A Distributed Algorithm for SI Transactions Serializability in Cloud Computing

Huang Bin, Peng Yuxing and Peng Xiuning
1School of Computer Science, Wuhan University, Wuhan, 430072, China
2Department of Computer, Huaihua University, Huaihua, 418008, China
3National Laboratory of Parallel and Distributed Processing,
National University of Defense Technology, Changsha, 410073, China

Abstract: There are well known anomalies permitted by snapshot isolation that can lead to violations of data consistency by interleaving transactions that individually maintain consistency. Until now, there are some ways to prevent these anomalies only in single computer and there are not the corresponding solving methods in cloud computing. This paper describes our PDCC algorithm to detect cycles in a snapshot isolation dependency graph and abort transactions to break the cycle in cloud computing. The algorithm ensures serializable executions for SI transactions in cloud computing. Based on the transaction concurrency control of Percolator, we have implemented our algorithm in an open source cloud database system (HBase) and our performance study shows that PDCC throughput and scalability are good.

Key words: Cloud computing, transaction, SI, dependency graph

INTRODUCTION

According to CAP theories (Brewer, 2000; Gilbert and Lynch, 2002), data sharing system can only satisfy two of the three features, namely consistency, availability and tolerance to partitions. Cloud computing system is a distributed system, so either consistency or availability can be met in cloud computing system. Some of current cloud computing systems, in order to meet the availability, reduce the requirements for consistency (Vogels, 2009) and do not support cross-line and cross-table (Levandoski et al., 2011; Helland, 2007) operations, such as Google’s BigTable (Chang et al., 2006), Amazon’s SimpleDB, Facebook’s Cassandra, Windows’ Azure, Dynamo (DeCandia et al., 2007) and PNUTS (Cooper et al., 2008), etc. and their application covers webpage search, user setting and recommendation on social network (Levandoski et al., 2011; Wei et al., 2009). While other cloud computing systems strive for consistency at the costs of availability, such as CloudIPS (Wei et al., 2009), Percolator (Peng and Dabek, 2010), ElasTras (Das et al., 2009, 2010a), G-Store (Das et al., 2010b), Deuteronomy (Levandoski et al., 2011), etc. and their application covers online auctions, CO editor, credit card withdrawals, air ticket booking (Levandoski et al., 2011), etc.

For the systems preferring consistency, the SI (Snapshot Isolation (Berenson et al., 1995) method is important to guarantee both consistency and high-efficient handling of transactions. With SI method, the data are read according to Read Latest Version (RLV) rules, namely, when data are read, the transaction can only read the latest version of data submitted before its start and it can't read the data updated by other transactions after its start. Transaction submission follows First-commit-wins (FCW) rules, that is, if two transactions update the same data, the transaction submitted earlier will be successfully allowed while that submitted later will be abandoned. SI has the following two major characteristics. (1) It avoids SQL defined in ANSI (ANSI, 1992; Revilak et al., 2011), because the RLV rules of SI can ensure that all the data read by transactions have already been committed, eliminating reading "dirty" data and non-repeatable reading; moreover, FCW rules ensure that the data to be modified by each successfully-submitted transaction, during its operation period, will not be modified by other transactions, avoiding "lost update". (2) With SI mechanism, the "read" operation will be never delayed by other concurrent transaction "write" operations and it will never delay the read and write operations of other transactions, thus it enjoys higher throughput rate and it is especially preferred to read data intensive environment (Revilak et al., 2011; Cahill et al., 2008).

The characteristics of SI endow it with good consistency and high transaction processing efficiency, thus SI is generally applied to concurrency control of all
kinds of transactions. However, SI method embraces transaction serializability anomaly (Revilak et al., 2011; Cahill et al., 2008, 2009; Fekete et al., 2004, 2005, 2009; Jorwekar et al., 2007), i.e., good consistency can be ensured when multiple transactions are separately executed while inconsistency may arise when they are simultaneously executed.

