proc. VLDB Endowment, vol. 7, no. 5, 2014, pp. 329-340.

[14] A. Dey, A. Fekete, and U. Röhm, "Scalable Distributed Transactions across Heterogeneous Store," Proc. ICDE, 2014, pp. 125-136.

[15] M. Zaharia, M. Chowdhury, M. J. Franklin, S. Shenker, and I. Stoica, "Spark: Cluster Computing with Working Sets," Proc. the 2nd USENIX Conference on Hot Topics in Cloud Computing, 2010, p. 10.

[16] A. Adya, B. Liskov, and P. O. Neil, "Generalized Isolation Level Definitions," Proc. IEEE Conference on Data Engineering, 2000, pp. 67-78.

[17] M. J. Cahill, U. Rohm, and A. D. Fekete, "Serializable Isolation for Snapshot Databases", TODS, vol. 34, no. 4, 2009, pp. 729-738.

Dojin Choi
He received the BS and MS degrees in Computer Engineering Department from Korea National University of Korea, Republic of Korea in 2014 and 2016 respectively. He is in the doctoral course in Chungbuk National University, Republic of Korea. His research interests are database systems, transaction processing systems, big data and so on.

Seokil Song
He received the BS, MS and PhD degrees in Computer and Communication Department from Chungbuk National University of South Korea in 1998, 2000 and 2003, respectively. He is an Associate Professor of the Computer Engineering Department, Korea National University of Transportation, Republic of Korea. His research interests are database systems, index structures, concurrency control, storage systems, sensor network and XML database.