ISSN 2348-1196 (print)
International Journal of Computer Science and Information Technology Research ISSN 2348-120X (online)
Vol. 4, Issue 2, pp: (262-271), Month: April - June 2016, Available at: www.researchpublish.com

Three Read Priority Locking for Concurrency Control in Distributed Databases

Christos Papanastasiou

Technological Educational Institution Stereas Elladas, Department of Electrical Engineering 35100 Lamia, GREECE

Abstract: **In Distributed Database Systems concurrency control is very important for transaction execution time and communication costs.**

In this paper I suggest a new method of concurrency control giving high priority in 3 read transactions which are waiting in the queue, every time an exclusive lock (after the execution of a write transaction) is released. This way the execution time is reduced significantly and at the same time read transactions waiting in the queue are executed and released from the queue without burdening the runtime of waiting write transactions. In the worst case, when many write transactions are waiting one after the other to be granted an exclusive lock and then served, granting a shared lock to 3 read transactions right after a write transaction has released it's lock, the waiting time of these transactions is reduced significantly. Besides all these, we overcome the disadvantages of the traditional locking mechanism for Distributed Database Systems (DDBS), like the low transaction concurrency and the high communication costs.

Keywords: **Locking, index-based, priority, global node.**

1. INTRODUCTION

Often, a collection of several operations on a database appears to be a single unit from the point of fiew of the database user [8]. For example, a transfer of funds from a checking account to a savings account is a single operation from the customer's standpoint; within the database system, however, it consists of several operations. Collections of operations that form a single logical unit of work are called transactions. A transaction is usually initiated by a user program written in a high-level data manipulation language or programming language (for example C++, SQl, or Java), where it is delimited by statements or function calls of the form **begin transaction** and **end transaction**.

To insure integrity of the data, we require that the database system maintain the following properties of the transactions:

- **Atomicity**. Either all operations of the transaction are reflected properly in the database, or none are.

- **Consistency**. Execution of a transaction in isolation (with no other transaction executing concurrently) preserves the consistency of the database.

- **Isolation**. Even though multiple transactions may execute concurrently, the system guarantees that, for every pair of transactions Ti and Tj, it appears to Ti that either Tj finished execution before Ti started or Tj started execution after Ti finished. Thus, each transaction is unaware of other transactions executing concurrently in the system.

- **Durability**. After a transaction completes successfully, the changes it has made to the database persist, even if there are system failures.

These properties are often called the ACID properties.

A DBMS must often serve many applications, and respond to requests from many users. The application load of a DBMS can be measured using the number of transactions per second (tps) managed by the DBMS to satisfy the needs of applications [9]. Typical systems, for example banks or financial information systems, must respond to loads of tens to hundreds of tps. The booking systems of large airlines or credit card management must reach thousands of tps. For this reason, it is essential that the transactions of a DBMS be carried out in sequence. Only the concurrency of transactions allows for the efficient operation of a DBMS, maximizing the number of transactions carried out per second and minimizing their response times.

In a serial execution of transactions, called serial schedule, each transaction is run to completion before any operation of other transactions is executed.

Several concurrency control algorithms have been proposed in the literature [11],[12]. Locking is one of the common techniques widely used to achieve concurrency control. In this paper, I present a new locking method called "assigning read transaction a higher priority" which is suggested for concurrency control in distributed DBMS.

2. TRANSACTION LOCK MANAGEMENT

Users are responsible for ensuring transaction consistency. That is, the user who submits a transaction must ensure that, when run to completion by itself against a "consistent" database instance, the transaction will leave the database in a "consistent" state. A transaction is seen by the DBMS as a series, or list, of actions. These actions that can be executed by a transaction include **reads** and **writes** of database objects.

In addition to reading and writing, each transaction must specify as its final action either a commit (i.e., complete successfully) or abort (i.e., terminate and undo all the actions carried out thus far). [10]

A schedule is a list of actions (reading, writing, aborting, or committing) from a set of transactions and the order in which two actions of a transaction T appear in a schedule must be the same as the order in which they appear in T.

A DBMS must be able to ensure that only serializable, recoverable schedules are allowed and that no actions of committed transactions are lost while undoing aborted transactions. A DBMS typically uses a *locking protocol* to achieve this. A **lock** is a small bookkeeping object associated with a database object. A **locking protocol** is a set of rules to be followed by each transaction (and enforced by the DBMS) to ensure that, even though actions of several transactions might be interleaved, the net effect is identical to executing all transactions in some serial order. Different locking protocols use different types of locks, such as shared locks or exclusive locks.

