DESIGN, IMPLEMENTATION AND PERFORMANCE ANALYSIS OF CONCURRENCY CONTROL ALGORITHM WITH ARCHITECTURE FOR TEMPORAL DATABASE

Jaypalsinh A. Gohil[1] and Dr. Prashant M. Dolia[2]

[1]Research Scholar, Department of Computer Science and Applications,
MK Bhavnagar University, Bhavnagar, Gujarat - India

[2]Asso.Prof. & Research Guide, Department of Computer Science and Applications,
MK Bhavnagar University, Bhavnagar, Gujarat - India

ABSTRACT

Concurrency control mechanism is one of the critical factors which affect the overall performance of database system. The conflicting situations are normal phenomena in multiuser distributed transaction environment. Concurrency situation is not desirable for time dependent and time critical large database applications; it must be efficiently controlled and managed to ensure consistency of database system. The time aspect of data can be easily represented by temporal database, it allows you to associate time element with data through temporal validity support. Concurrency control mechanism elevates to a new height when it is implemented on temporal database especially for multiuser distributed transaction environment and it requires a special treatment. For consistency requirement to be true different user sessions of a distributed transaction running simultaneously must produce meaningful and consistent results otherwise it leaves database into inconsistent state. Traditionally timestamp and locking based approaches are used to encounter the concurrency problem. This research study proposes an efficient concurrency algorithm which is a mixture of both timestamp and locking approach suitable for temporal database environment where more than one users of distributed transaction tries to access the common resource. The scope of research involves design, implementation, experimental study and performance analysis of proposed algorithm in comparison with pessimistic and optimistic concurrency control mechanisms. The proposed algorithm is implemented through trigger using Oracle 12c which supports temporal database consistency through temporal validity support and efficient locking mechanism. Concurrency and locking situation is graphically represented in the experiment with the help of Oracle 12c enterprise manager.

KEYWORDS

CC, DB, TDB, TIMESTAMP, LOCK, PESSIMISTIC, OPTIMISTIC, SCN, JAG_TDB_CC, Oracle 12c.

1. INTRODUCTION

The main objective of concurrency control (CC) is to provide consistency for any database system. The concurrency must ensure that any simultaneous execution of distributed transactions produces the same result as a sequential execution [1]. An execution is serializable if it is computationally equivalent to a serial execution. A serial execution of two or more transactions means that all operations of one transaction are executed before any operation from another transaction can execute. Since serial executions preserve consistency by definition and every serializable execution is equivalent to a serial one, every serializable execution also preserves consistency [2]. The CC takes new dimensions when applied to temporal database (TDB). The objective of TDB is the management of the data history.

DOI: 10.5121/ijdms.2016.8503

Generally time dependent and time critical real time applications of database technology are temporal in nature. Examples include financial applications such as portfolio management, stock exchange, accounting, and banking; record-keeping applications such as personnel, medical-record, and inventory management; scheduling applications such as airline, train, and hotel reservations and project management; and scientific applications such as weather monitoring. The above mention applications can be classified as temporal databases applications, which record time-referenced and time-relevant data [3].

The two main concurrency control mechanism available for any database system can be classified into either pessimistic or optimistic. The pessimistic approach prevents execution of concurrent transactions in case of conflict. As a contrast the optimistic concurrency control mechanism allows the concurrent transactions to proceed at the time of conflict with a risk of restarting them [4,5]. Locking is an efficient mean to provide concurrency control in database system environment. Lock based concurrency control mechanism works on simple lock mechanism to control the concurrent access to the data item. If lock is acquired by the transaction then and then only permission is given to access the data item [6].

The working mechanism of pessimistic concurrency control avoids any concurrent execution of transactions as soon as potential conflicts between these transactions are detected. Alternately, optimistic concurrency control allows such transactions to proceed at the risk of having to restart them in case this suspected conflict actually occurs. In optimistic concurrency mechanism the concentration is on the fact that the resource should not be blocked for longer period of time [7].

The major drawbacks of pessimistic locking approach are frequent lockouts and deadlocks. The optimistic locking provides alternative solution to the problems. Optimistic locking does not lock records when they are read, and proceeds on the assumption that the data being updated has not changed since the read. Since no locks are taken out during the read, the deadlocks are eliminated since users should never have to wait on each other's locks. The Oracle database uses optimistic locking by default [7].

However in an experimental study when the optimistic locking approach for temporal database environment was checked for efficiency and performance it was not up to the mark as per the requirement of temporal database systems and needed improvement [8].

The historical data can be represented in a systematic manner using temporal database [9,10]. Temporal database provides mechanisms to store and manipulate time-varying information. Temporal databases encompass all database applications that require some aspect of time when organizing their information. So consistency in temporal database is a critical area needs to be addressed by database administrator. Oracle introduced Oracle Database 12c on June 25, 2013, which is considered to be the important architectural transformation in the legacy of the world's leading database in its 25 years with respect to market presence and dominance [11].

Oracle 12c supports temporal database consistency through temporal validity support and efficient locking mechanism. Oracle enterprise manager of Oracle 12c provides graphical view of distributed transaction and various user sessions with locking and unlocking details.

2. RELATED WORK

In past two decades the researchers have focused on time-referenced or temporal data and tried to develop concepts, tools and techniques that better suits for temporal data management. The recent research guided by the observations found that most of time-centric real time databases applications contain temporal data. The traditional database technology lacks the support to such

databases application especially when we address the concurrency aspect. Temporal database system differs from conventional database systems in way that data stored in a temporal database is different from the data stored in non-temporal database because time element attached to the data expresses when it was valid or stored in the database.

