Non-Blocking Approach in Concurrency Control

Minakshi Sangwan
Department of Computer Science and Engineering
Shree Ram College Of Engg. & Management, Palwal.

Abstract—Majority of the research in database management system on conventional techniques requires additional efforts for deadlock detection and elimination. A deadlock occurs when a transaction waits for locks held by another transaction, which in turn, is waiting (directly or indirectly) for locks held by the first transaction. In this paper we have proposed a method for non blocking approach in concurrency control. The proposed approach is a distributed approach for construction of data access graphs at data sites, within a distributed static locking environment.

Keywords—Deadlock, Distributed database management system, Local Access Graph, Data Access Graph.

I. INTRODUCTION

In a distributed database system, database accesses by various transactions are synchronized in order to preserve database consistency. The most commonly used algorithms are based on two phase locking. The distributed two-phase locking techniques can be further divided into two classes; static locking and dynamic locking. In static locking, all lock requests are granted prior to start of transaction execution. In dynamic locking, a lock request is issued whenever a data item is needed during execution. In a distributed database system, where transmission delay may be substantial, static locking schemes are preferable to dynamic ones, as these allow concurrent transmission of lock requests. A deadlock occurs when a transaction waits for locks held by another transaction, which in turn, is waiting (directly or indirectly) for locks held by the first transaction. In such a case, one of the deadlocked transactions must release all its locks and send out the lock requests again. Deadlock detection is difficult in a distributed database environment, because no site has complete and up-to-date information about the system. Some algorithms detect deadlocks by finding cycles in a transaction-wait-for-graph (a directed graph whose nodes represent transactions and arcs represent the wait-for relationships). The proposed approach is a distributed approach for construction of data access graphs at data sites, within a distributed static locking environment.

Example: In a distributed system, scheduling can be carried out by constructing partial graphs at different sites for individual transactions. Such a data flow graph is referred to as local access graph (LAG) or data access graph (DAG). Based on the approach, a lock request (LRI) of transaction (Ti) is sent to the concerned sites. At the site, the LAG is formed. A LAG of Ti at site Sk contains the edges (<Tj, Ti>) of all Tj such that, both Ti and Tj have a conflict on some data items residing at Sk. The possibility of deadlocks is removed by eliminating the existence of odd edges.

In the proposed algorithm, transaction Ti arrives at a site. Transaction is executed and its read_set and write_set are identified. These are sent to visit a (majority) number of sites. During the visit, if no transaction is found to be in conflict with transaction Ti, then it returns, and commits the update values. In case of a conflict, the conflicting transactions are ordered with the help of their own TDFGs. This process results in a non blocking protocol.

II. NON-BLOCKING BASED CONCURRENCY CONTROL SYSTEM

The distributed database system (DDBS) consists of a set of data items (set D). A data item is the smallest accessible unit of data. It may be a file, a record, an array, or an object. X, Y, . . . , Z represent data items. Each data item is stored at one site only. Transactions are represented by Ti, Tj, . . . ; and sites are represented by Si, Sj, . . . ; where i, j, . . . are integer values as shown in fig. 1. The data items are stored at database sites connected by a computer network. Each site supports a transaction manager and a data manager. The transaction manager supervises the execution of the transactions. The data managers manage individual databases. The network is assumed to detect failures, wherever these occur. When a site fails, it simply stops running and other sites detect this fact. The communication medium is assumed to provide the facility of message transfer between sites. A site always hands over a message to the communication medium, which delivers it to the destination site in finite time. For any pair of sites Si and Sj, the communication medium always delivers the messages to Sj in the same order in which they were handed to the medium by Si.
A transaction is modeled as a sequence of read and writes operations. A read only transaction does not issue write operations. The transactions represent complete and correct computations, i.e., if executed alone on a consistent database, these leave the database in a consistent state. The items read by a transaction constitute its read-set (RS), and the items written by it constitute its write-set (WS). Both read-set and write-set constitute transaction variables (TVs). We assume that TVs of a transaction are fully determined at the end of an initial computation. Two transactions $Ti$, $Tj$ are said to have R-W, W-R or W-W conflict, if $RS(Ti)$∩$WS(Tj)$≠ϕ ; $WS(Ti)$∩$RS(Tj)$≠ϕ ; or $WS(Ti)$∩$WS(Tj)$≠ϕ ; respectively. Also, both $Ti$, $Tj$, are said to be in conflict (say $TinTj$≠ϕ), if at least one of the above conflicts exists between them.