Generally, there is one lock for each data object in a database. Locks are used to synchronize the access by concurrent transactions to the database items.

Concurrency occurs when two transactions access the same data object, say O, at the same time. Concurrency. control deals with preventing concurrently running processes from improperly inserting, deleting or updating the same data object, i.e. it ensures that correct results for concurrent operations are generated, while getting those results as quickly as possible.

A lock is a variable that is associated with a data object which describes the status of the object with respect to possible operations that can be applied to it. Whenever a transaction requests for a lock, and at that time if the lock is already locked by any other transaction then the requesting transaction waits and is queued up. Locking technique can result in deadlocks.

Deadlock: If we consider that a transaction T1 sets an exclusive lock on object A, T2 sets an exclusive lock on B, T1 requests an exclusive lock on B and is queued, and T2 requests an exclusive lock on A and is queued, it means, that, T1 is waiting for T2 to release its lock and T2 is waiting for T1 to release its lock. Such a cycle of transactions waiting for locks to be released is called a deadlock. Clearly, these two transactions will make no further progress. Worse, they hold locks that may be required by other transactions. The DBMS must either prevent or detect (and resolve) such deadlock situations. The most common approach is to detect and resolve deadlocks. A simple way to identify deadlocks is to use a timeout mechanism. If a transaction has been waiting too long for a lock, we can assume that it is in a deadlock situation and abort it

3. SIMILAR WORK

Concurrency control technique that is used in the development of our database system uses locking technology. To ensure serializability scheduling, locking protocol must be used, which means that if a transaction requests for a lock on an object and the lock has already been given to another transaction, the requesting transaction must wait.

A globalized locking concurrency algorithm based on wait mechanism, was developed by Bharat Bhargava[13]. His suggestion is that when two transactions conflict, then one transaction must wait until the other transaction has released the objects that are common to both transactions. To implement this, the system must provide locks on the database objects. There are two locks that can be implemented:

Read lock: The transaction locks the object in a shared mode. Any other transaction waiting to read the same object can also obtain a read lock.

Write lock: The transaction locks the object in an exclusive mode. If one transaction wants to write on an object, no other transaction can be granted a read lock or a write lock.

3.1. Globalized Locking Concurrency Control Algorithm:

As shown in figure 1 when a transaction Ti arrives at node A, the following steps are performed: [22]

- Node A requests from global node the locks for all objects that are referenced by transaction Ti.

- Global node checks all requested locks. If some object is already locked by another transaction, then the request is queued. There is a queue for each object and the request waits for one queue at a time.

- When the transaction gets all its locks, it is executed at global node. The values that correspond to a read job are read from the database, and the values that correspond to a write job are written in the database at the global node.

- The values of the write job are transmitted by the global node to all other nodes (if the database is fully replicated).

- Each node receives the new values for the write job and updates the database; then an acknowledgment is sent back to global node.

- When the global node receives all acknowledgments from all nodes in the system, it knows that transaction Ti has been completed at all nodes. The global node releases the locks and starts processing the next transaction.

