Design and Develop New Concurrency Protocol for Distributed Real Time Database System

Ayushi Gupta, Divya Srivastava, Ankit Kanaujia, Dr. Sanjeev Kumar Singh

Student, Information Technology Dept. Of Galgotias College of Engg. and Tech., Greater Noida, Uttar Pradesh, India

ayushisworkingdesk@gmail.com

Student, Information Technology Dept. Of Galgotias College of Engg. and Tech., Greater Noida, Uttar Pradesh, India

srivastavad955@gmail.com

Student, Information Technology Dept. Of Galgotias College of Engg. and Tech., Greater Noida, Uttar Pradesh, India

ankit.rank1@gmail.com

Head Of Department, Information Technology Dept. Of Galgotias College of Engg. and Tech., Greater Noida, Uttar Pradesh, India

sksingh@gmail.com

ABSTRACT

The issue of need reversal happens when a high need task is required to sit right for finishing of some other assignment with low priority because of contention in getting to the common framework resource(s). This issue is examined by numerous scientists covering a wide scope of examination territories. A portion of the key exploration territories are ongoing working frameworks, continuous frameworks, constant databases, and appropriated constant databases. Independent of the application region, in any case, the issue lies with the way that need reversal must be controlled with no strategy accessible to dispose of it altogether. In this part, the need reversal related booking issues and exploration endeavors toward this path are talked about. Various methodologies and their viability to determine this issue are systematically analyzed. At long last, significant examination achievements to date have been summed up and a few unanswered exploration questions have likewise been recorded. High priority two phase locking (HP2PL) concurrency control algorithm can be used for accessing of data items to resolve conflicts amongst the concurrently executing transactions. Inclusion of priority inversion and data inaccessibility are the most undesirable problems in transactions’ execution which seriously affect the system performance. Previously developed protocols for resolving such issues put a lot of messages and time overhead which is not desirable. In distributed real-time database system (DRTDBS), basic aim is to minimize the number of transactions missing their deadline. This can be achieved by minimizing commit time. In this paper, A One Phase Priority Inheritance Commit Protocol (OPPIC) has been proposed specifically reducing one round of message transfer among coordinator and participating cohorts’ sites in case of priority inversion problem at any of the cohort of low priority transaction. Focus of this protocol is to minimize the priority inversion duration that in turn minimizes commit time. A distributed database system is simulated for measuring the performance of this protocol with 2PC and PIC protocols. The results confirm the significant improvement in system performance with the OPPIC protocol.

Keywords: Database management system, Distributed System, DDBS, Real Time Database System, Query Process, Concurrency Control Protocols.
1. INTRODUCTION

We are living in a real time world with access to real time applications. Real time applications needs a database management system where the database system is not only fast but can also handle concurrent requests at a time. In order to handle concurrent requests a database system must have priority assignment policies.

Now let us understand it in context of databases, a single logical operation on data is called as transaction. Organizations which are worldwide they distribute their data among local databases and centralized database system. These kind of databases are called Distributed Database management System and when these operations execute with predictable response and with application acceptable levels of logical and temporal consistency of data, in addition to timely execution of transactions with the ACID (Atomicity, Consistency, Isolation, Durability) properties then this kind of database system is called as Real Time Database management system(RTDBS).

In RTDBS systems, it is very important to design an efficient commit protocols in order to avoid data inconsistency. RTDBS should have such protocols where if for a particular data item we have multiple requests then it should order all those requests in such a way so that the process should be completed in shortest time while maintaining ACID properties of a transaction.

In this paper we have tried to explain about what is DBMS , RTDBS , transactions, concurrency control techniques etc. And we have proposed a new theory of assigning priorities to the transactions in Real Time Distributed Database System.

DISTRIBUTED REAL TIME DATABASE

As the world is growing in all it's technical aspects we can feel the urge of the requirement for real-time data services in a distributed environment. The electronics services and electronic commerce applications that we are using in today's world are characterized by a high volume of transactions and probably cannot survive without the online support of computer systems and database technology. Many applications such as Aircraft control, Shipboard control, Military tracking, Stock arbitrage system, Sensory system, and Traffic control, the transaction should be processed within their deadlines using the fresh data reflect the control real-world status. The existing and emerging application requires a distributed database system (DDBS) consists of a collection of sites, connected together via some communication network, in which each site has a database system site in its own right but the site is agreed to work together. This is the secret behind the users accessing the data in the same way as they had the enormous data stored in their own system.

