An Optimistic Approach in Distributed Database Concurrency Control

Obaidah A. Rawashdeh
MSc. Student, Computer Science Dept.
Faculty of Information Technology
Applied Science University
Amman, Jordan
placed@live.com

Hiba A. Muhareb
MSc. Student, Computing Science Dept.
Faculty of Information Technology
Princess Sumaya University of Technology.
Amman, Jordan
hiba.m99@gmail.com

Nedhal A. Al-Sayid
Associate Professor, Computer Science Dept., Faculty of Information Technology
Applied Science University
Amman, Jordan
Nedhal_alsaiyd@asu.edu.jo

Abstract— **In this paper, we present an optimistic concurrency control solution. The proposed solution represents an excellent blossom in the concurrency control field. It deals with the concurrency control anomalies, and, simultaneously, assures the reliability of the data before read-write transactions and after successfully committed. It can be used within the distributed database to track data logs and roll back processes to overcome distributed database anomalies. The method is based on commit timestamps for validation and an integer flag that is incremented each time a successful update on the record is committed.**

We show that this method overcomes inefficient taken methods (i.e. unnecessary restarts) in order to improve the performance and to increase productivity. Moreover, the method includes information about the time-stamping of transactions and flagging successfully updated data in the main-memory as it is proposed.

Finally, the implemented work finds a need for an adaptive and an integrated concurrency control method in distributed database systems. Thus, a new optimistic concurrency control method is presented in this paper where it is expected to produce correct and reliable results.

Keywords- Concurrency Control, Optimistic, Distributed database;

I. INTRODUCTION

The Concurrency Control Problem is the activity of coordinating concurrent accesses to a Database in a multi user Management System (DBMS). Concurrency control authorizes users to access a database in a multiple programmed fashion while preserving the illusion that each user is executing alone on a dedicated system [1, 3]. The main technical difficulty in achieving this goal is to prevent database updates performed by one user from interfering with database retrievals and updates performed by another user. The concurrency control problem is worsened in a distributed (DDBMS) because (1) users may access data stored in many different computers in a distributed system, and (2) a concurrency control mechanism at one computer cannot instantly be acknowledged. As Distributed database concurrency control is one of the most complex and challenging areas in database systems [2]. It is also of great significance, where real-time software is necessary to all reliable and critical applications of concurrency control in distributed database. Real-time systems play a dominant role here, where too many applications involve real-time tasks, which often carry significant drawbacks in terms of cost and loss time.

In recent studies, a lot of research work has been dedicated to the concurrency controls methods and mechanisms as they are highly required to maintain Real-time distributed database reliability and stability as they also chain several features from real-time applications used in distributed database that assist (1) the description of data (metadata), (2) the preservation of correctness and integrity of the data, (3) efficient access to the data, and (4) the correct accomplishments of transactions despite concurrency and failures [4,5,6 and 12].

Traditionally, databases deal with persistent data. Main transactions that access data while preserving its consistency. The goal of transaction and query processing methods chosen in databases is to get a good throughput or response time in order to guarantee a better performance. On the contrary, real-time database systems can also deal with temporary data, (i.e., data timed out after a certain time). The important difference is that the goal of real-time database systems is to meet the time constraints of the transactions. It is also worth to mention here that real-time does not necessarily mean fast [9, 10 and 13]. Real-time means the need to manage categorical time constraints in a predictable manner, that is, to use time-conscious methods to deal with deadlines or periodicity constraints associated with tasks and transactions [11, 18].

The work magnifies the need for sophisticated concurrency control methods in real-time distributed database systems [36]. Hence, a new optimistic concurrency control approach is presented where it is based on reworking to the current workload.

Every database transaction conforms the *ACID* rules*; Atomicity, Consistency, Isolation, and Durability rules.* Depending on the transaction types, the level of parallelism, and other factors, different concurrency control mechanisms are implemented. a) *Pessimistic concurrency control*: if more

than one transaction intended to update the same data at the same time, then lock the data and prevents the overlapping update operations of the transactions to access the same data but permit the read operations to access the locked data. b) Optimistic *concurrency control*: Delay checking whether the transaction meets the ACID rules to the commit state, to prevent violation.