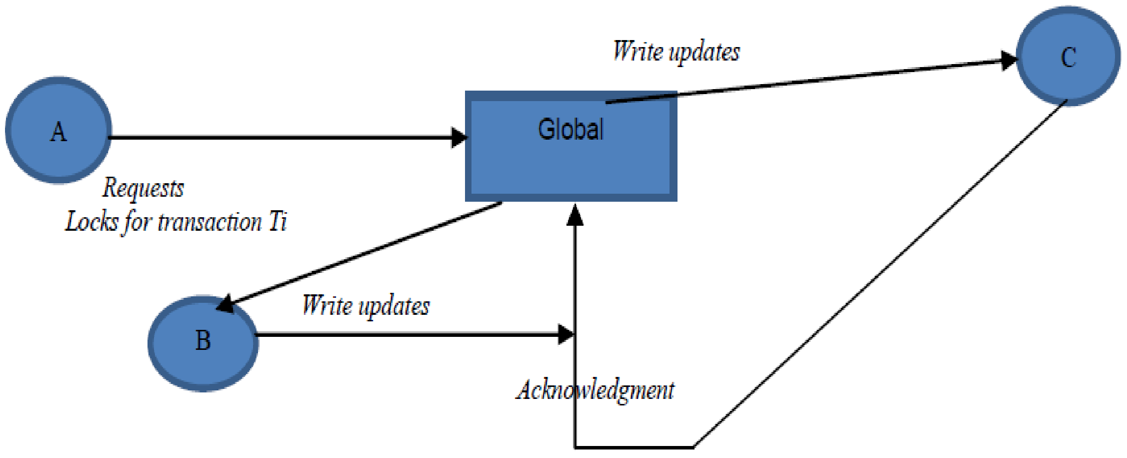

Fig 1: Centralized Locking Concurrency Control Protocol

There are some common variations of the globalized locking algorithm as follows:

3.2. Locking at Global Node, Execution at all Nodes:

Instead of executing the transaction at the global node, we can only assign locks at global node; send the transaction back to node *A* and the transaction *Ti* is read, and the values from the write job are obtained at node *A*. Node *A* sends the values from write job and obtains acknowledgments from all other nodes. It then knows that transaction *Ti* has been completed. Node *A* sends a message to unlock objects referenced by *Ti*. The global node after receiving this message, releases all it's locks and starts assigning locks to waiting transactions.

3.3. Avoid Acknowledgments, Assign Sequence Numbers:

In the globalized control algorithm, the acknowledgments are needed by global node (or node A in the above extension) to find out if the values of the write job have been written in the database at every node. For global node it is not necessary to wait for this to happen. [19] It can guarantee that the values from the write job will be written at every node in the same order as they were sent at global node. To achieve this, the global node can assign an increasing sequence number to each transaction. The sequence number is appended to the write job of the transaction and is used to keep in order the update of the new values into the database at each node.

This way, global node does not have to wait for any acknowledgments, but the equivalent effect is achieved. This way the globalized control algorithm is more efficient.

The basic disadvantage of this algorithm is that it takes long time for transactions to execute and this increases the waiting time for the transactions in the queue.

4. SUGGESTED LOCKING PROTOCOL

In the suggested protocol, I assume that the database is fully replicated. The system consists of N number of nodes where transactions can appear. Each node sends its read/write request to the global node. Global node assigns the requested lock to the transaction which has the higher priority. A waiting queue is maintained where waiting transactions for a lock are placed.

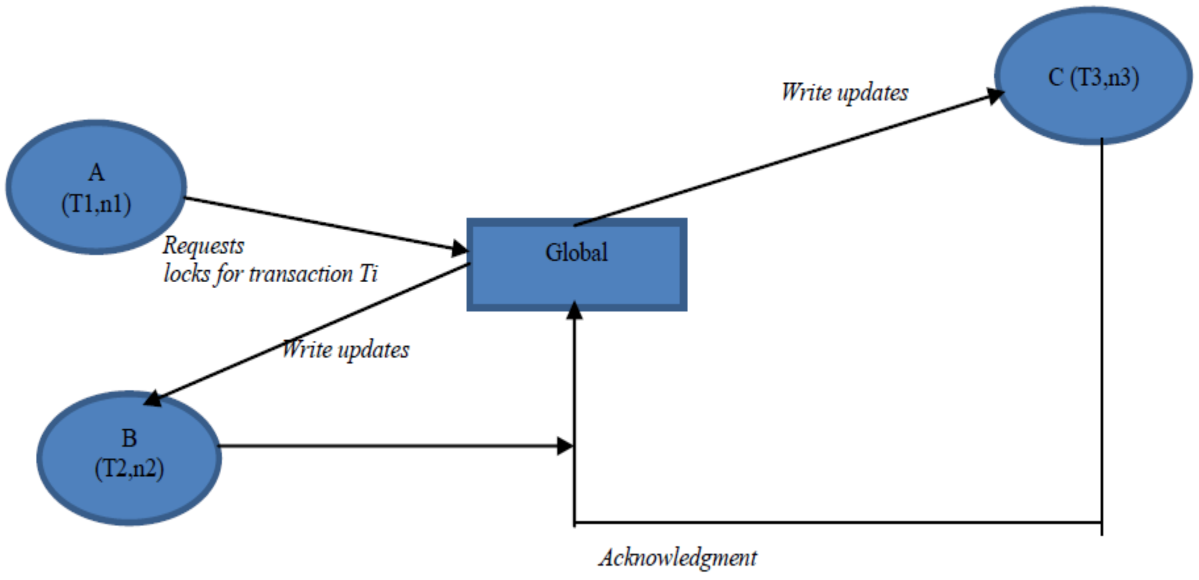

Fig 2: Distributed Locking Concurrency Control Protocol

The first transaction from the queue is granted it's requested lock.