Many solutions have been put forward to solve SI transaction serializability anomaly, such as static analysis method (Fekete, 2005; Jorwekar et al., 2007), dangerous-structure determination method (Cahill et al., 2008, 2009) and centralized loop detection method (Revilak et al., 2011). With static analysis method, Static Dependency Graph (SDG) is constructed when the possible dependent relationship between static analysis applications is designed and, if dangerous structure is shown in the graph (i.e., There are dependencies among three transactions, Ti, Tj and Tk.), the application code of dangerous structure must be modified and WW dependency will be introduced between these applications, in order to solve the SI transaction serializability anomaly but this method can not be applied in random transaction environment. Dangerous-structure determination method can be used to determine dangerous structure between transactions in operation period and, if any dangerous structure arises, corresponding transaction will be abandoned. This method can solve the transaction serializability anomaly and it is suitable in random transaction environment but it can abandon many transactions that should not be abandoned. With centralized loop detection method, central server is utilized to record all the Transaction Dependency Graphs (TDG) and all data lock tables through which the dependencies between transactions can be determined and the dependence graphs can be changed in timely manner according to the dependencies; in addition, this method can help to find the loops in graphs when transactions are submitted, if there is indeed loop, corresponding transaction will be abandoned to eliminate the loop. This method matches well with random transaction environment and it can accurately abandon transactions which effectively solves SI transaction serializability anomaly, thus it has been widely recognized.

The good consistency and high-efficiency transaction processing of SI enable SI to be widely used in cloud computing environment (Perg and Dabek, 2010; Zhang and De Sterck, 2010, 2011). However, SI transaction serializability anomaly arising in cloud computing environment impacts the data consistency in transaction implementation process. The above three methods can solve SI transaction serializability anomaly but they are proposed specially for stand-alone environment and there has been no distributed algorithm corresponding to cloud computing environment.

In order to better solve SI transaction serializability anomaly in cloud computing environment, transaction-dependent loop distributed detection method is proposed, this method integrates many technologies related with transaction-dependency distributed discovery, construction of distributed TDG and distributed detection algorithm of transaction dependency loop, thus it overcomes the difficulty in construction of TDG and transaction-dependency loop detection in cloud computing environment, so as to realize SI transaction serializable execution in cloud computing.

**DESIGN IDEAS**

Transaction dependency refers to the relationship of transaction operation conflicts. If two transactions, namely T<sub>m</sub> and T<sub>n</sub>, operate on the same data item x, at least one transaction will write data item x which means T<sub>m</sub> and T<sub>n</sub> depend on each other. Under SI mechanism, transaction dependency is divided into the following three varieties:

- **T<sub>m</sub> writ</sub>T<sub>n</sub>**: When T<sub>m</sub> writes data item x (for x<sub>m</sub> version) and then T<sub>n</sub> reads x<sub>m</sub> no new versions of data X arise during the period from T<sub>m</sub> generates x<sub>m</sub> to T<sub>n</sub> reads x<sub>m</sub>

- **T<sub>m</sub> writ</sub>T<sub>n</sub>**: When T<sub>m</sub> writes data item x (for x<sub>m</sub> version) and then T<sub>n</sub> writes data item x (for successor version of x<sub>m</sub>) there is no other version between x<sub>m</sub> and x<sub>n</sub>

- **T<sub>m</sub> r</sub>T<sub>n</sub>**: T<sub>n</sub> reads x<sub>n</sub> and then T<sub>m</sub> writes data item x, x<sub>n</sub>’s successor version x<sub>n</sub> is produced

It is the dependency between SI transactions that causes SI transaction serializability anomaly in SI mechanism (Revilak et al., 2011; Cahill et al., 2008, 2009; Fekete et al., 2004, 2005, 2009; Jorwekar et al., 2007) with SI writing skew as an example:

Suppose X and Y are two accounts in a bank, with premise of X+Y>0 and initially X<sub>0</sub> = 50 and Y<sub>0</sub> = 80. With SI mechanism, transaction T<sub>i</sub> reads X<sub>i</sub> and Y<sub>i</sub>, since X<sub>i</sub>+Y<sub>i</sub> = 130, X<sub>i</sub> minus 100 leads to new version value X<sub>i</sub> = 50 and it still complies with the constraint X+Y>0; Similarly, concurrent transaction T<sub>j</sub> reads X<sub>j</sub> and Y<sub>j</sub> and Y<sub>j</sub> minus 120 produces a new version value Y<sub>j</sub> = 40, then it also complies with the constraint X+Y>0. Adya model (Adya, 1999; Adya et al., 2000) (R<sub>n</sub>, W<sub>n</sub> and C<sub>n</sub> means the operations of respective write and submit of, respectively).
transaction \( i \), with the subscript \( i \) meaning transaction No., \( d \) represents data item \( d \) of version \( i \) which means corresponding transaction No.) is adopted to describe the concurrent operation history as follows:

\[
H_i: \ r(X_{i.5})r(X_{i.5})r'(Y_{i.80})w(X_{i.5})w(X_{i.5})c(Y_{i.40})c_2
\]

The final result of operation sequence \( H_i \), \( X+Y \rightarrow 90 \) violates the constraint \( X+Y>0 \).

TDG is an effective tool to determine the transaction serialization and it takes one transaction from transaction operation history as vertex and the dependencies between transactions as sides to form a directed graph. If there is a loop in TDG over transaction operation history, then the operation history cannot be serializable. For example, Fig. 1 is TDG of operation history \( H_i: r(Y_{1.5})r(X_{1.5})w(Y_{1.5})r(Z_{1.5})w(X_{1.5})r(Y_{1.5})c(X_{1.5})c(Y_{1.40})c_2 \). In \( H_i \), transaction \( T_1 \) reads \( Y_{1.5} \), then transaction \( T_2 \) is submitted, data version \( Y_{1.5} \) is produced, so, \( T_1 \) and \( T_2 \) embrace dependency relationship: \( T_1 \rightarrow wT_2 \). In Fig. 1, there is a directed side (rw) from \( T_1 \) to \( T_2 \); when transaction \( T_1 \) is submitted, data version \( Y_{1.5} \) is produced and then transaction \( T_2 \) reads \( Y_{1.5} \); so, \( T_1 \) and \( T_2 \) embrace dependency relationship: \( T_1 \rightarrow wT_2 \) and there is a directed side (wr) from \( T_1 \) to \( T_2 \) in Fig. 1. Similarly, there are two directed sides (rw) from \( T_1 \) to \( T_3 \) and one side (rw) from \( T_2 \) to \( T_3 \). There are three loops: \( T_1 \rightarrow wT_2 \rightarrow T_3 \rightarrow wT_1 \) and \( [T_1 \rightarrow T_2 \rightarrow T_1] \) in Fig. 1, showing that the operation history cannot be serializable.

Since only the existence of directed loop shall be determined, so TDG can be simplified, then the sides toward the same direction can be removed and the sides enjoy no affiliated dependency relationship. Figure 2 is the simplified diagram of Fig. 1.

In a centralized environment, the directed loops can be determined as follows. At first, TDG is empty. When a transaction needs submitting, the management system adds this transaction and its depending sides into the TDG, then the existence of loops is detected; if there is any loop, this transaction and its depending sides will be abandoned.

In cloud computing environment, transactions and their dependency show the following characteristics:

- Data access operations of each transaction are distributed to multiple nodes. For example, in Fig. 3, operations of transaction \( T_1 \) are distributed in Server 1 and Server 2; those of transaction \( T_2 \) are distributed in Server 2 and Server 3.
- In each node, only some transactions access the data in this node. For example, in Fig. 2, three transactions (\( T_1 \), \( T_2 \), and \( T_3 \)) access the data in Server 1.
- The dependency between two transactions exists in multiple nodes. For example, in Fig. 2, \( T_1 \) and \( T_2 \) operate in both Server 1 and Server 2, so the two transactions may show dependency in both Server 1 and Server 2.
No overall TDG exists in any node and each node only operates the dependency of some transactions allocated on it, without knowing the dependency of transactions in other nodes. For example, in Fig. 2, Sever 3 can only know the dependency of T1, T3, and T4 in Sever 3 but does not know the dependency of transactions existing in Sever 1 and Sever 2.