Concurrency control is a critical aspect for any database systems. Many researchers tried to develop many protocol for achieving the serializability. The basic concurrency approaches used by these protocols are locking, timestamp and multiple versions [12, 13, 14].

Various concurrency control schemes are discussed and all of them follow serializability concept. Conflicting situation can be resolved by delaying or aborting the transactions. Locking protocols, timestamp, timestamp validation techniques and multiversion are general concurrency control schemes [15].

The work proposed for concurrency control in database systems identifies various classes of concurrency control approaches and done a brief survey focused on designing concurrency control algorithms which are flexible [16].

In the paper titled Concurrency Control: Methods, Performance and Analysis the author has analyzed the performance of locking model. The study shows the parameters which can lead to degradation in performance. The paper also highlights comparative performance analysis of concurrency control methods applicable to standard locking, restart-oriented locking methods and optimistic concurrency control approach with reference to conclusions of past simulation results [17].

The concept of multiversion concurrency-control is explained relative to one-copy serializability. As per the study the transactions are one-copy serializable in a multiversion database. In the research study the application of three multiversion concurrency control algorithms wherein one algorithm user timestamps, one uses locking and final one uses a combination of locking with timestamps [18].

In another research related to concurrency control through multiple data versions examined the performance and efficiency of concurrency control approach. In the study the authors have limited the degree of parallelism resulting from multiversion approach. The approach also shows space-parallelism trade-off with its pros and cons [19].

The work presented by Kung and Robinson focused on optimistic concurrency control approach. In the research study two categories of non-locking concurrency control approaches are proposed. Since the methods are based on optimistic approach, there is an assumption that conflicts between transactions will not occur. The main point highlighted in the work suggests that locking is not always feasible and creates an overhead, so locking approach must be used in worst situation only [12].

Application of concurrency control for temporal database requires a special treatment because temporal database application stores time-relevant historical data. Some of the concurrency control algorithms based on pessimistic approach have tried to find solution of concurrency problem especially for temporal database by improvement in performance. These algorithms tried to minimize the conflicting situation by use of prior knowledge or by reduction of granule size. Studies have suggested that these algorithm are not in-line with desired objectives and particularly not suitable for temporal database environment [20].

The pessimistic approach for concurrency control has some limitations, whereas optimistic approach is more reliable because it improves the degree of parallelism and it is based on the

assumption that conflicts are rare. The research work of Achraf Makani and Rafik Bouaziz has proposed an optimistic algorithm suitable for temporal database environment. The proposed algorithm cantered on the temporal specifications to reduce the granule size and then to minimize the conflict degree. The algorithm is also capable of detecting conflicting situations as soon as possible. In the research they have also introduced and used end of transaction marker technique. The main aim of study is to reduce to the maximum the period during which resources are locked. According to their conclusion the optimistic approach is more suitable for temporal database as compared to pessimistic approach which has some limitations [20].

3. PROPOSED WORK

The research work proposes JAG_TDB_CC as an effective and efficient concurrency control algorithm along with its architecture which is suitable for temporal database environment. The proposed work is a blend of timestamp and locking approach. The study also combines the merits of pessimistic and optimistic concurrency control mechanisms by eliminating the dangers of deadlocks, frequent lockouts and long waiting times among the various user sessions of distributed transaction. The study also uses a unique System Change Number (SCN) for individual row as an additional concurrency parameter for condition checking.

To implement the JAG_TDB_CC algorithm initially we have designed five procedures, the description of each are as follows:

Procedure_1: PRO_TRANDETAILS: This procedure finds the details of current running transactions and sessions and stores those details into TRANDETAILS table.

Procedure_2: PRO_VST(MVST): This procedure outputs and stores start time of active session of a distributed transaction. The output value is stored in MVST variable which will be used in a trigger for condition checking.

Procedure_3: PRO_MINVST(MMINVST): This procedure outputs and stores start time of oldest running session of distributed transaction. The output is stored in MMINVST variable and will be passed to trigger for condition checking.

Procedure_4: PRO_MAXSCN(MMAXSCN): This procedure gives maximum values of System Change Number (ORA_ROWSCN) form the table and passes it through MMAXSCN variable to trigger for condition checking.

Procedure_5: PRO_LOCKSESSION(LC): This procedure finds locking details among all running user sessions and stores them in LOCK_DETAILS table. The procedure also stores lock count (number of locks held by various sessions) into LC variable and passes it to trigger for condition verification.

3.1. PROPOSED JAG_TDB_CC ALGORITHM

begin

Call PRO_TRANDETAILS;

Call PRO_VST(MVST);

Call PRO_MINVST(MMINVST);

Call PRO_MAXSCN(MMAXSCN);

Call PRO_LOCKSESSION(LC);

```
        If Operation=INSERT then
                If MVST>MMINVST && MVST>MMAXSCN && LC>1 then
                        Error: "Concurrency error for Insert operation"
                Else
                        Msg: "Record/s are successfully Inserted"
                End If;
        End If;

        If Operation=UPDATE then
                If MVST>MMINVST && MVST>MMAXSCN && LC>1 then
                        Error: "Concurrency error for Update operation"
                Else
                        Msg: "Record/s are successfully Updated"
                End If;
        End If;

        If Operation=DELETE then
                If MVST>MMINVST && MVST>MMAXSCN && LC>1 then
                        Error: "Concurrency error for Delete operation"
                Else
                        Msg: "Record/s are successfully Deleted"
                End If;
        End If;

end;
```

The proposed algorithm is designed and implemented through a trigger which denies or allows INSERT, UPDATE or DELTE operations among various user sessions of a distributed transaction after performing concurrency condition check. Whenever any user sessions of a distributed transaction say S1 (S1,S2,S3,...Sn) waiting in a session queue tries to perform either INSERT, UPDATE or DELETE operation on conflicting data object on which the lock already held by some other session the trigger performs concurrency checks and then either grants or denies the access based on concurrency rules specified in the proposed algorithm. The flow of algorithm logic is highlighted in the following diagram.