This paper has been organized as follows. Chapter 2 presents a literature description of concurrency control studies over the years. Meanwhile, Chapter 3 presents concurrency control in real-time databases. And Chapter 4 presents a new optimistic concurrency control mechanism to lessen overhead and improve efficiency. Chapter 5 draws the conclusions of this study.

II. LITRATURE

In this section, we present some well-known pessimistic concurrency control methods. Most of these methods are based on Two Phase Locking (2PL). 2PL High Priority In the 2PL-HP (2PL High Priority) concurrency control method [14, 15, 22, 24] conflicts are resolved approving of the higher priority transactions. If the priority of the lock requester is higher than the priority of the lock holder, the lock holder is aborted, rolled back and restarted. The lock is granted to this requester and the requester can continue its execution. If the priority of the lock requester is lower than that of the lock holder, the requesting transaction blocks to wait for the lock holder to finish and release its locks [18, 20, and 29]. High Priority concurrency control may lead to the cascading blocking problem, a deadlock situation, and priority inversion. 2PL Wait Promote In 2PL-WP (2PL Wait Promote) [14,17] the analysis of concurrency control method is developed from [28]. The mechanism presented uses shared and exclusive locks. Shared locks permit multiple concurrent readers. A new definition is made - the priority of a data object, which is defined to be the highest priority of all the transactions holding a lock on the data object. If the data object is not locked, its priority is undefined [23, 31, 42].

A transaction can join in the read group of an object if and only if its priority is higher than the maximum priority of all transactions in the write group of an object. Thus, conflicts arise from incompatibility of locking modes as usual. Particular attention is given to conflicts that lead to priority inversions. A priority inversion occurs when a transaction of high priority requests and blocks for an object which has lesser priority. This means that all the lock holders have lesser priority than the requesting transaction. This same method is also called 2PL-PI (2PL Priority Inheritance) [34]. 2PL Conditional Priority Inheritance sometimes high priority may be too strict policy. If the lock holding transaction T can finish in the time that the lock requesting transaction T can afford to wait, that is within the slack time of T, and let T proceed to execution and T wait for the completion of the transaction. This policy is called 2PL-CR (2PL Conditional Restart or 2PL-CPI (2PL Conditional Priority Inheritance) [33, 34, 44]. Priority Ceiling Protocol [44] the focus is to minimize the duration of blocking to at most one lower priority task and prevent the formation of deadlocks. A real-time database can often be decomposed into sets of database objects that can be modeled as atomic data sets. The Consistency constraints can be checked and validated locally. The notion of atomic data sets is especially useful for tracking multiple targets.

A simple locking method for elementary transactions is the two-phase locking method; a transaction cannot release any lock on any atomic data set unless it has obtained all the locks on that atomic data set. Once it has released its locks it cannot obtain new locks on the same atomic data set, however, it can obtain new locks on different data sets. The theory of modular concurrency control permits an elementary transaction to hold locks across atomic data sets. This increases the duration of locking and decreases perceptibility. In this study transactions do not hold locks across atomic data sets.

Priority Ceiling Protocol minimizes the duration of blocking to at most one elementary lower priority task and prevents the formation of deadlocks [26,29 and 27]. The idea is that when a new higher priority transaction preempts a running transaction its priority must exceed the priorities of all preempted transactions, taking the priority inheritance protocol into consideration. If this condition cannot be met, the new transaction is suspended and the blocking transaction inherits the priority of the highest transaction it blocks.

The priority ceiling of a data object is the priority of the highest priority transaction that may lock this object [23, 31 and 40]. A new transaction can preempt a lock-holding transaction if and only if its priority is higher than the priority ceilings of all the data objects locked by the lock-holding transaction. If this condition is not satisfied, the new transaction will wait and the lock-holding transaction inherits the priority of the highest transaction that it blocks. The lock-holder continues its execution, and when it releases the locks, its original priority is resumed. All blocked transactions are awaked, and the one with the highest priority will start its execution.

The fact that the priority of the new lock-requesting transaction must be higher than the priority ceiling of all the data objects that it accesses, prevents the formation of a potential deadlock. The fact that the lock- requesting transaction is blocked only at most the execution time of one. Lower priority transaction guarantees, the formation of blocking chains is not possible [25, 37]. Read-Write Priority Ceiling the Priority Ceiling Protocol is further advanced in [32, 36 and 43], it is introduced. It contains two basic ideas. The first idea is the notion of priority inheritance. The second idea is a total priority ordering of active transactions. A transaction is said to be active if it has started but not completed its execution. Thus, a transaction can execute or wait caused by preemption in the middle of its execution. Total priority ordering requires that each active transaction execute at a higher priority level than the active lower priority transaction, taking priority inheritance and read-write semantics into consideration [36].