In view of the above characteristics, distributed detection method of transaction-dependency loop is proposed, including the following steps. (1) Distributed transaction dependency table is designed to store the transactions and their dependency in nodes; (2) Transaction dependencies are transferred between nodes to build more extensive dependencies; (3) Loop-oriented judgment is conducted and the judgment results are in transmitted between nodes.

**DESCRIPTION OF TRANSACTION DEPENDENCY AND CONSTRUCTION OF DISTRIBUTED ALGORITHM**

A transaction dependency table is presented in this paper to record dependencies between transactions (Fig. 4): Header contains the record of transactions set and each transaction Ti is in charge of an Inner Transaction Set (ITSi) and an Outer Transaction Set (OTSi) which describes the dependency between transactions; ITSi consists of transactions in Ti, while OTSi is composed of transactions fanned out from Ti.

For example, Transaction dependencies shown in Fig. 5 are simplified as those shown in Fig. 6.

In distributed environment, transaction dependency tables are distributed to all nodes for description and storage. For example, Fig. 7 shows one type of transaction dependency table corresponding to Fig. 5.
distribution of Fig. 6. After the transaction dependency tables are distributed to all nodes, each node only holds some of the transactions and their dependencies, as is shown in partial dependency graph. For example, the partial dependency graph of transaction dependency table on each node in Fig. 7 is shown in Fig. 8.

In order to locate transactions into nodes, hash method is adopted to match transactions on corresponding nodes, i.e., the node number = hash (transaction identifier).

FIGURE 7: DISTRIBUTION OF TRANSACTION DEPENDENCY TABLES IN NODES OF FIG. 6

Fig. 8: Partial dependence graphs corresponding to Fig. 7 Transaction dependency table

TRANSACTION DEPENDENCY LOOP DISTRIBUTED DETECTION ALGORITHM

Principles of loop detection: The basic principles to detect transaction dependency loops are as follows. The transaction initiating loop detection send detection messages (whose content is the identifiers of corresponding transactions) to all its fanned out transactions, or, if not fanning out some other transactions, will return to the original fanning in transactions; the transactions that have received returned messages will determine whether all the fanned-out transactions sent back messages, if they do, the returned messages will be sent back to corresponding fanning-in transactions.

So, according to the transferring of messages:

1. If the transaction launching loop detection sends detection messages and receives the same messages, it means this transaction locates in the loop. For example, in Fig. 9, the transaction T1 initiates loop detection and then, from fanning-in transaction T5, receives loop detection message it sends, so T1 is located in the loop.

2. If the transaction launching loop detection sends detection messages and then receives messages returned by all fanned-out transactions, it means this transaction is not located in the loop. For example, in Fig. 10, T9 initiates loop detection and receives the messages returned from T4 and T10, so T9 is not located in the loop.
For the transaction not initiating detection, if it receives the same detection messages for twice from the same fanned-in transaction, it means that this transaction is located in the loop. For example, in Fig. 11, after T2 initiates detection, T6 receives T2's detection messages for twice from T3, so T6 is located in the loop.

In case 1, the initiative transaction shall be abandoned to break the loop composed by this transaction; in case 2, initiative transaction can be submitted; in case 3, the first transaction existing in loop and not yet been committed shall be abandoned, break the loop composed by this transaction and eventually it can be determined that the initiative transaction is not located in the loop.
**Distributed algorithm:** According to the above principles, transaction dependency loop detection algorithm consists of initialization algorithm, message-reception algorithm, forward-processing algorithm and backward processing algorithm.