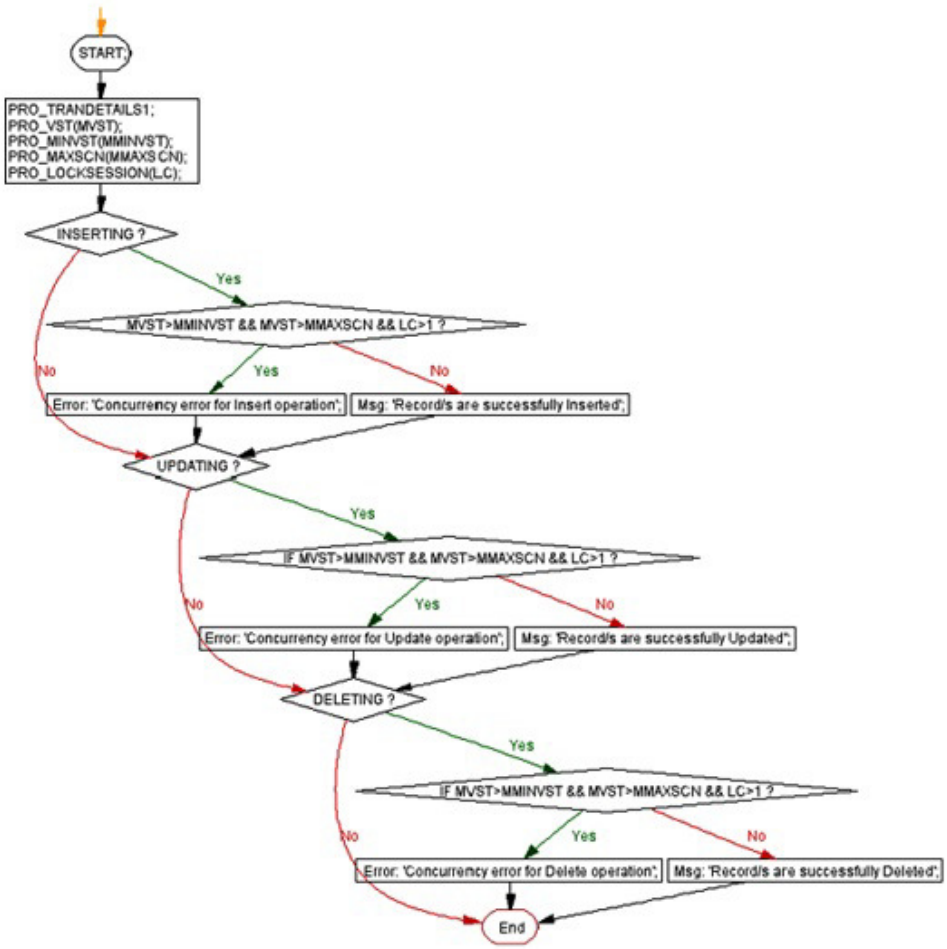


Figure 1. Flow Chart for Jag_Tdb_Cc Algorithm

If we analyse the main concurrency condition i.e. IF MVST>MMINVST && MVST>MMAXSCN && LC>1 of proposed algorithm it comprises of three parts. In the first phase MVST>MMIMVST will check that the start time of currently active session must be grater then start time of oldest running session. In the second phase MVST>MMAXSCN the algorithm checks that the start time of currently active session must be grater then maximum values of SCN-System Change Number of targeted conflicting relation. So the first two phases of condition checking of the proposed algorithm follows a timestamp approach. The locking approach is used in third and final phase of condition where the proposed algorithm checks whether the lock count LC>1 or not. The LC (lock count) is nothing but numerical counter generated by a procedure which indicates the number of locks held by different user sessions on conflicting data object. For the concurrency condition to be true all three parts of the condition must be executed and evaluated together, so at last the trigger evaluates all three phases of condition together using && operator. Based on condition finally the trigger of proposed algorithm either denies or allows the access of conflicting temporal data object.

3.2 Proposed Architecture

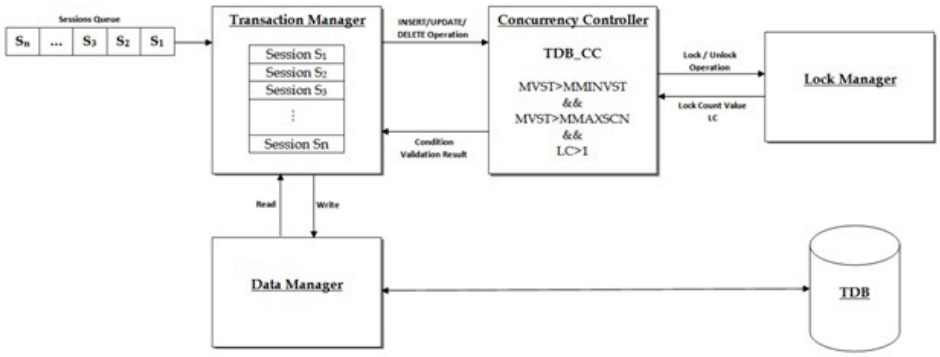


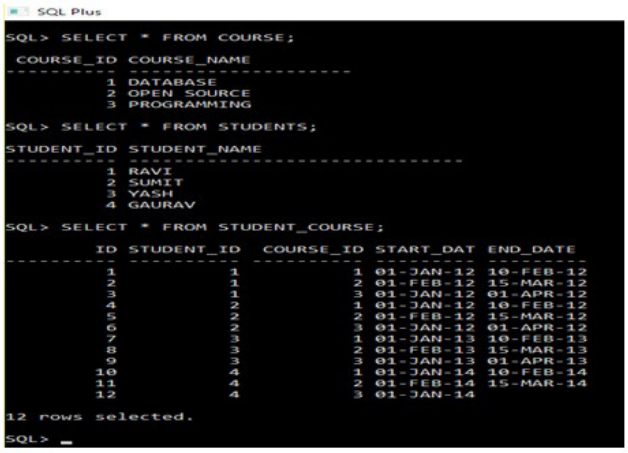
Figure 2. Proposed Concurrency Control Architecture