As the world is growing in all it's technical aspects we can feel the urge of the requirement for real-time data services in a distributed environment. The electronics services and electronic commerce applications that we are using in today's world are characterized by a high volume of transactions and probably cannot survive without the online support of computer systems and database technology. Many applications such as Aircraft control, Shipboard control, Military tracking, Stock arbitrage system, Sensory system, and Traffic control, the transaction should be processed within their deadlines using the fresh data reflect the control real-world status. The existing and emerging application requires a distributed database system (DDBS) consists of a collection of sites, connected together via some communication network, in which each site has a database system site in its own right but the site is agreed to work together. This is the secret behind the users accessing the data in the same way as they had the enormous data stored in their own system.

The presence of multiple sites in the distributed environment raises the issue that is not present in a centralized system. The Real-Time System (RTS) are those systems for which correctness depends not only on logical properties of the produced result but also on the temporal properties of these results. Typically RTS is associated with critical application in which human lives or expensive machinery may be at stake. Hence in such a system, an action performed too late (or too early) or a computation that uses temporal invalid data may be useless and the same time harmful even if such action or computation is functionally correct. As RTS keeps on evolving their applications they become more and more complex. Many of these applications providing real-time data services in a distributed environment are essential. The issues involved in providing predictable real-time data services in a centralized database system have researched as a Distributed Real-Time Database System (DRTDBS). DRTDBS is a collection of multiple interrelated databases distributed over a computer network where transactions have explicit timing constraints usually in the form of deadline. In such a system data items must be controlled in order to maintain databases logically consistency satisfying the timing constraints of various real-time activities. In Distributed System may be difficult due to the distributed nature of the transaction and database consistency required. The implementation of Distributed Real-Time Database Systems
Suppose with properties simultaneously. Concurrency Control protocols are there to ensure that if multiple transactions are executed simultaneously while maintaining the ACID properties of the transactions and serializability in the schedules. Let us understand Concurrency control problem with the help of one simple example –

Suppose you are booking a flight ticket and there is only one seat left. At the time you are booking the ticket there will be 100’s other traveller who might be booking the same seat. But without concurrency control, it’s possible that many people will end up buying the ticket for the same seat.

In these kind of scenarios the concept of Concurrency Control comes, so that when multiple users are accessing the same object, then concurrency control will allow only one user to make changes and preserve the data accuracy.

We can face many problems if concurrent transactions are executed in an uncontrolled manner. The following are the three problems in concurrency control.

1. Lost updates
2. Dirty read
3. Unrepeatable read

1) Lost update- Lost update problem arises when two transactions that have access to the same database items contain their operations in such a way that makes the value of any object, column or row in database item incorrect, then the lost update problem occurs.

If two transactions T1 and T2 read a record and then update it, then the effect of updating the first record will be overwritten by the second update.

<table>
<thead>
<tr>
<th>Transaction-X</th>
<th>Time</th>
<th>Transaction-Y</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>$t_1$</td>
<td></td>
</tr>
<tr>
<td>Read A</td>
<td>$t_2$</td>
<td></td>
</tr>
<tr>
<td></td>
<td>$t_3$</td>
<td>Read A</td>
</tr>
<tr>
<td>Update A</td>
<td>$t_4$</td>
<td>Update A</td>
</tr>
<tr>
<td></td>
<td>$t_5$</td>
<td></td>
</tr>
</tbody>
</table>

Here,

- At time $t_2$, transaction-X reads A’s value.
- At time $t_3$, Transaction-Y reads A’s value.
- At time $t_4$, Transactions-X writes A’s value based on the value seen at time $t_2$.
- At time $t_5$, Transactions-Y writes A’s value based on the value seen at time $t_3$.
- If you notice at time $t_5$, the update of Transaction-X is lost because Transaction-Y overwrites it without looking at its current value.