Static locking technique is assumed. Accordingly, the LVs are always predeclared. The transactions may be local (LVs reside at the site of origin), or global (a part of LVs resides at another site). The lock requests for data items are sent to the sites. The sites include the request in their local access graphs, and complete the process. After a lock is granted, a grant message (with data) is sent to the requesting site. Local computation starts after all lock grants have been received. After the computation, the write phase is initiated. Updated data items are sent to the data sites and are stored in a temporary memory. In the commit phase, updated values are written into the database.

**Transaction number (TN):** Every site $Si$ has a logical clock $Ci$, which takes a monotonically non decreasing integer value. A TN is assigned to the transaction $Ti$, on its arrival, by a site $Si$. It is triple element value, as $(S, I, C)$. $S$ is a site identifier ($Si$). The $I$ field is the transaction identifier, which is the value of the local clock $(Ci)$ at the instant of $Ti$’s arrival at site $Si$. The $C$ field is used to synchronize different clocks, in a manner described below.

For any transaction message, $T(S, I, C)$ being sent from site $Si$ to site $Sj$;

$C := Ci$ of the message sender site.

$Cj$ of the site $Sj$, on receipt of a transaction $T(message)$:

$Cj := \max(C + 1, Cj)$

$Cj$ of the site $Sj$, before dispatch of a transaction $T(message)$:

$Cj := Cj + 1$.

Let, $TN_1 = (S_1, I_1, C_1)$ and $TN_2 = (S_2, I_2, C_2)$, then any pair of TNs can be compared using the following criteria.

Equal to (=) : $TN_1 = TN_2$, if and only if, $S_1 = S_2$ and $I_1 = I_2$.

Greater than (>) : $TN_1 > TN_2$, if and only if,

$I_1 > I_2$, or 
$(I_1 = I_2$ and $S_1 > S_2)$;

Less than (<) : $TN_1 < TN_2$, if and only if,

$I_1 < I_2$, or 
$(I_1 = I_2$ and $S_1 < S_2)$;

**III. CONSTRUCTION OF LOCAL ACCESS GRAPH**

The transactions are ordered by constructing local access graphs (LAGS) for access requests.

**Directed graph:** A directed graph $G$ consists of a set of vertices $V = V_1, V_2, \ldots$, a set of edges $E = E_1, E_2, \ldots$, and a mapping function $\Psi$ that maps every edge on to some ordered pair of vertices $< Vi, Vj >$. A pair is ordered, if $< Vi, Vj >$ is different from $< Vj, Vi >$. A vertex is represented by a point, and an edge is represented by a line segment between $Vi$ and $Vj$ with an arrow directed from $Vi$ to $Vj$.

Insertion of an edge $(Vi, Vj)$ into the graph $G = (V, E)$ results in graph $G’ = (V’ , E’)$, where $V’ = V \cup \{Vi, Vj\}$ and $E’ = E \cup \{< Vi, Vj >\}$. The union of two graphs $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$ is another graph $G_3$ (written as $G_3 = G_1 \cup G_2$), whose vertex set is $V_3 = V_1 \cup V_2$ and the edge set is $E_3 = E_1 \cup E_2$.

Let, $< < \tau$ be the partial ordering relation over T.

**Access Graph:** An access graph of $Ti$ ($AGi$) is a graph $AGi(V, E)$, where $V = C T$, and $E = \{< Ti, Tj > | LV \ n LV’ \neq \phi$ and $Tj < < \tau Ti\}$

**Example:**

Consider the transactions $Ta$, $ Tb$, $Tc$, $Td$ and $Te$, as shown below. Let $X$, $Y$, $Z$ be data items. Also,

- $Ta = Ra(X)Ra(Y)Wa(X)Wa(Y)$
- $Tb = Rb(X)Rb(Y)Wb(X)Wb(Y)Wb(Z)$
- $Tc = Rc(Z)Wc(Z)$
Consider a situation, where X, Y and Z are located at one site. The execution of above transactions' operations can follow any one of the sequences as per the criteria of serializability. All such executions are correct executions. For each execution equivalent to a serial execution), the AGs of transactions can be different. If we consider arrival pattern of transactions in the order Ta, Tb, Tc, Td and Te, then, the corresponding AGs of above transactions are shown in fig.2. In this, Ti -> Tj (with x above the arrow) indicates, Tj is waiting for data item X which will be released after completion of Ti.