The figure 2 shows proposed concurrency control architecture. A transaction manager module allows more than one user sessions of a distributed transaction to operate concurrently and maintains sequence among them. Whenever any user session tries to perform either INSERT, UPDATE or DELETE operation on the shared temporal data object, the control is shifted to concurrency controller. Before verification of concurrency condition of proposed algorithm the concurrency controller consults with lock manager for lock/unlock operation. In turn the lock manager passes lock counter values LC back to the concurrency controller. Now concurrency controller performs condition checking as per concurrency rules specified in proposed algorithm. If a session passes the condition check successfully, the transaction manager sends read/write operation to data manager. Finally data is permanently stored in database by session's COMMIT operation.

4. EXPERIMENT AND RESULTS

4.1 Creation of Temporal Relation

The experiment starts with designing three relations namely COURSE, STUDENTS and third temporal relation STUDENT_COURSE by using the PERIOD FOR clause at the time of creation of relation STUDENT_COURSE using following query [20, 21]. The following figure 3 shows the structure of table with inserted records:



```
SQL Plus
SQL> SELECT * FROM COURSE;

 COURSE_ID COURSE_NAME
---------- --------------------
         1 DATABASE
         2 OPEN SOURCE
         3 PROGRAMMING

SQL> SELECT * FROM STUDENTS;

STUDENT_ID STUDENT_NAME
---------- --------------------------------
         1 RAVI
         2 SUMIT
         3 YASH
         4 GAURAV

SQL> SELECT * FROM STUDENT_COURSE;

        ID STUDENT_ID  COURSE_ID START_DAT END_DATE
---------- ---------- ---------- --------- ---------
         1          1          1 01-JAN-12 10-FEB-12
         2          1          2 01-FEB-12 15-MAR-12
         3          1          3 01-JAN-12 01-APR-12
         4          2          1 01-JAN-12 10-FEB-12
         5          2          2 01-FEB-12 15-MAR-12
         6          2          3 01-JAN-12 01-APR-12
         7          3          1 01-JAN-13 10-FEB-13
         8          3          2 01-FEB-13 15-MAR-13
         9          3          3 01-JAN-13 01-APR-13
        10          4          1 01-JAN-14 10-FEB-14
        11          4          2 01-FEB-14 15-MAR-14
        12          4          3 01-JAN-14

12 rows selected.

SQL>
```

Figure 3. Structure of Three Tables Along with its Data

4.2 Session View at the Start of Experiment

Initially three distinct users sessions namely C##JAG1, C##JAG2 and C##JAG3 are invoked by experiment as shown in the below figure 4. The C##JAG1 is the owner of the STUDENT_COURSE temporal relation. The user C##JAG1 grants ALL permission on STUDENT_COURSE relation to other two users' namely C##JAG2 and C##JAG3 respectively. At the beginning of the experiment all three users' issues a SELECT command on STUDENT_COURSE shared temporal relation.

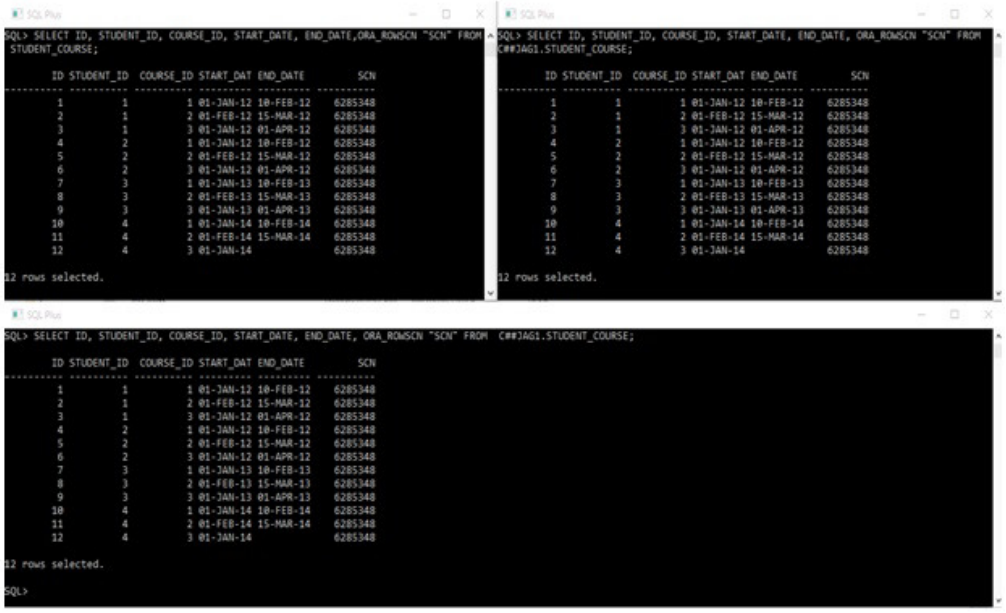


Figure 4. Three User Sessions C##JAG1, C##JAG2 And C##JAG3 at the Start of Experiment with SCN Value

4.3 Concurrency Control Checks for JAG_TDB_CC Algorithm

In the next phase of experiment the proposed algorithm is implemented by a trigger. Now the conceptual idea of proposed algorithm is materialized through trigger implementation. The following sequence of snapshots describes the working mechanism of proposed algorithm in different concurrency situations.