When the lock manager has released a write's lock (exclusive lock), then it will check the next transactions in the queue and it will grant a shared lock to the next three read transactions. This happens every time a new write transaction from the queue is granted a lock and released it.

When a transaction T1 arrives at node A with an index number n1 (according to the time when it was submitted) and T2 arrives at node B with an index number n2, the following steps are performed (shown in Fig 2):

- Node A and node B request from the global node the locks for all objects referenced by transaction T1 and T2.

- Global node checks all requested locks. If some object is already locked by another transaction, then the request is queued. Lock manager will serve first requests of transaction who has the smaller (higher priority) index number.

- When transactions get all their shared locks, they are executed at the global node Global node then assigns the requested lock to the transaction waiting in the queue.

- The values of write job (for each exclusive lock) are transmitted by the global node to all other nodes.

- Each node when receives the new values from a write job, updates the database and then acknowledges global node .

- When global node receives acknowledgments from all other nodes in the system, it knows that transaction T1 has been completed at all nodes. Then, global node releases the locks and starts processing the next transaction.

ISSN 2348-1196 (print)
International Journal of Computer Science and Information Technology Research ISSN 2348-120X (online)
Vol. 4, Issue 2, pp: (262-271), Month: April - June 2016, Available at: www.researchpublish.com

5. EXAMPLES

I will explain the suggested algorithm using some examples. I consider a database which consists of large number of nodes and I will assume that multiple transaction requests happen for read/write lock for the same data object from the global node.

5.1 Example 1:

In this example I will use 6 nodes, 1,2, 3, 4, 5 and 6. Each transaction is represented by a T followed by its index number n1 [T, n1]. The global node is responsible to assign locks to the requesting transactions on the basis of priority according to their sequencing index number. Each node sends its read/write request to the global node. Global node assigns the requested lock to the transaction having the higher priority. A waiting queue is maintained in which the transactions are waiting for a lock to be placed.

We will calculate the total execution time of the transactions performing read/write operation based on index-based concurrency control and then we will calculate the total execution time based three read (3R) priority. Then we will compare the results between the two cases.