Our distributed real-time database (DRDTDBS) model consists of a number of inter-connected sites. It is assumed that the sites are fully connected to avoid complicating our analysis due to the different network configurations. Each site is a local database system which consists of a transaction generator, a scheduler, a CPU, a ready queue, a local database, a communication interface, and a block queue.

Two types of transactions are defined in the model, global and local. Local transactions only access data objects in its originating site. They create no transactions. Global transactions, on the other hand, consist of a series of sub-transactions. A global transaction consists of one or more operations. The objective is to determine the priorities (deadlines) of the sub-transactions so that the percentage of missed deadlines is kept as low as possible.

The processing of the operations for both local transactions and sub-transactions of global transactions are the same. It involves CPU computations and database accesses. In order to eliminate the impact of disk scheduling on the system performance, we assume that the database is the main memory resident. For global transactions, if a sub-transaction requests a remote object, the sub-transaction will be transmitted to, and processed by the remote site through the communication network. To simplify the model, we assume that each sub-transaction has one operation and each operation only accesses one data object.

At each site, the CPU is scheduled using preemptive EDF. Outstanding operations are maintained in the ready queue in priority order.

WHAT IS CONCURRENCY CONTROL PROTOCOL AND WHY IS IT IMPORTANT FOR DATABASE SYSTEM?

Concurrency control protocols are there to ensure that if multiple transactions are executed simultaneously while maintaining the ACID properties of the transactions and serializability in the schedules.
Such type of problem is known as the Lost Update Problem as an update made by one transaction is lost here.

2) Dirty Read

- The dirty read occurs in the case when one transaction updates an item of the database, and then after updating the value the transaction fails due to some reason and then updated database item is accessed by another transaction before the updated value is changed back to the original value.

- A transaction T1 updates a record that is read by T2. If T1 aborts then T2 now has values that have never formed part of the stable database.

<table>
<thead>
<tr>
<th>Transaction X</th>
<th>Time</th>
<th>Transaction Y</th>
</tr>
</thead>
<tbody>
<tr>
<td>—</td>
<td>t1</td>
<td>—</td>
</tr>
<tr>
<td>—</td>
<td>t2</td>
<td>Update A</td>
</tr>
<tr>
<td>Read A</td>
<td>t3</td>
<td>—</td>
</tr>
<tr>
<td>—</td>
<td>t4</td>
<td>Rollback</td>
</tr>
<tr>
<td>—</td>
<td>t5</td>
<td>—</td>
</tr>
</tbody>
</table>

- At time t2, transaction-Y writes A’s value.

- At time t3, Transaction-X reads A’s value.

- At time t4, Transactions-Y rollbacks. So, it changes A’s value back to that of before t1

- So, Transaction-X now contains a value that has never become part of the stable database.

- Such type of problem is known as Dirty Read Problem, as one transaction reads a dirty value that has not been committed.

3) Inconsistent Retrievals Problem

- The inconsistent Retrievals Problem is also known as unrepeatable read. When some transaction calculates a summary function( any function which calculates summary) over a set of data while the other transactions are updating the data, then the Inconsistent Retrievals Problem occurs.

- Suppose there is a transaction T1, which reads a record and then does some other processing during the time when transaction T2 updates the record. So now if at this time transaction T1 reads the record, then the new value will be inconsistent with the previous value.

Example:

Suppose two transactions operate on three accounts.