The above figure 5 highlights the concurrency control mechanism of trigger during update operation. If one user session is performing the update operation on locked row then trigger description restricts the update operation of other user session from modifying the same row. If all the users in our case C##JAG1, C##JAG2 and C##JAG3 tries to perform UPDATE operation on the same shared resource STUDENT_COURSE relation, only one user i.e. C##JAG1 is allowed to proceed with INSERT operation as per rules of algorithm meanwhile the algorithm denies the UPDATE operation of other two users C##JAG2 and C##JAG3 and gives them appropriate concurrency error message.

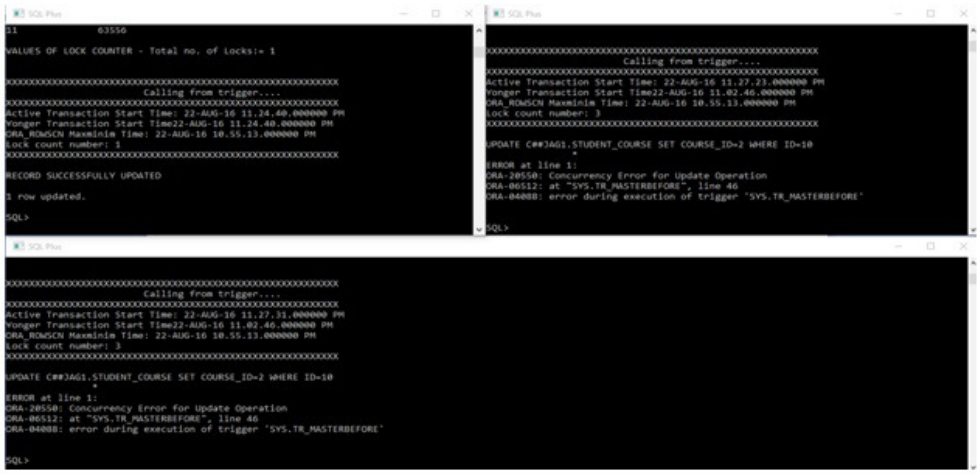


Figure 5. Concurrency Check for UPDATE Operation

Note that the algorithm only denies the user from accessing locked resource, instead of blocking it. So other users C##JAG2 and C##JAG3 are allowed to perform their normal non-conflicting operations without waiting for C##JAG1 to release the lock. So if lock is held by one user in our case C##JAG1 on the conflicting data object then on the same object other user sessions C##JAG2 and C##JAG3 cannot acquire the lock simultaneously until the session holding the lock i.e. C##JAG1 releases it. So ultimately no user session goes into waiting state and does not remain idle for indefinite time until lock is released. Since no user sessions are blocked they can perform their non-conflicting operations in normal manner without waiting for locked resource to be released which can be seen in the below figure 6.

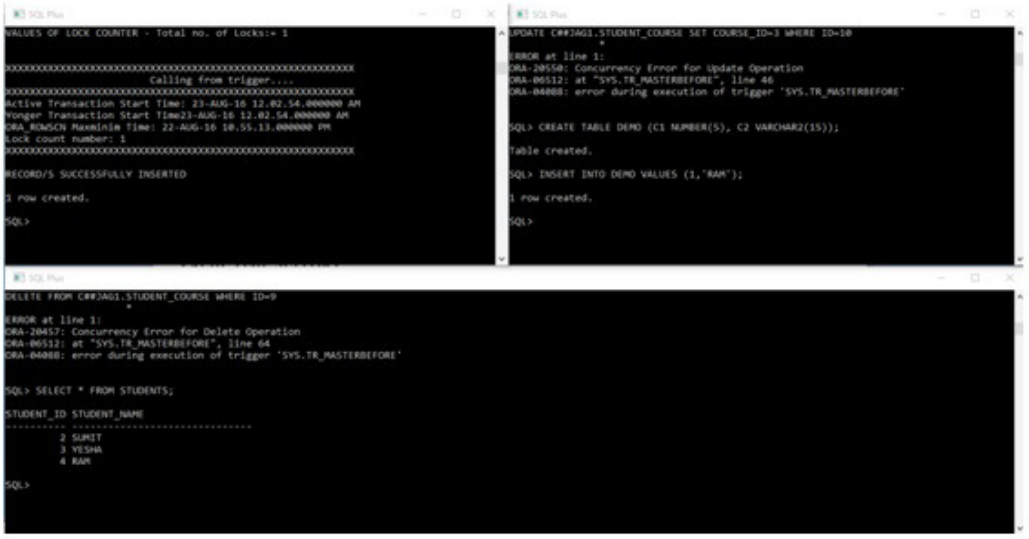


Figure 6. User Sessions Performing Non-Conflicting Operations

Apart from concurrency control checks on UPDATE operation, in our experimental study we have performed similar concurrency control checks on INSERT and DELETE operations and they yields the same results as per the rules of algorithm.

