MTA Számítástechnikai és Automatizálási Kutató Intézete, Közlemények 27/1982

A NEW TIMING BASED ALGORITHM FOR CONCURRENCY CONTROL OF DISTRIBUTED DATABASES

ANDRÁS POGÁNY

Computer and Automation Institute
Hungarian Academy of Sciences

1. INTRODUCTION

In the last few years very high research effort has been devoted to the development of new concurrency control algorithms.

The goal of concurrency control is to ensure database consistency despite of paralel database accesses. The problem is presented by an example:

The database is composed of three records A,B,C.
The consistency criteria is A=B+C.
There are two accessesl and access2.
accessl: A=A+1, B=B+1
built up from the following steps
a1/ read A
b1/ write A
c1/ read B
d1/ write B
access2: B=B-1, C=C+1
built up from the following steps
a2/ read B
b2/ write B
c2/ read C
d2/ write C

In spite of that each access executed alone, preserves database consistency but, the next paralel execution for example will destory that: a1, b1, a2, c1, d1, b2, c2, d2.

The concurrency control algorithm deal only with those accesses which executed alone preserve database consistency. These accesses are callaed transactions. The step of a transaction is an access to a database element (a read or a write).

The sequence of transaction steps built up from the steps of a given transaction set is called a log. Some logs preserve database consistency while others do not. The task of concurrency control is to avoid the execution of those logs which destroy consistency.

Algorithms can be classified with respect to the permitted logs. Algorithms which result in not strictly serial logs (logs in which the steps of a transaction are side by side) are quite sophisticated and always need some preliminary information about the transactions. A typical example of this type is the algorithm of SDD-1 [1,2,7].

To achieve a strictly serial log seems to be a very simple task. What it only needs is to lock the database elements which are to be accessed, before the execution of a transaction. Here we note that the term strictly serial log often means it is strictly serial only with respect to the conflicting transactions (two transactions are said to be conflicting if one of them reads or writes a data element which is to be written by the other one). Some distributed methods have been developed to solve the problem of mutual exclusion, i.e. Agrawala [4], E. Chang [5], but generally the concurrency control algorithms do not use them. The reason is the fact that they increase the number of messages needed to execute a transaction.

Most concurrency control methods are optimistic. Issuing the transaction, they assume that the system does not contain any conflicting transactions. If in spite of this expectation there is a conflict then the algorithm ensures that there is at least one node where the conflicting transaction meet. This meeting results in suspending one of the transactions. This suspension either means waiting for the end of the other one or causes the death of the suspended transaction (which should be rolled back and started again). If sufficiantly great number of nodes (not necessarily all) with data element written by the transaction has been visited then the so called synchronization phase is terminated. The number of nodes required to achieve synchronization depends on the concurrency control algorithm which has been used. We might say, a transaction must visit as many nodes as necessary to meet all the possible conflicting transactions. If concurrency control works correctly then one and only one of the concurrent transactions can finish its synchronization phase. As a result of this concurrency determination method, the nodes should not execute the updates of a transaction until its synchronization phase is terminated. The nodes are informed about the termination of the synchronization by a message. This message is often called confirmation.

There are many methods known from the literature which really result in serial logs i.e. Thomas [8], Rosenkrantz et. al. [6]. They differ in the solution of synchronization, but confirmation is resolved always by messages.

The algorithm described in this paper has a different solution for the realization of confirmation. It uses timing instead of messages. In case of reliable network, in this way, a transaction can be executed with minimal number of messages.

2. ENVIRONMENT CONDITIONS

The conditions our algorithm works among are quite strong, but the methods applied are very simple. Later it will be shown how the conditions can be weakened or even left out while adding new features to the algorithm.

Conditions:

1/ fully duplicated database.

2/ A clock to every node. If the clock in node i shows $C_i(t)$ at the moment t then

$$\forall_{i,j}\ C_i(t)=C_j(t)$$

3/ The transactions must be moment-like. This means that the time of the first read and last write must be at the same instant.

4/ The network, used by the database should be reliable. This means that every node always works correctly and each node can have access to all other nodes.