**Local Access Graph (LAG):** A local access graph (LAG) of Ti, at Sk, is a graph $LAG_i(V, E)$, where, $V C T$, and $E = \{<T_j, Ti, > | LVjk n LVik ≠ φ and Tj << Ti\}$. In this expression, Tj has previously visited site Sk, and LVik denotes the part of LVj, resident at Sk.

Example: Consider the case of a distributed system where, data items X, Y and Z are stored at S1, S2 and S3, respectively. The LR of a transaction forms LAGs at these sites. The LAGs are shown in figure 2.

The home site of Ta is assumed to be S1; for Tb, Te, it is S2; and for Tc, Td it is S3.

In fig 2, edge $< Ti, Tj >$ denotes that, both Ti and Tj have a conflict on some data item.

So, Tj is ordered for an access after Ti. After receiving Ti’s updates for the conflicting data item, Tj can access it. At the beginning, Ta gets an access to the data items X, Y at S1, S2 respectively, and executes at S1. Next, Tb gets an access to the data items X, Y, Z at S1, S2, S3 respectively, and executes at S2. Similarly, transactions Tc, Td execute at S3 and Te, executes at S2, after receiving the updates of preceding transactions. Thus, the equivalent serial order obtained in this way is indicated by: $Ta<< Tb<< Tc<< Td<< Te$

**IV. ALGORITHM TO CONSTRUCT LOCAL ACCESS GRAPH**

Formal description of the algorithm: Terms used to describe the Algorithm are:

- **Home site (SHi):** The site of origin of Ti is referred to as the home site. It is represented by SHi.
- **Transaction number (TNi):** A unique number (TNi) is assigned to the transaction Ti on its arrival at the home site.
Locking variables (LVi/LVk): The items read or to be written by a Ti, constitute the LVi. The locking variables at Sk constitute LVik.

Lock request (LRik): It consists of TNi and LVik. It is prepared by SHi on arrival of Ti and is sent to each concerned site Sk.

Odd edge, Even edge: An edge < TJ, Ti >, such that TNj > TNi, is called as odd edge. Otherwise, it is called as an even edge.

Access grant status (AGSik): It has values 0 or 1. After granting of all the requested locks of data items at Sk to LRik, the AGSik is changed to 1 at site Sk. Otherwise, AGSik is 0 for waiting transactions.

Active list: The Active list is maintained by each Sk. The Active list of Sk is divided into two tables: active list of lock requests at Sk (ALTk), and active list of LAGs at Sk (ALGk). These tables are:

\[ ALTk = \{(LVi, AGSik) \mid Ti, requested data items at Sk\} \]

\[ ALGk = \{LAGik \mid Ti \text{ requested data items at Sk} \} \]

A transaction Ti is inserted into the ALTik, after initializing AGSik. On getting the access grants for LVik, AGSik is changed to 1. As a next step, these access grants are sent to SHi.

Data table (DTi): This table is maintained at the SHi for each Ti. The DTi contains the lock grants (with values) of Ti. Whenever Sk receives any lock grant from another site, it stores it in a corresponding DTi.

Status of a transaction (STi): For a Ti, STi is maintained at the SHi. It has values 0 or 1. Initially, STi is 0. After receiving all lock grants, STi is changed to 1. After this, the execution of Ti begins.

Informal description of the algorithm: In this algorithm, whenever a transaction arrives at Si, its LRik are prepared and are sent to each concerned site Sk. The LAGik is prepared at these sites. At any site Sk, if LAGik contains odd edge < TJ, Ti >, then it is confirmed by checking the existing AGSj locally, or by consulting the SHj. That is, if the AGSj is 0, then an even edge < Ti, TJ > is inserted into the LAGjk and odd edge < TJ, Ti > is deleted from LAGik. Otherwise, at the SHj, if TJ is under execution, then the odd edge < TJ, Ti > is not deleted from the LAGik. Otherwise, the even edge < Ti, TJ > is inserted into the LAGjk, and odd edge < TJ, Ti > is deleted from LAGik. Thus, odd edges are confirmed to eliminate the possibility of a deadlock.