In this paper we highlight the priority protocol by implementing a timestamp where another value of flag is added and this value does not change unless transaction update has been committed successfully.

III. METHODOLOGY

Optimistic concurrency control is useful for long-running transactions that rarely affect the same instances, and therefore

the data-store will exhibit better performance by deferring the data-store exclusion on modified instances until commit. In this study we utilize the use of optimistic concurrency control to be applied on real-time distributed database systems.

A stamps will be added to each record and are sent to the user each time a user accesses that record, that stamp will be a counter and will hold the value of how many times a certain record is being accessed whether through read or write or read-write operation.

The stamps will be hidden from the user, the process then executes the procedure normally and performs a commit. The values of all stamps of accessed records that are received from the server and stored in the client's DBMS are then sent to the server. On the server side, a comparison takes place between the stamps received from the client after it finished execution and the ones already stored in the server's database, if values are equal, then data is concurrent, however; if the client requests to modify some of the records accessed, the stamps will be increased by one to acknowledge modifications, and a corresponding flag will be returned to the user to indicate the successful commit process of the whole transaction, however; if values do not match then that indicates values of accessed records are modified by another transaction, all the records and their new flags are returned to the client and a restart of the transaction is then needed.

The below Pseudo code shows the algorithm of the described above process, as follows:

```
While (operational)
{
   If (UserRequestsData (user))
   {
   Send data and stamps to the addressed user
   }
   If (UserCommits (Stamps))
   { //his will include the stamps
      If (Compare (data.stamps))
      { //if stamps are equal//
         If (ChangesNeeded ())
         {
            PerformChanges (data);
            IncreaseStamps ()
         }
          ReturnSuccessful ( );
      }

      Else {
            ReturnFaliure ();
            }
      Continue;
}
```

The figure below demonstrates the above method described previously using an active diagram as follows:

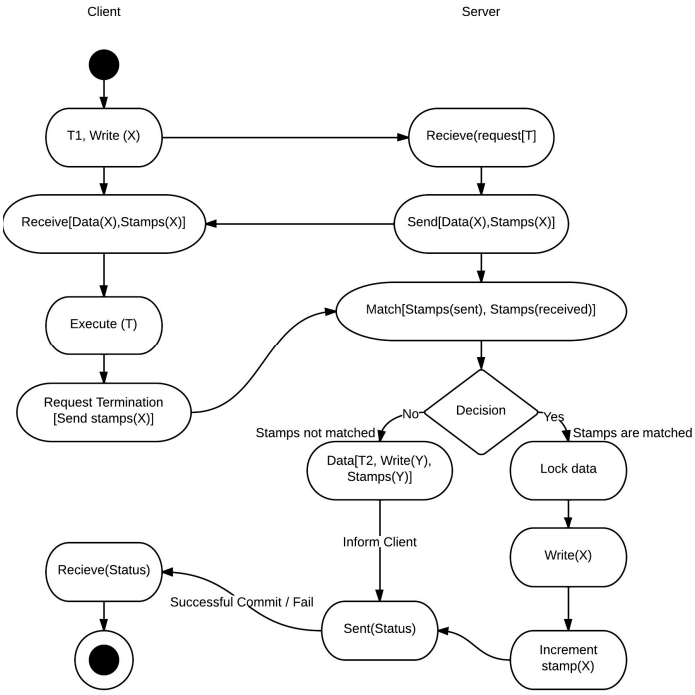

Figure 1. The sequence of actions over time and the behavior of both client and server.

Basically, a user receives data and their stamps and awaits few cycles before the commit process (until the procedure is complete), the DBMS will be able to determine if the data requested is updated meanwhile the execution of the procedure or not. Simply by, comparing the stamp flags received from the client side when he commits to the server's side stamps stored in database, in the case of match then no changes have been made to the requested data. This way implicitly the server will return success flag to the client's side and process will be carried forward. In the case of no match, a failure flag is returned to the client that indicates restart of the transaction is needed.