<table>
<thead>
<tr>
<th>Account-1</th>
<th>Account-2</th>
<th>Account-3</th>
</tr>
</thead>
<tbody>
<tr>
<td>Balance = 200</td>
<td>Balance = 250</td>
<td>Balance = 150</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Transaction-X</th>
<th>Time</th>
<th>Transaction-Y</th>
</tr>
</thead>
<tbody>
<tr>
<td>Read Balance of Acc-1</td>
<td>t1</td>
<td>—</td>
</tr>
<tr>
<td>Read Balance of Acc-2</td>
<td>t2</td>
<td>—</td>
</tr>
<tr>
<td>Sum &lt;- Sum + 250 = 450</td>
<td>t3</td>
<td>—</td>
</tr>
<tr>
<td>—</td>
<td>t4</td>
<td>Read Balance of Acc-3</td>
</tr>
<tr>
<td>—</td>
<td>t5</td>
<td>Update Balance of Acc-3 150 -&gt; 150 - 50 -&gt; 100</td>
</tr>
<tr>
<td>—</td>
<td>t6</td>
<td>Read Balance of Acc-1</td>
</tr>
<tr>
<td>—</td>
<td>t7</td>
<td>Update Balance of Acc-1 200 -&gt; 200 + 50 -&gt; 250</td>
</tr>
<tr>
<td>Read Balance of Acc-3</td>
<td>t8</td>
<td>COMMIT</td>
</tr>
<tr>
<td>Sum &lt;- Sum + 250 = 550</td>
<td>t9</td>
<td>—</td>
</tr>
</tbody>
</table>

- In the above table, Transaction-X is doing the sum of all the balances an at this time transaction-Y is transferring an amount 50 from Account-1 to Account-3.

- Because of this, transaction-X produces the result of 550 which is incorrect. If we write the produced result by transaction-x in the database then the database will become an inconsistent state because the actual sum is 600.

- Hence, transaction-X is seeing the inconsistent state of the database.

So to avoid all the above problems we need Concurrency Control Protocol techniques.

ISSUES WITH THE CURRENT CONCURRENCY CONTROL PROTOCOL IN THE DISTRIBUTED REAL TIME DATABASE SYSTEM

Previously different Concurrency control protocols have been proposed for simultaneously proceeding transactions based on the real time requirements of the tasks. These includes the ultimate deadlines of their tasks, slack time and number of locks. In these concurrency control protocols, it is assumed that EDF that is the end of time is used for local CPU Scheduling. Here, let us briefly summarize these heuristics. In the following description, we assume that different m distributed transactions, T1, T2, ..., Tm, to be executed in series. We also denote the
The deadline of a (sub-) transaction X by dl(X). The four heuristics suggested till date are:
> Ultimate deadline (UD);
> Effective deadline (ED);
> Equal slack (EQS); and
> Equal flexibility (EQF).
The most automatic way to determine the priorities of a transaction is to follow the priority. Thus, a transaction Ti is assigned the same priority as that of the transaction from which it has evolved. That is,

\[ dl(Ti) = dl(T) \]

The problem of UD is that it does not consider the execution time of a transaction. A transaction with farthest deadline is given the lowest priority even if it does not have any slack. Under ED, the deadline of a transaction Ti is the ultimate deadline minus the total predicted execution time of the transaction Ti. That is,

\[ dl(Ti) = dl(T) - pex(Ti) \]

where pex(Ti) is the predicted execution time of transaction Ti.

The problem with UD and ED is that they assign all the current slack time to the current executing transactions. The transactions in the later stages of the process may find that they do not have sufficient slack time for their executions. In EQS and EQF, the total slack time of a transaction is shared by all the transactions. Under EQS, the slack is evenly distributed to the remaining transactions.

2. THEORY PROPOSED

Different heuristics have been suggested for concurrency control for the simultaneous transactions based on the real-time requirements of the tasks. These include the ultimate deadlines of their tasks and their slack times. Now the question is what is slack time? The difference between the scheduled completion date and the required date to meet the critical path is the amount of slack time available.

Till the date people have been ignorant around the number of write locks when talking of distributed real time database system. And also we should keep in mind the timestamp of the transaction. Timestamp is like a unique identifier that is created for every transaction when it was created. It is the time and date when the transaction was created. Refer to the timestamp of a transaction \( T \) as \( TS(T) \). To ensure this, use two Timestamp Values relating to each database item A. 

\( >>WTS(A) \) is the largest timestamp of any transaction that was created. 

\( >>RTS(A) \) is the largest timestamp of any transaction that executed read(A) successfully.