V. ILLUSTRATIVE EXAMPLE

Consider the transactions Ta, Tb, Tc, Td and Te, as shown below. Let X, Y, Z be data items. Also,

- \( \text{Ta} = \text{Wa}(X) \text{Wa}(Y) \text{Wa}(Z) \)
- \( \text{Tb} = \text{Rb}(X) \text{Rb}(Z) \text{Wb}(X) \)
- \( \text{Tc} = \text{Rc}(Y) \text{Rc}(Z) \text{Wc}(Y) \)
- \( \text{Td} = \text{Rd}(Z) \text{Wd}(X) \text{Wd}(Z) \)
- \( \text{Te} = \text{Re}(X) \text{Re}(Y) \text{Re}(Z) \)

Consider a situation, where X, Y and Z are located at one site. The execution of above transactions’ operations can follow any one of the sequences as per the criteria of Serializability.

Consider a situation, where X, Y and Z are located at one site. The execution of above transactions’ operations can follow any one of the sequences as per the criteria of Serializability. All such executions are correct executions. For each execution equivalent to a serial execution), the AGs of transactions can be different. If we consider arrival pattern of transactions in the order Ta, Tb, Tc, Td and Te, then, the corresponding AGs of above transactions are shown in fig.4. In this, Ti -> TJ (with x above the arrow) indicates, Ti is waiting for data item X which will be released after completion of Ti.

Now, consider the case of a distributed system where, data items X, Y and Z are stored at S1, S2 and S3, respectively. The LR of a
transaction forms LAGs at these sites. The LAGs are shown in figure 2.

<table>
<thead>
<tr>
<th>Site</th>
<th>LAGx,site</th>
<th>LAGy,site</th>
<th>LAGz,site</th>
<th>LAGw,site</th>
<th>LAGv,site</th>
</tr>
</thead>
<tbody>
<tr>
<td>S1</td>
<td>Ta</td>
<td>Tb</td>
<td>Ti</td>
<td>Tj</td>
<td>Te</td>
</tr>
<tr>
<td>S2</td>
<td>Ta</td>
<td>Tb</td>
<td>Ti</td>
<td>Tj</td>
<td>Te</td>
</tr>
<tr>
<td>S3</td>
<td>Ta</td>
<td>Tb</td>
<td>Ti</td>
<td>Tj</td>
<td>Te</td>
</tr>
</tbody>
</table>

Fig. 5 LAGs of active transactions at various sites.

The home site of Ta, Tb is assumed to be S1; for Te, it is S2; and for Tc,Td it is S3.

In fig.2, edge < Ti, Tj > denotes that, both Ti and Tj have a conflict on some data item.

So, T j is ordered for an access after Ti. After receiving Ti’s updates for the conflicting data item, Tj can access it. At the beginning, Ta gets an access to the data items X, Y, Z at S1, S2 and S3 respectively, and executes at S1. Next, Tb gets an access to the data items X, Z at S1, S3 respectively, and executes at S1. Similarly, transactions Tc, Td execute at S3 and Te, executes at S2, after receiving the updates of preceding transactions. Thus, the equivalent serial order obtained in this way is indicated by : Ta< Tb< Tc< Td< Te

VI. PERFORMANCE COMPARISON

In the existing voting based approaches, an update request is executed, and the update variables are sent to other sites for majority approval. If the submitted transactions are in serializability conflict, then some of the transactions are rejected. We have proposed a technique that generates a partial local access graph (LAG) during the visits to sites, by the transaction. These LAGs are used to achieve one-copy serializability. In the proposed algorithm, the possibility of a transaction rejection or abort is reduced. On the whole, this algorithm works well in cases of conflicts, and a large number of copies with replication. In order to get performance benefits due to weighted voting, the sites may be assigned different weights. In case of a conflict among ‘n’ transactions T1, T2, ..., Tn, transaction T commits on its return. Transaction T2 must validate or compute new values again, and so on. In this way, a drawback of the voting based approach, such as repeated roll-back, is eliminated. The algorithm provides a framework for obtaining more deterministic estimates of processing time for computations.

VII. CONCLUDING REMARK

In the distributed locking based approaches, if transactions from different sites, are in serializability conflict, then some of the submitted transactions are rejected. In these systems, deadlock removal requires extra messages, and processing time. Also, the transactions are resubmitted for execution, and by this, these incur additional processing delays and overheads. In the proposed technique, data access graphs are constructed at every site, which result in avoidance of deadlocks. Hence, transaction rejections, or aborts do not occur.

REFERENCES