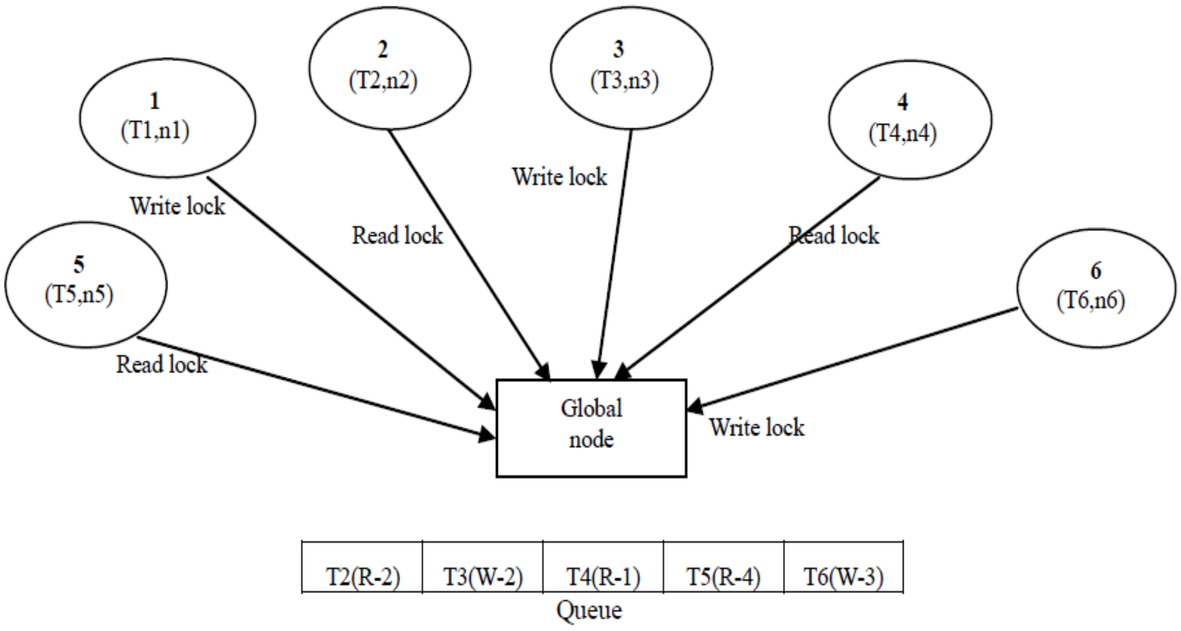

Fig 3: Distributed Locking Concurrency Control Protocol with 6 nodes

Let's suppose that Transactions T2,T4,T5 request for read_lock, and T1,T3,T6 request for write_lock as we can see in Fig. 3

If we suppose that the required time for each transaction to execute (in seconds) is: T1= 2, T2= 1, T3= 3, T4= 2, T5= 3 and T6= 5 sec.

All transactions request for the corresponding Transaction a lock on the same data object O.

Transaction T1 will be granted a write_lock because T1 is the oldest transaction with index n1. When the lock of transaction T1 is released, Lock manager in Global node checks the next request. Next transaction in the queue T2 is a read transaction and it will be granted with a shared lock. Lock manager will check the next transaction in the queue T3, which is a write transaction and this means that it will wait till lock request of T2 is released.

The execution time for T1 will be 2 sec.

Execution time for T2 will be 2+1=3 sec. When lock manager releases the lock for transaction T2, it will grant the requested exclusive lock for T3 for write. Execution time for T3 is 2+1+3=6 sec.

When the exclusive lock of transaction T3 will be released then the lock manager will grant a shared lock for read operation of T4. Execution time for transaction T4 is :2+1+3+2=8 sec.

Since the lock that was granted was a shared one, lock manager will look into the queue for the next transaction and since this transaction T5 is also a read transaction it will grant the shared lock to this transaction too and this means that the execution time for transaction T5 will be the same as it was for T4, that is 8 sec. Lock manager will check for the next transaction in the queue T6 which is a write transaction and this means that it will wait till the shared lock for transaction T5 will be released. When the lock of transaction T5 will be released, then the lock manager will grant the exclusive lock for transaction T6. The required execution time for transaction T6 is : 8+5=13 sec.

We will calculate now the execution time using the suggested 3R (3 Read priority) protocol.

The execution time of transaction T1 is : 2sec.

When the exclusive lock of transaction T1 will be released, the lock manager will check for the next 3 read transactions in the queue (if they exist) and it will grant a shared lock for the next 3 Read transactions. This means that it will grant the shared lock to transactions T2, T4 and T5, which means that the execution time for these transactions will be the same and it will be T2 = T4 = T5 = 2 + 1 = 3 sec.

When the shared lock is released, then an exclusive lock will be granted to T3. The execution time for transaction T3 will be : 2 + 1 + 3 = 6 sec.

When this exclusive lock for transaction T3 is released, then the lock manager will check in the queue to find the next 3 read transactions. Since there are not any more read transactions in our example, it will proceed to the next transaction, which is T6, a write transaction and it will grant the exclusive lock to this. The execution time for transaction T6 is : 6 + 5 = 11 sec.

We see that the total execution time using my suggested 3R (3 read priority protocol) is 11 sec, and is less than the total execution time when we applied the index-based concurrency control, which was 13 sec.

5.2 Example 2:

In this example I will use 10 nodes, 1,2, 3, 4, 5, 6, 7, 8, 9 and 10. Each transaction is represented as in the previous example by a T followed by its index number n1 [T, n1]. The global node is responsible to assign locks to the requesting transactions on the basis of priority according to their sequencing index number. Each node sends its read/write request to the global node. Global node assigns the requested lock to the transaction having the higher priority. A waiting queue is maintained in which the transactions are waiting for a lock to be placed.

We can calculate the total execution time of the transactions performing read/write operation based on index-based concurrency control. Then we will calculate the total execution time based on my suggested 3R priority protocol. Then we will compare the results between the two cases.