**Initialization algorithm:** When the node N receives T₀’s detection request, it initiates loop detection operation, according to the algorithm below:

**Algorithm 1: Initialization algorithm**

1. This node is marked as the source node;
2. For T₀’s all fanned-out transactions T
3. Host node H of T is calculated;
4. The messages are sealed according to the following format: <type = “forward”, initiating transaction = "T₀", sending transaction = "T" >;
5. The messages are sent to H;

**Message-reception algorithm:** Each node will receive two kinds of messages: the messages for loop detection and returned messages due to no existence of loop. Correct algorithm corresponding to the message type is used to process the messages according to the following algorithm.

**Algorithm 2: Message-reception algorithm**

1. If the message belongs to “forward” type;
2. Forward-processing algorithm is utilized;
3. else
4. Backward processing algorithm

**Forward-processing algorithm:** Each node, after receiving loop detection message, gets relay transaction from “target transaction” field of the messages and then gets all the fanned-out transactions of the relay transaction from transaction dependency table of this node and forwards the messages to them. The specific algorithm is as follows:

**Algorithm 3: Forward-processing algorithm**

1. If (target transaction = T₀)
2. “Loop with transaction existence” is output
3. Else
4. If (target transaction has forwarded T₀’s messages and not yet submitted them)
5. Target transaction is abandoned
6. Sent-out Ts is extracted from the messages
7. Host node (H) of Ts is calculated
8. The messages are sealed according to the following format: <type = backward, initiating transaction = T₀, target transaction = Ts>
9. The messages are sent to H
10. Else
11. Ts is obtained from “target transaction” field of the messages
12. All the fanned-out transactions are obtained from T’s OTS
13. If (T contains fanned-out transactions)
14. The fact that T sends initiative transaction T₀’s message is recorded
15. For T’s all fanned-out transactions T
16. The messages are sealed according to the following format: <type = forward, initiating transaction = T₀..., sending transaction = T₄, target transaction = T₈>
17. The messages are sent to H
18. Else
19. Sending transaction Ts is extracted from the messages
20. Host node (H) of Ts is calculated
21. The messages are sealed according to the following format: <type = “backward”, initiating transaction = T₀, target transaction = Ts>
22. The messages are sent to H
Backward-processing algorithm: Each node, after receiving “returned message”, gets relay transaction from “target transaction” field of the messages and then it is determined whether those fanned-out sides of relay transactions that have sent T0’s messages receive returned messages and if they do, the node will forward the returned messages to its fanned-in transactions. The specific algorithm is as follows:

Algorithm 4: Backward-processing algorithm
1. $T_s$ is extracted from target transaction of the messages
2. If (fanned-out sides of relay transactions that have sent T0’s messages receive returned messages)
3. If (T1 = T0)
4. “Loop with transaction existence” is output
5. Else
6. All the fanned-in transactions $T_s$ are obtained from the Td’s ITS
7. For Td’s all fanned-in transactions $T_s$
8. Host node (H) of Td is calculated
9. The messages are sealed according to the following format: $<type = \text{backward}, \text{initiating transaction} = T0, \text{target transaction} = T_s>$
10. The messages are sent to H

EXPERIMENTS AND PERFORMANCE EVALUATION

In order to evaluate transaction-dependency loop distributed detection method (hereinafter referred to as PDCC), a SI-oriented serializability distributed algorithm integrating Percolator’s method of transaction processing (denoted as Percolator) is defined, achieving cross-line cross-table transaction processing in HBase. Percolator, a system Google uses to handle incremental webpage index, realizes distributed transaction by two-phase submission and optimistic locking. Moreover, Percolator’s transaction processing method is modified, so as to realize transaction-dependency loop distributed detection.

To evaluate the effects of making SI serializable, we need a benchmark that is not already serializable under SI. The SmallBank benchmark (Alomari et al., 2008) was designed to model a simple banking application involving checking and savings accounts, with transaction types for Balance (Bal), Deposit/Checking (DC), Withdraw-from-checking (WC), Transfer-to-savings (TS) and Amalgamate (Amg) operations. Each of the transaction types involves a small number of simple read and update operations. The static dependency graph for SmallBank is given in Fig. 12, where the double arrows represent write-write conicts and the dashed arrows represent read-write conicts. It can be seen by inspection that there is a loop: Bal→WC→TS→Bal.