The main idea for this protocol is to stick around number of write locks used and the timestamp when the transaction was created. The purpose of considering write locks is basically that majority of conflicts are faced due to write locks only. We don't face conflicts under reading circumstances on normal cases. Here we will be using NW(A) for annoting the number of write locks that must be required for pursuing with the transaction A and NR(A) for number of read locks that will be required lately.

**How will the protocol work?**

Basically, we had noticed that concurrency control protocols are similar to a scheduling algorithm where we have to assign data items that are like resources and need to prevent abortion or rolling back of transactions.

The next idea is to include the number of write locks and timestamp along with slack time and EDF.

<table>
<thead>
<tr>
<th>Transaction i</th>
<th>Slack time (s)</th>
<th>Number Of write Locks NW(i)</th>
<th>Timestamp TS(i)</th>
</tr>
</thead>
<tbody>
<tr>
<td>T1</td>
<td>2s</td>
<td>4</td>
<td>2020-06-19 03:13:07 UTC</td>
</tr>
<tr>
<td>T2</td>
<td>10s</td>
<td>9</td>
<td>2020-06-19 03:14:17 UTC</td>
</tr>
<tr>
<td>T3</td>
<td>4s</td>
<td>1</td>
<td>2020-06-19 03:13:07 UTC</td>
</tr>
<tr>
<td>T4</td>
<td>7s</td>
<td>6</td>
<td>2020-06-19 03:14:07 UTC</td>
</tr>
<tr>
<td>T5</td>
<td>13s</td>
<td>0</td>
<td>2020-06-19 03:14:07 UTC</td>
</tr>
</tbody>
</table>

Table 1.1 Example of transactions with their slack time, number of write locks, and timestamps.

Using the above table as an example here is an explanation as to how the transactions will be allocated the priority and get scheduled in the concurrency control protocol. Let us assume...
that all the above transactions are requesting for the same data item.

Timestamp of T3 is the least which means that T3 will be processed first. Total time taken by T3 will be 1s and as we can see that after transaction T3 there is a lot of time interval when compared to timestamp so we can say that when T1, T4, T5 will be compared by that time data items will be unlocked. Since T1 has least slack time so we will proceed with T1 and by that time T4 and T5 will wait to get hold in the data items.

T4 will take total of 9s where waiting time and slack time is included and T5 will take 22s. While the transaction T5 is processing T2 requests for the data item locking. Since data items were already being used by T5 therefore T2 had to wait and took 31s in itself to complete the transaction.

The following bar chart show how much time a particular transaction will take to commit.

The following Gantt chart could ease the study of time taken by each transactions.

3. Algorithm Proposed

1. If TS(i) = TS(j) i.e. timestamp is same for arrival of both the transactions, go to next step.
2. Check for the slack time for both the transactions Ti and Tj.
3. Choose the transaction with least slack time ST.
4. Lock all the data items before processing the transaction required by the transaction.
5. Start the query on stored procedure.
6. As the use of data items finishes while processing the SPs.

The order in which Data locks are possessed is shown by the given graph:

![Data lock pattern](image)

**4. FUTURE SCOPE**

Although a lot of work has been done on various concurrency control protocols in a distributed environment and distributed real-time environments distinctly, still there are many unresolved issues that need further investigation. Some of these issues are as follows:

I. Performance evaluation is important to examine when it comes to DRTDBS.
II. Concurrency control protocols proposed for real-time systems can be extended for Mobile DRTDBS, where, battery power, communication bandwidth, and memory are limited.
III. Logical integration of the concept of replication with existing protocols can also increase the overall performance.
IV. For grid database systems, new protocols are required for fulfilling the specific needs of such an environment

**CONCLUSION**

The database system is one of the most active research areas in the field of computer science. This paper reviewed the work carried out in the area of concurrency control protocol in a distributed real-time environment. The use of timestamp and slack time is described for distributed database systems. A representative set of real-time concurrency control protocols have also been proposed which can be specifically used in distributed real-time environments. Most of the distributed real-time concurrency control protocols try to focus just on earliest deadline but we should work more around number of locks required and timestamp along with it. Some other research directions which may improve the performance as optimized memory use, creating less temporal objects need to be investigated for getting better results.

**REFERENCES**