Let's suppose that Transactions T1,T3,T4,T6,T8,T9 request for read_lock, and T2,T5,T7,T10 request for write_lock as we can see in Fig. 4

If we suppose that the required time for each transaction to execute (in seconds) is: T1= 2, T2= 3, T3= 1, T4= 2, T5= 4, T6= 3, T7= 5, T8= 2, T9= 2 and T10=3.

All transactions request for the corresponding transaction a lock on the same data object O. Transaction T1 will be granted a read_lock because T1 is the oldest transaction with index n1. Lock manager will grant a shared lock to transaction T1. Lock manager will check the next transaction in the queue T2, which is a write transaction and this means that it will wait till lock request of T1 is released.The execution time for T1 is : 2 sec. When the lock of transaction T1 is released, Lock manager will grant the lock request for the next transaction which is T2 and is a write transaction. An exclusive lock will be granted for transaction T2. The execution time for T2 is : 2 +3 = 5 sec. When the exclusive Lock of transaction T2 is released, Lock manager in Global node checks the next request. Next transaction in the queue T3 is a read transaction and it will be granted with a shared lock. Lock manager will check the next transaction in the queue T4, which is a read transaction and it will grant a shared lock to this transaction too. Lock manager will look for next transaction in the queue. Next transaction is a read transaction and lock manager will wait till the shared lock that was granted to T3 and T4 be released. Execution time for transactions T3 and T4 will be : T3 = T4 = 5 + 1 = 6 sec. Lock manager will grant an exclusive lock now to write transaction T5. The execution time for transactionT5 is : 6 + 4 = 10 sec. When the exclusive lock of T5 is released, lock manager will check for the next transaction in the queue. Next transaction T6 is a read transaction and it will be granted a shared lock. Lock manager will check the next transaction, transaction T7, which is a write transaction and lock manager will wait till the shared lock for transaction T6 is released. The execution time for transaction T6 is : 10 + 3 = 13 sec. Lock manager will grant an exclusive lock now to transaction T7. The execution time for T7 will be : 13 + 5 = 18 sec. When the exclusive lock of transaction T7 is released, the lock manager will grant a shared lock to read transaction T8. Then it will check for the next transaction in the queue. Since the next transaction, T9 is also a read transaction, it will grant a shared lock for this transaction too. The execution time for T8 and T9 is : T8 = T9 = 18 + 2 = 20 sec. When the shared lock for transactions T8 and T9 is released, then the lock manager will grant an exclusive lock to the write transaction T10.

The execution time for transaction T10 is : 20 + 3 = 23 sec.