5/ A time interval τ can be defined in the network so that for every pair of nodes the transmission time of a message from one node to another is always less than τ.

3. THE PRINCIPLE OF THE ALGORITHM

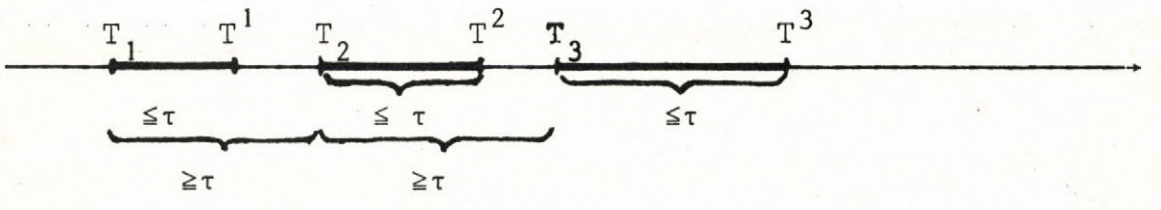
Independently from the concrete concurrency control method, to execute a transaction the minimum of one message per node is necessary. The time of a message exchange depends on the participant nodes, the type of message forwarding (broadcast, daisy-chain), the state of the network etc. There is no concurrency control algorithm which provides minimal execution time for every transaction on any network at any time. An optimal algorithm must be constructed in a way that permits an optimal implementation, that is the execution of a transaction should need at most one message per node and should not constraint the choice of the message forwarding method.

For algorithms which work by locking all the database elements accessed by a transaction, a sufficiant condition to achieve this locking is that the system executes only those transactions which are issued at a moment when no conflicting transaction is under execution. To decide whether there are transactions under execution conflicting with the transaction to be issued, the system has either to wait until the effect of conflicting transactions arrive at the node where the transaction is to be issued or to send inquires about transactions to every node. If we are to achieve an optimal solution then the former possebility is the better choice. The system has to wait before initiating a transaction as long as a message needs, in worst case, to arrive from the farthest node in time at the node where the transaction is to be initiated. This time is not more than τ, defined among the conditions.

An algorithm is to be constructed that matches the above mentioned considerations. This algorithm, in order to execute a transaction

a/ sends only one message to a node
b/ with arbitrary mode of message transferring, while
c/ the time interval between two conflicting transactions is at least τ.

The execution history of the transactions represented on a time axis will look like this:



T_i is the issue and
T^i is the execution time

To fulfil condition a/ the confirmation can not be done by explicit messages but can be done i.e. by timing. This timing is started at every node that receives a synchronization message.

To execute a transaction both of the following conditions must hold:

C1: Every node in the network is noticed of the transaction.
C2: At the time when the transaction is issued there are no conflicting transaction under execution.

If the node where the transaction is initiated, distributes the description of the transaction simultaneously with issue time (T_m) then C1 is fulfilled at the latest $T_m+\tau$. However, at time $T_m+\tau$ condition C2 is also hold because transactions conflicting T_m had to be issued in the interval $T_m-\tau$, T_m thus they arrived at each node till $T_m+\tau$.

To avoid concurrency let the strategy be the aborting and restarting of the transaction issued later.

Each node can check the concurrency independently from others. That is, if a node did not receive any transaction until $T_m+\tau$ that was concurrent with the transaction issued at T_m then conditions, C1, C2 are hold and the transaction of T_m can be executed.

With the terminology used heretofore, for a transaction issued at T_m, the synchronization will be finished at $T_m+\tau$ and the confirmation is the termination of timing τ started at T_m.

4. THE IMPLEMENTATION OF THE ALGORITHM

The issuing node assigns a timestamp (the value of the local clock at the local virtual execution of the transaction) to the transaction. Following this the transaction is compared, in the same way as described for an arbitrary node later, with the outstanding ones what have been received by the node. If the transaction has not been aborted (at the issuing node) then it is distributed in the network (timestamp which is the identifier, the read or written data elements).

The events of the system are: the termination of a timing, and the reception of a transaction.