4.4 Graphical Representation of Concurrency Environment

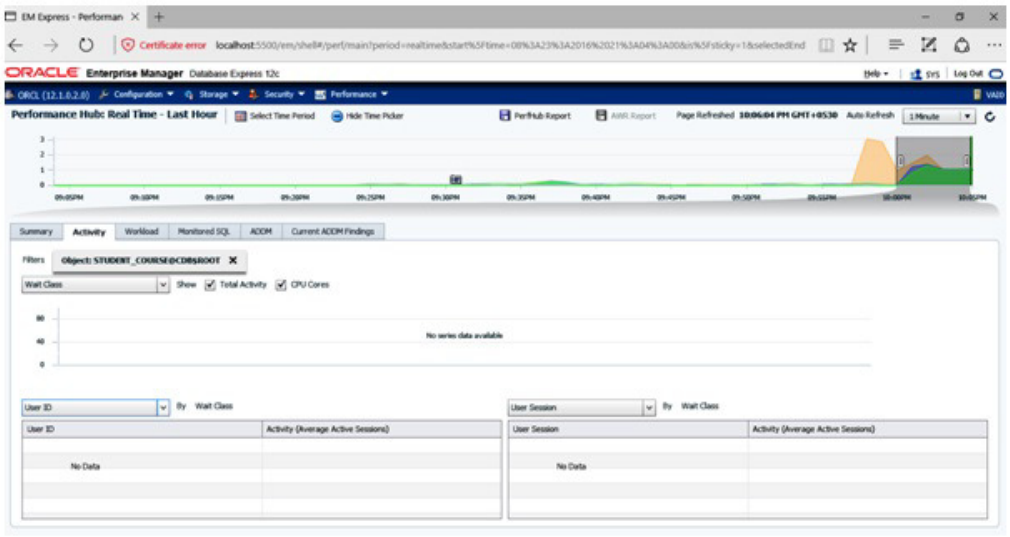


Figure 7. Enterprise Manager Showing Concurrency Environment

The above image graphically shows the concurrency environment for proposed experiment. It shows that the resource is not locked for other user sessions and they can execute their non-conflicting operations. As per proposed concurrency control method no concurrent locks are acquired by more than one user on conflicting object. So if lock is held by C##JAG1 user the other users i.e. C##JAG2 and C##JAG3 are waiting for their turn only denies the access of locked resource instead of blocking it. So one can say that C##JAG1 user's session is not blocking C##JAG2 and C#JAG3 user's respectively but only denies them the access of conflicting resource as a contrast to both pessimistic and optimistic locking experiments [7,8]. The following figure 8 confirms the correctness of algorithm by showing the details that no concurrent user sessions blocking each other in case of conflict.

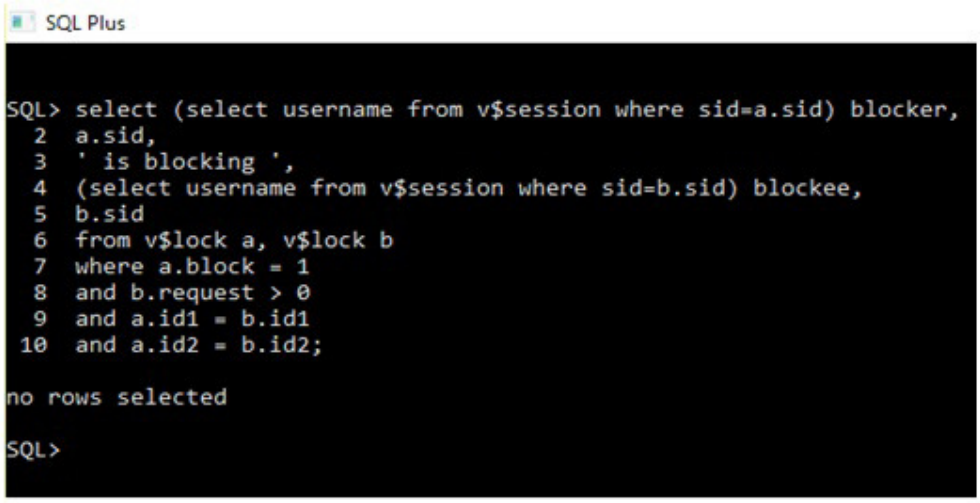


Figure 8. Who is Blocking Whom?

5. SESSION OUTLINE VIEW

The following table 1 represents session outline view for C##JAG1, C##JAG2 and C##JAG3 users showing step by step sequence of operations when three different users sessions tries to modify (UPDATE) the same resource roughly the same time. Here the session outline view is given for UPDATE operation, but the same concurrency control mechanism can be applied on INSERT and DELETE operations using a proposed algorithm.