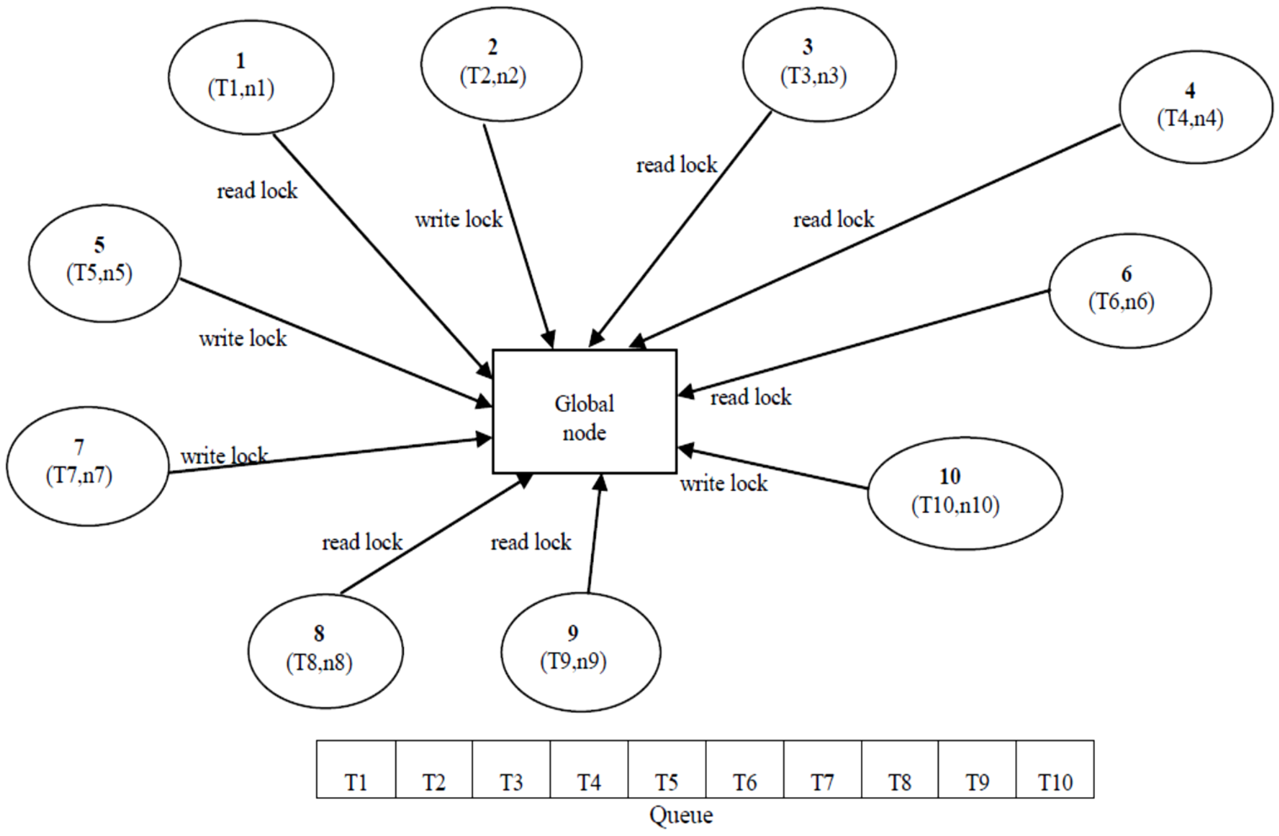

Fig 4: Distributed Locking Concurrency Control Protocol with 10 nodes

We will calculate now the execution time using my suggested 3R (3 Read priority) protocol.

Transaction T1 is a read transaction and it will be granted with a shared lock. The execution time of transaction T1 is : 2sec.

When shared lock for transaction T1 is released, lock manager will grant an exclusive lock to the next transaction T2 which is a write transaction.The execution time for transaction T2 is : 2 + 3 = 5 sec.

When the exclusive lock of transaction T2 is released, the lock manager will check for the next 3 read transactions in the queue (if they exist) and it will grant a shared lock for the next 3 Read transactions. This means that it will grant the shared lock to transactions T3, T4 and T6, which means that the execution time for these transactions will be the same and it will be T3 = T4 = T6 = 5 + 1 = 6 sec.

When the shared lock is released, then an exclusive lock will be granted to T5. The execution time for transaction T5 will be : 6 + 4 = 10 sec.

When this exclusive lock for transaction T5 is released, the lock manager will check in the queue to find the next 3 read transactions. There are only two of them, T8 and T9 and it will grant a shared lock to both of them. Execution time for T8 and T9 is : T8 = T9 = 10 +2 =12 sec..

When the shared lock that was granted to T8 and T9 is released, then the next transaction from the queue, T7 will be granted with an exclusive lock. Execution time for T7 is : 12 + 5 = 17.

When the exclusive lock for transaction T7 is released, lock manager will look for the next read transactions in the queue. Since there are not any more read transactions in our example, it will proceed to the next transaction, which is T10, a write transaction and it will grant the exclusive lock to this. The execution time for transaction T10 is : 17 + 3 = 20 sec.

We see that the total execution time using my suggested 3R (3 read priority protocol) is 20 sec, and is less than the total execution time when we applied the index-based concurrency control, which was 23 sec.

Our conclusion is that when we apply the suggested 3R protocol, the total execution time is less that the total execution time that was needed when we applied the index- based concurrency control. We also observe that this difference is greater when the number of transactions waiting in the queue is increasing. This difference will increase when there is a large number of continuous write transaction requests waiting before read transactions in the queue.

We summarize the above results in the following two tables.

Table 1: Execution time applying the index- based concurrency control of example 1.

Transactions	Execution time
T1	2 sec
T2	3 sec
T3	6 sec
T4, T5	8 sec
T6	13 sec

Table 2: Execution time applying the suggested 3R based priority of example 1.

Transactions	Execution time
T1	2 sec
T2, T4, T5	3 sec
T3	6 sec
T6	11 sec

Table 3: Execution time applying the index-based concurrency control of example2.