Node actions taken at events:

A/ receiving the description of a transaction

1/ If the node does not contain any outstanding transaction conflicting with the received one then the received will be outstanding and a $\tau-(T_h-T_m)$ long timing is started. T_h is the local time at receiving the transaction. This timing will be referred to as primary timing.

2/ If the node contains any outstanding transaction conflicting with the received one then

i/ if $T_m < T_w$ (T_w is the issue time of the outstanding transaction)
the transaction timestamped by T_w is aborted and the transaction of T_m will be treated as described in point 1.;

ii/ $T_w + \tau > T_m > T_w$ the transaction of T_m is aborted

iii/ $T_m > T_w + \tau$ the transaction of T_m is treated as described in point 1..

B/ a primary timing is terminated
The transaction indicated in the timing is executed and an auxilary timing with interval τ is initiated.

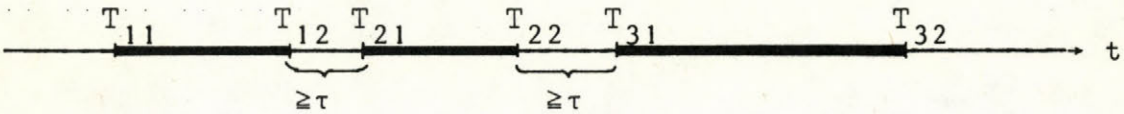
C/ an auxilary timing is terminated
The transaction indicated in the timing from this time onward is not outstanding.

At first sight, auxilary timing seems to be unnecessary. Though not unnecessary, it is not the only solution of the next implementation problem: each node at every time instant T_m must have the ability of checking every confliction it has got enough information for. Two conflicting transactions can meet at a node in such a way that the one issued earlier has been executed when the later arrives. Therefore the transactions, having been executed, must be reserved for a time interval τ.

5. MODIFICATIONS TO LEAVE OUT OR TO WEAKEN THE CONDITIONS

A. Eliminating the condition of moment-like transaction

If the reads and writes of the transactions form a finite notzero interval then the previous time axis representation shows the following picture. For the sake of simplicity the moment of execution is not displayed.



The T_{i1} is the time of the first data access of the transaction i while T_{i2} is the moment when the transaction is finished, that is, from this time the transaction can be distributed.

Theorem 1.

If T_{m2} is regarded as issue time then using T_{m2} instead of T_m the actions to be taken at events remain unchanged supposing that C3 is holding at the issue node.

C3: a transaction m is distributed only if the node did not receive any conflicting transaction during T_{m1}, T_{m2}.

Proof

For every pair of conflicting transactions in the network

1.1 $$T_{i2}-T_{j1}<\tau \rightarrow T_{i2}-T_{j2}<\tau$$

is true. This is because if $T_{i2}-T_{j2}>\tau$ then the node, where j is initiated, received the transaction i until T_{j2}. In this case C3 ensures that transaction j is not distributed. It follows from 1.1 that

1.2 $$T_{i2}-T_{j2}\geq\tau \rightarrow T_{i2}-T_{j1}\geq\tau$$

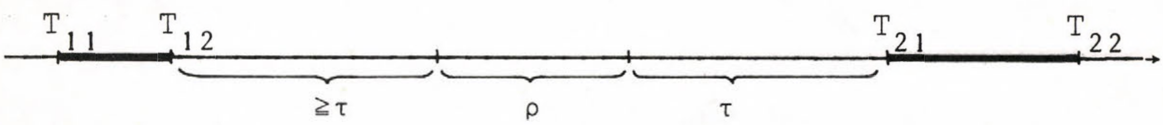
The condition part of this implication is true if the algorithm is used with the substitution $T_m=T_{m2}$.
q.e.d.

B. Eliminating the condition of reliable network

Instead of reliablity each node is expected to notice in ρ time the failure of its message.
Let's complete the algorithm with the following supplement:

S1: if a node regonises that its message (containing the descritption of a transaction) is undelivered then distributes an abort message. A node, having received an abort messages, does not execute the referred transaction.