Table 1. Session Outline View for UPDATE Operation

Time	Session 1 for user C##JAG1	Session 2 for user C##JAG2	Session 3 for user C##JAG3	Explanation
T1	GRANT ALL ON STUDENT_COURSE TO C#JAG2,C#JAG3;			The C##JAG1 user session grants all the permission on STUDENT_COURSE relation to two distinct users' C##JAG2 and C#JAG3.
T2	SELECT * FROM STUDENT_COURSE;	SELECT * FROM C##JAG1.STUDENT_COURSE;	SELECT * FROM C##JAG1.STUDENT_COURSE;	Initially three user sessions namely C##JAG1, C##JAG2 and C##JAG3 issue a select statement on STUDENT_COURSE relation owned by C##JAG1 user.
T3	UPDATE STUDENT_COURSE SET COURSE_ID=3 WHERE ID=10; 1 row updated…			Initially in session 1 the user C##JAG1 tries to update a row and trigger allows the update operation to C##JAG1 based on algorithm rules. Here observe that user session C##JAG1 is still running and yet not committed itself.
T4		UPDATE C##JAG1.STUDENT_COURSE SET COURSE_ID=2 WHERE ID=10; ORA-20550: Concurrency Error for Update Operation… CREATE TABLE DEMO (C1 NUMBER(5), C2 VARCHAR2(15)); INSERT INTO DEMO VALUES (1,'RAM');		Immediately user session 2 of C##JAG2 tries to perform same update operation on STUDENT_COURSE table but the user denied the access because it violates algorithm rules. Note that here algorithm only denies the user from accessing locked resource, instead of blocking it. So algorithm allows the user session C##JAG2 to perform its normal operation on non conflicting resources in our case create table and insert commands on DEMO table without waiting for conflicting resource to be released.
T5			UPDATE C##JAG1.STUDENT_COURSE SET COURSE_ID=2 WHERE ID=10; ORA-20550: Concurrency Error for Update Operation… SELECT * FROM C##JAG2.DEMO;	As a next step the session 3 of user C##JAG3 tries to update the same row currently locked by C##JAG1. Again algorithm mechanism denies the access and prevents the update operation of C##JAG3 user but user is still allowed to perform its normal non conflicting operations, in our case select operation on DEMO table.
T6	Commit;	UPDATE C##JAG1.STUDENT_COURSE SET COURSE_ID=2 WHERE ID=10; 1 row updated	SELECT * FROM C##JAG2.DEMO;	At this point C##JAG1 user issues a commit command which allows the C##JAG2 to successfully update the row as per algorithm description. Still at this point C##JAG3 can perform its normal operations.
T7		Commit;	UPDATE C##JAG1.STUDENT_COURSE SET COURSE_ID=2 WHERE ID=10; 1 row updated	At this moment the C##JAG2 user issues a commit statement so now lock which is being held by user C##JAG2 is released and granted to next user C##JAG3 which now can allowed to update the row according to the algorithm description.
T8			Commit;	Finally the user C##JAG3 commits its operation and releases the lock.

6. PERFORMANCE EVALUATION

The following table represents average waiting time statistics collected during experimental study of pessimistic, optimistic and proposed JAG_TDB_CC concurrency control approach [7,8].

Table 2. Session Waiting Time Statistics

	C##JAG1	C##JAG2	C##JAG3
Pessimistic	0.45	0.52	0.59
Optimistic	0.10	0.02	0.02
Proposed	0.05	0.01	0.01

The comparative performance analysis of pessimistic, optimistic and proposed concurrency control approach is highlighted in the below chart.

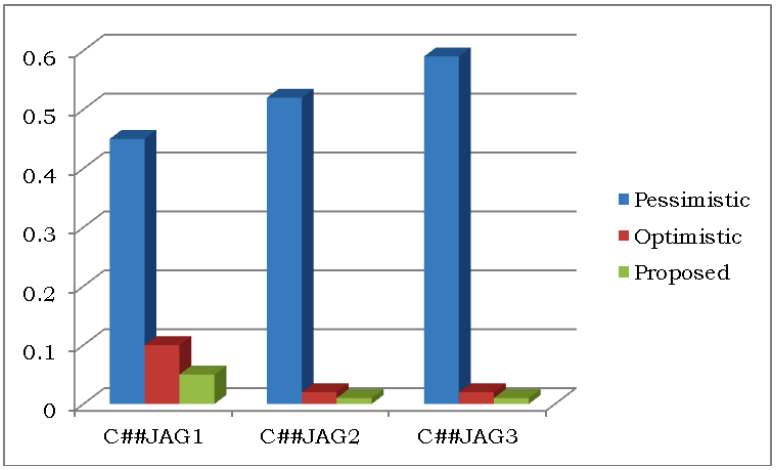


Figure 9. Performance Analysis

The performance evaluation gives the hindsight to the fact that the session waiting time for all three user sessions i.e. for C##JAG1, C##JAG2 and C##JAG3 are decreased significantly near to almost nil in proposed method as compared to traditional pessimistic and optimistic concurrency control approaches [7,8]. The statistical analysis also highlights the fact that in pessimistic approach the different user sessions remain idle in waiting state for longer period of time as compared to either optimistic or proposed concurrency control approaches. The performance analysis clearly indicates that proposed approach is best suited for concurrency control over either pessimistic or optimistic approaches.

7. CONCLUSION

Concurrency control is critical aspect for any database application. If concurrency situation is not efficiently tackled, it can leave a database into unstable state. Concurrency control touches a new horizon when it is applied on temporal database especially in multiuser distributed transaction environment. Timestamp and locking are two common approaches for concurrency control. To provide efficient concurrency control the proposed algorithm combines both timestamp and locking approaches together. The algorithm is applied through trigger which either denies or allows a user session of a distributed transaction to perform its operation based on the concurrency control rules specified in algorithm. In conflicting situations instead of blocking the user session the proposed algorithm denies the user from accessing the locked resource. In the third phase concurrency control mechanism of proposed algorithm denies the acquisition of simultaneous lock on conflicting resource. So if lock is already obtained by one user session on shared data object then no other user session can obtain the lock on the same resource

simultaneously until the session holding the lock released it. So the algorithm prevents other conflicting user sessions from going into waiting state and allowed them to remain idle for indefinite time until lock is released which was the case in pessimistic and optimistic concurrency control approaches [7,8]. Since no user sessions are blocked they can perform their non-conflicting operations in normal manner without waiting for locked resource to be released. The performance evaluation of proposed algorithm confirms that not only it overcomes the limitations of both pessimistic and optimistic concurrency control approaches but also it is the best alternative for temporal database environment. The future scope of proposed algorithm can be extended to time centred and time critical real time database applications for instance banking, stock exchange, personal and medical record keeping, airlines, train, bus and hostel reservation system, movie ticket booking and online shopping where time dependant distributed transactions are frequent.