Transactions	Execution time
T1	2 sec
T2	5 sec
T3, T4	6 sec
T5	10 sec
T6	13 sec
T7	18 sec
T8, T9	20
T10	23

Table 4: Execution time applying the suggested 3R based priority of example2.

Transactions	Execution time
T1	2 sec
T2	5 sec
T3, T4, T6	6 sec
T5	10 sec
T8, T9	12 sec
T7	17 sec
T10	20 sec

5.3 Performance:

In Table 5, we can see the performance comparison between the Index-based concurrency control technique and the 3R priority based technique. It is obvious from this table for both examples that the average execution time of 3R priority technique is less than the index-based concurrency control technique.

Table 5. Performance table

Technique used	Average execution time of example 1	Average execution time of example 2
Index-based concurrency control	6.4 sec	12,1 sec
3R	5.5 sec	10.2 sec

6. CONCLUSIONS

The suggested 3R locking protocol is an efficient technique for concurrency control in the distributed databases. Enabling the lock manager to serve 3 read transactions waiting in the queue reduces the total execution time as well as the average execution time of all transactions. At the same time, it frees read transactions waiting in the queue without burdening the waiting time for write transactions. The execution time is reduced much more when a large number of continuous write transaction requests are waiting before read transactions in the queue.

REFERENCES

[1] Buretta, Marie. (1997) Data Replication, New York: John Wiley & Sons

[2] Burleson, Donald K. (1994) Managing Distributed Databases. New York: John Wiley & Sons

[3] M. T. Φzsu, P. Valduriez, Principles of Distributed Database Systems, Prentice-Hall, 1991.

[4] Oracle8 Server Documentation, Oracle Corporation, 1997.

[5] D. Bell and J. Grimson. Distributed Database Systems, Reading, MA: Addison-Wesley, 1993.

[6] M.T. ¨ Ozsu and P. Valduriez. Principles of Distributed Database Systems, Englewood Cliffs, NJ: Prentice-Hall, 1991.

[7] Ozsu, Tamer M., and Valduriez, Patrick [1991], Principles of Distributed Database Systems, Prentice Hall.

[8] Anraham Silberschartz, Henry F. Korth (2006) Database SystemConcepts, Mc Graw Hill International edition.

[9] Paolo Atzeni, Stefano Ceri, Stefano Paraboschi amd Riccardo Torlone (1999), Database Systems Concepts, Languages and Architectures, Mc Graw Hill.

[10] Ramakrishnan, Gehrke Database Management Systems, Mc Graw Hill International edition (2003)

[11] Bhargava Bharat: Concurrency Control in Database systems: IEEE Transactions on knowledge and data engineering, VOL.11, No1,1999 (IEEE)

[12] Ali R. Abdou, Hany M. Harb: Two phase locking concurrency control in distributed databases with N-Tier architecture; IEEE 2004.

[13] Bhargava Bharat: Concurrency Control in Database systems: IEEE Transactions on knowledge and data engineering, VOL.11, No1,1999 (IEEE)

[14] Y. Yoon, "Transaction Scheduling and Commit Processing for Real-Time Distributed Database Systems", Ph. D. Thesis, Korea Adv. Inst. of Science and Technology, May 1994.

[15] A. Bestavros, "Multi-version Speculative Concurrency Control with Delayed Commit", P m. of Zntl. Conf. on Computers and their Applications, March 1994.

[16] E. Cooper, "Analysis of Distributed Commit Protocols", Proc. of ACM Sigmod Conf., June 1982.

[17] R. Gupta, J. Haritsa, K. Ramamritham and S. Seshadri, "Commit Processing in Distributed RTDBS", TR-96-01, DSL/SERC, Indian Institute of Science.

[18] J. Haritsa, M. Carey, and M. Livny, "Data Access Scheduling in Firm Real-Time Database Systems", Real-Time Systems Journal, 4 (3), 1992.

[19] 0. Ulusoy and G. Belford, "Real-Time Lock Based Concurrency Control in a Distributed Database System", Proc. Of 12th Zntl. Conf. on Distributed Computing Systems, 1992.

[20] C. Mohan, B. Lindsay and R. Obermarck, "Transaction Management in the R* Distributed Database Management System", ACM TODS, 11(4), 1986.

[21] R. McFadden, Jeffrey A. Hoffer, "Modern Database Management", fourth edition, The Benjamin/Cummings publishing company Inc, 1991.