IV. CONCLUSION

In conclusion the novelty of the proposed method is shown in assigning an extra field of type integer to every record in the real-time DBMS, for addressing purposes where it was mentioned above as 'stamp' field, this stamp will hold an initial value of zero and will be incremented by one each time a successful update is held on a record. However, complexity issues arise where it shows that the process after each commits has to update the stamp back and forth from the client to the

server's side which creates overhead. On the other hand the proposed method decreases the restart times needed and enhances efficiency regarding the matter.

In the soon future, the complexity concerns will be brought to the discussion table in order to enhance the proposed method and carry it on to the next level.

REFERENCES

[1] R. Abbott and H. Garcia-Molina. Scheduling real-time transactions. ACM SIGMOD Record, 17(1):7181, March 1988.

[2] D. Agrawal, A. E. Abbadi, and R. Jeffers. Using delayed commitment in locking protocols for real-time databases. In Proceedings of the 1992 ACM SIGMOD International Conference on Management of Data, pages 104-113. ACM Press, 1992.

[3] I. Ahn. Database issues in telecommunications network management. ACM SIGMOD Record, 23(2):37-43, June 1994.

[4] P. A. Bernstein, V. Hadzilacos, and N. Goodman. Concurrency Control and Recovery in database Systems. Addison-Wesley, 1987.

[5] S. R. Biyabani, J. A. Stankovic, and K. Ramamritham. The integration of deadline and criticalness in hard real-time scheduling. In Proceedings of the 8th IEEE Real-Time Systems Symposium, pages 487-507. IEEE Computer Society Press, 1988.

[6] J. O. Boese and A. A. Hood. Service control point — database for 800 service. In Proceedings of GLOBECOM'8 , December 1986.

[7] T. F. Bowen, G. Gopal, G. Herman, and W. Mansfield Jr. A scale database architecture for network services. IEEE Communications Magazine, 29(1):52-59, January 1991.

[8] A. Burns and A. Wellings. Real-Time Systems and Programming Languages. Addison-Wesley, 2001.

[9] S. Chen, J. A. Stankovic, J. F. Kurose, and D. Towsley. Performance evaluation of two new disk scheduling algorithms for real-time systems. The Journal of Real-Time Systems, 3(3):307-336, September 1991.

[10] A. Datta and S. Mukherjee. Buffer management in real-time active database systems. In Real-Time Database Systems Architecture and Techniques, pages 77-96. Kluwer Academic Publishers, 2001.

[11] 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.

[12] M. H. Graham. Issues in real-time data management. The Journal of Real-Time Systems, 4:185-202, 1992.

[13] P. Graham and K. Barker. Effective optimistic concurrency control in multiversion object bases. Lecture Notes in Computer Science, 58:313-323, 1994.

[14] J. Gray and A. Reuter. Transaction Processing: Concepts and Techniques. Morgan Kaufmann, 1993.

[15] U. Halici and A. Dogac. An optimistic locking technique for concurrency control in distributed databases. IEEE Transactions on Software Engineering, 17(7):712-724, July 1991.

[16] H. Han, S. Park, and C. Park. A concurrency control protocol for read-only transactions in real-time secure database systems. In Proceedings of the 7th International Conference on Real-Time Computing Systems and Applications, pages 458-462. IEEE Computer Society Press, 2000.

[17] J. Hansson and H. Son, S. Overload management in RTDBS. In Real-Time Database Systems - Architecture and Techniques, pages 125-140. Kluwer Academic Publishers, 2001.

[18] T. Höarder. Observations on optimistic concurrency control schemes. Information Systems, 9(2):111-120, 1984.

[19] J. Haritsa, M. Carey, and M. Livny. Value-based scheduling in real-time database systems. Tech. Rep. CS-TR-91-1024, University of Winconsin, Madison, 1991.

[20] J. R. Haritsa, M. J. Carey, and M. Livny. Dynamic real-time optimistic concurrency control. In Proceedings of the 11th IEEE Real-Time Systems Symposium, pages 94-103. IEEE Computer Society Press, 1990.

[21] D. Hong, S. Chakravarthy, and T. Johnson. Locking based concurrency control for integrated real-time database systems. In Proceedings of the First International Workshop on Real-Time Databases: Issues and Applications, March 7-8, 1996.

[22] J. Huang, J. A. Stankovic, K. Ramamritham, and D. Towsley. Experimental evaluation of real-time optimistic concurrency control schemes. In Proceedings of the 17th VLDB Conference, pages 35-46, September 1991. Morgan Kaufmann.