REFERENCES

[1] Jaypalsinh A. Gohil, Dr. Prashant M. Dolia "Study and Comparative Analysis of Basic Pessimistic and Optimistic Concurrency Control Methods for Database Management System", International Journal of Advanced Research in Computer and Communication Engineering Vol. 5, Issue 1, January 2016.

[2] P.A. Bernstein and N. Goodman, "Concurrency Control in Distributed Database Systems," ACM Computing Surveys, Vol. 13(2), June 1981, pp. 186 - 221.

[3] C. S. Jensen, & al., (l998) "The Consensus Glossary of Temporal Database Concepts"—February l998 Version, O.Etzion, S.Jajodia, S.Sripada, (Eds.), Temporal Databases—Research and Practice, Lecture Notes in Computer Science, Vol. l399, Springer-Verlag, Berlin, pp. 367–405.

[4] K. P. Eswaran, J. N. Gray, R. A. Lorie, and I. L. Traiger, "The notions of consistency and predicate locks in a database system", Communications of the ACM, 19(11):624{633, November 1976.

[5] J. N. Gray, R. A. Lorie, G. R. Putzolu, and I. L. Traiger, "Granularity of locks and degrees of consistensy in a shared data base", In G. M. Nijssen, editor, Modeling in Data Base Management Systems, pages 365-395. North-Holland, Amsterdam, The Netherlands, 1976.

[6] J.A.Gohil, Dr.P.M.Dolia, "Comparative Study and Performance Analysis of Optimistic and Pessimistic Approaches for Concurrency Control Suitable for Temporal Database Environment", National Conference on Emerging Trends in Information & Communication Technology (NCETICT), 2013.

[7] Jaypalsinh A. Gohil, Dr. Prashant M. Dolia, "Graphical Representation of Pessimistic Locking and Concurrency Control for Temporal Database using Oracle 12c Enterprise Manager", International Journal of Innovations in Engineering and Technology (IJIET), Volume 6, Issue 4, April 2016.

[8] Jaypalsinh A. Gohil, Dr. Prashant M. Dolia, "Graphical Representation of Optimistic Locking and Concurrency Control for Temporal Database using Oracle 12c Enterprise Manager", International Journal of Computer Applications (IJCA), Volume 145, August 2016.

[9] Kung H. T. and Robinson J. T., "On Optimistic Methods for Concurrency Control", ACM Trans. on Database Systems, V. 6. No. 2, 1981.

[10] Richard T. Snodgrass and Ilsoo Ahn, "Temporal Databases", IEEE Computer 19(9), pp. 35–42, September, 1986.

[11] Oracle DBA "Tips and Techniques Gavin Soorma Oracle 12c New Feature - Temporal Validity", http://gavinsoorma.com/2013/08/oracle-12c-new-feature-emporal-validity/.

[12] H.T. Kung and John T. Robinson, "On Optimistic Methods for Concurrency Control", ACM Transactions on Database Systems, Vol. 6, No. 2, June 1981, Pages 213-226.

[13] Alexander Thomasian, "Concurrency Control : Methods, Performance, and Analysis:, ACM Computing Surveys, Vol. 30, No. 1, March 1998.

[14] Partha Dasgupta, Zvi Kedem, "The Five Color Concurrency Control Protocol: Non-Two-Phase Locking in General Databases", ACM Transactions on Database Systems, Vol. 15, No.2, June 1990, Pages 281-307.

[15] Henry F. Korth, Abraham Silberchatz, S. Sudarshan, "Concurrency Control: Database system Concepts", (Forth Edition) Page : 591 -617.

[16] Bharat Bhargava, "Concurrency Control in Database Systems", IEEE Transactions on Knowledge and Data Engineering, Vol. 11, NO. 1, January/ February 1999.

[17] Alexander Thomasian, "Concurrency Control: Methods, Performance, and Analysis", ACM Computing Surveys, Vol. 30, No. 1, March 1998.
[18] Philip A. Bernstein, Nathan Goodman, "Multiversion Concurrency Control-Theory and Algorithms", ACM Transactions on Database Systems, Vol. 8, No. 4, December 1983, Pages 465-483.
[19] Christos H. Papadimitrious, Prais C. Kanellakis, "On Concurrency Control by Multiple Versions", ACM Transactions on Database Systems, Vol. 9, No. 1,March 1984, Pages 89-99.
[20] Achraf Makani, Rafik Bouaziz, "Concurrency Control for Temporal Databases", The International Journal of Database Management Systems (IJDMS), Vol.2, No.1, February 2010.
[21] J. A. Gohil, P.M.Dolia, "Testing Temporal Data Validity in Oracle 12c using Valid Time Temporal Dimension and Queries", Journal of Engineering Computers & Applied Sciences(JECAS), Volume 4, No.4, April 2015.
[22] Jaypalsinh A. Gohil, Dr. Prashant M. Dolia "Checking and Verifying Temporal Data Validity using Valid Time Temporal Dimension and Queries In Oracle 12c", International Journal of Database Management Systems (IJDMS), Vol.7, No.4, August 2015.

AUTHORS

Jaypalsinh A. Gohil is a research scholar at Department of Computer Science & Applications, MK Bhavnagar University, Bhavnagar, Gujarat-India. He is having more than 10 years of teaching experience at U.G. & P.G. level. He has already published in total 11 research papers in various international and national journals. He has also contributed in the area of computer science and applications by publishing 02 books.

Dr. Prashant M. Dolia is working as an Associate Professor at Department of Computer Science & Applications, MK Bhavnagar University, Bhavnagar, Gujarat-India. He is possessing more than eighteen years of teaching experience at PG level. He has published 28 research papers in various international journals and 01 research paper in national journal. He has also published 09 books in subject of Computer Science & Applications.