Experimental environment: Our testing infrastructure had 126 machines on 4 racks connected by Gigabit Ethernet switches. Intra-rack bisection bandwidth was ~14 Gbps while inter-rack bisection bandwidth was ~6.5 Gbps. Each machine had two 2.4 GHz Intel Xeon CPUs, 4 GB of main memory and two 7200RPM SCSI disks with 200 GB each. Machines ran Red Hat Enterprise Linux AS 4 with kernel version 2.6.9.

We adapted the original relational data model defined by SmallBank to the Bigtable data model, so that, the application data can be stored into HBase. The relational data model of SmallBank comprises three tables that are accessed by these transactions. To adapt this data model to Bigtable, we combine the three tables: Account (Name, CustomerID), Savings (CustomerID, Balance) and Checking (CustomerID, Balance) into one Bigtable named “Bank”. Each of the original tables is stored as a column family, whose primary key is “CustomerID”. Before each experiment, we populate 1,440,000 customer information and 100 records for each customer.

Performance evaluation: Firstly, transaction submission rate and transaction abandon rate of loop detection method is evaluated and those of Percolator are also measured, then they are comparatively analyzed. In this experiment, 3 HBase servers to which all of the data are distributed are adopted and then the quantity of concurrent client machines are gradually increased in order to increase the amount of concurrent transactions [also known as the Multiprogramming Level (MPL)] (Revilak et al., 2011). The test results are not shown in Fig. 13 and 14.

As is shown in the figures, when the system’s transaction processing ability is not fully saturated, with
the increase of concurrent transactions, the two systems’ transaction submission rate displays logarithmic growth and their transaction abandon rate increases exponentially, for the reason that, with the increase of concurrent transactions, the possibility for many transactions to process the same data set increases, so does transactions abandon rate according to the principles of FUW. What’s more, Percolator is non-serializable transaction processing method, some transactions resulting in abnormal data are also submitted, however, PDCC will abandon these abnormal transactions, so Percolator’s transaction submission rate is higher than PDCC’s while its transaction abandon rate is lower than PDCC’s.

Fig. 13: Transaction submission rate

Fig. 14: Transaction abandon rate

Then, PDCC’s scalability is evaluated. When the transactions are over-loaded, the system’s transaction submission rate will decrease rapidly, so, in this experiment, only the maximum transaction submission rate at different server scales is recorded. The test results as shown in Fig. 15 which demonstrates that both of them enjoy good scalability but, with the increased server quantity, transaction throughput also increases, followed by more fierce transaction competition; therefore, with the increased server quantity, the enhancement of transaction submission rate gradually weakens.

CONCLUSION

For SI transaction serializability anomalies in cloud computing environment, this paper puts forward a distributed transaction dependency loop detection method which overcomes some affairs in the cloud computing environment, such as building dependence graph (TDG), detecting transaction dependency loop etc., by the following three technology: (1) In each node stores, we design a distributed transaction dependency table to record transaction and transaction dependency relations; (2) More extensive transaction dependency relations are transferred between nodes; (3) Detecting transaction dependency loop and transferring the result among nodes. Based on the transaction concurrency control method: Percolator and open source cloud database: HBase, we Implementation and test in the PDCC. The experimental results show that PDCC solves the SI transaction serializability anomaly problem and has better performance and scalability.
ACKNOWLEDGMENTS

The authors would like to thank for the support by National Basic Research Program of China (973 Program) under Grant No. 2011CB302601, National High Technology Research and Development program of China (863 Program) under Grant No. 2011AAA1A0202, Science and technology program of Hunan Province under Grant 2013FJ4335 and 2013FJ4295 and the constructing program of the key discipline in Huazhuan University.

REFERENCES