[23] S. Hvasshovd, . Torbj rnsen, S. E. Bratsberg, and P. Holager. The ClustRa telecom database: High availability, high throughput, and real-time response. In Proceedings of the 21th VLDB Conference, pages 469-477. Morgan Kaufmann, 1995.

[24] ITU. Introduction to Intelligent Network Capability Set 1. Recommendation É.1211. ITU, International Telecommunications Union, Geneva, Switzerland, 1993.

[25] M. Jarke and M. Nicola. Telecommunication databases – applications and performance analysis. In Databases in telecommunications, Lecture Notes in Computer Science, vol 1819, pages 1-15, 1999.

[26] B.-S. Jeong, D. Kim, and S. Lee. Optimistic secure real-time concurrency control using multiple data version. In Lecture Notes in Computer Science, volume 1985, 2001.

[27] B. Kao and R. Cheng. Disk scheduling. In Real-Time Database Systems -Architecture and Techniques, pages 97-108. Kluwer Academic Publishers, 2001.

[28] R. Kerboul, J.-M. Pageot, and V. Robin. Database requirements for intelligent network: How to customize mechanisms to implement policies. In Proceedings of the 4th TINA Workshop, volume 2, pages 35-46, September 1993.

[29] N. Kline. An update of the temporal database bibliography. ACM Sigmod Record, 22(4):66-80, 1993.

[30] H. T. Kung and J. T. Robinson. On optimistic methods for concurrency control. ACM Transactions on Database Systems, 6(2):213-226,June 1981.

[31] T.-W. Kuo and K.-Y. Lam. Conservative and optimistic protocols. In Real-Time Database Systems - Architecture and Techniques, pages 29-44. Kluwer Academic Publishers, 2001.

[32] K.-Y. Lam and S.-L. Hung. Concurrency control for time-constrained transactions in distributed databases systems. The Computer Journal, 38(9):704-716, 1995.

[33] J. Lee. Concurrency Control Algorithms for Real-Time Database Systems. PhD thesis, Faculty of the School of Engineering and Applied Science, University of Virginia, January 1994.

[34] V. C. S. Lee and K-W. Lam. Optimistic concurrency control in broad-cast environments: Looking forward at the server and backward at the clients. In H. V. Leong, W-C. Lee, B. Li, and L. Yin, editors, First International Conference on Mobile Data Access, Lecture Notes in Computer Science, vol 1748, pages 97-106. Springer Verlag, 1999.

[35] K.-J. Lin and M.-J. Lin. Enhancing vailability in distributed real-time databases. ACM SIGMOD Record, 17(1):34-43, March 1988.

[36] J. Lindström. Extensions to optimistic concurrency control with time intervals. In Proceedings of 7th International Conference on Real-Time Computing Systems and Applications, ages 108-115. IEEE Computer Society Press, 2000.

[37] C.L.Liu and J.W. Layland. Scheduling algorithms for multiprogramming in a hard real-time environment. Journal of the ACM, 20(1):46-61, January 1973.

[38] K. Marzullo. Concurrency control for transactions with priorities. Tech. Report TR 89-996, apartment of Computer Science, CornellUniversity, Ithaca, NY, May 1989.

[39] C. Papadimitriou. The Theory of Database Concurrency Control. Computer Science Press, 1986.

[40] Dick Pountain. The Chorus microkernel. Byte, pages 131-138, January 1994.

[41] B. Purimetla, R. M. Sivasankaran, K. Ramamritham, and J. A. Stankovic. Real-time databases: Issues and applications. In S. H. Son,

editor, Advances in Real-time Systems, pages 487-507. Prentice Hall, 1996.

[42] R. Ramamritham, R. M. Sivasankaran, J. A. Stankovic, D. F. Towsley, and M. Xiong. Integrating temporal, real-time, and active databases. SIGMOD Record, 25(1):8-12, 1996.

[43] L. Sha, R. Rajkumar, and J. P. Lehoczky. Concurrency control for distributed real-time databases. ACM SIGMOD Record, 17(1):82-98, March 1988.

[44] L. Sha, R. Rajkumar, S. H. Son, and C.-H. Chang. A real-time locking protocol. IEEE Transactions on Computers, 40(7):793-800, January 1991.