To ensure a possible abort message to arrive at each node in time, instead of τ, $\tau+\rho+\tau$ should be used everywhere in the algorithm. With this modification the next execution log is obtained:



message distribution time — error recognition time — abort message distribution time

In spite of introducing abort messages, there may be some nodes (those whose separation from the network happens after receiving the transaction but before its corresponding abort message is obtained) which execute the transaction, causing inconsistency. Since -in case of network failure- the inconsistency is unavoidable, the algorithm must contain methods to recover the database according to the last proper state (the state resulted by the last properly executed transaction).

To solve the problem discussed above let's add the following supplements to the algorithm:

S2: a node interrupts its work until the system is recovered if it receives or generates an abort message.

S3: for every node, a log is maintained containing the identifiers of locally executed transactions.

Suplement S2 ensures every node to suspend its work in case of network failure sooner or later. Only a newly arised network failure can prevent the delivery of an abort message to a node which has got the transaction to be aborted. This means for a node that to the execution of each aborted transaction belongs a network failure. This sequence of failures approaches the node until, in the worst case, the node discoveres the failure itself. The maximum number of transactions a node can execute during the interval of network failure and its recognition can be calculated for each node from the number of nodes and from the topology of the network. Let K denote the largest value of the above counted maximums. Because each failure which prevents the delivery of an abort message separates at least one node from the network, a topology independent upper bound for K is equeal to n-1, where n is the number of nodes in the network.

Supplement S3 is used during database recovery. In an error free network every log contains the same identifiers. Logs diverge, when -because of failure- some nodes start an independent life (they execute transactions that were to be aborted). There is at least one node which detected the first failure of the network. The log and the database of this node are correct and this log will be a common slice of all logs. After system recovery, this log must be searched and on the basis of the database of the same node the whole database can be recovered. The sufficiant length of the logs to be kept at the nodes for the above search is K, while logs are circular lists of transactions in execution order.

To use the algorithm in an unreliable network, the $\tau=2\tau+\rho$ substitution can't be left out. The further discussion above describe only a possible extension of the algorithm in the field of unreliable networks. Certainly, there are other solutions of this problem and these considerations can be used to solve different problems.

C. Eliminating the condition of fully synchronized clocks

Instead of the hypotetical $\forall_{i,j}\, C_i(t)=C_j(t)$ heretofore the realizable $\forall_{i,j}\, |C_i(t)-C_j(t)|<\varepsilon$ condition will be used.

Because of asynchronous clocks the nodes aren't able to determine the exact time a transaction has spent in the network. This time determination has a twofold role: first to ensure the simultanious execution of a transaction at different nodes secondly, to be the base of concurrency checking. To modify the algorithm, first the clock conditions should be exactly defined.

C4: $\forall_{i,j} |C_i(t)-C_j(t)|<\varepsilon$

C5: $\sum_{i=1}^{n} C_i(t)/n=t+\Delta$

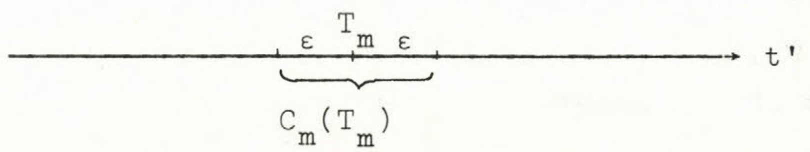
Where Δ is an additional factor to the real physical time and it is constant in the interval of our investigation $(2-3\tau)$.

Although, only time differences have role in the algorithm, for the sake of exactness $t'=t+\Delta$ instead of t will be used.

Theorem 2.

The difference between the real physical issue time (T_m) and the timestamp $C_m(T_m)$ of a transaction is at most ε.

$|C_m(T_m)-T_m|<\varepsilon$



Proof

According to condition C5, the real physical time always lies between the minimum and the maximum of local times. By C4 the differences between local times at a given moment are less than ε. Consequently, the local time at any node has a smaller difference from the physical time than ε.

Theorem 3

The differences between the local execution times of a transaction are not more than ε.

Proof

Let the transaction arrive with timestamp $C_m(T_m)$ at node i at physical time T_i and at node j at physical time T_j. The timing started at node i terminates at:

$$\tau-[C_i(T_i)-C_m(T_m)]+T_i$$

The timing started at node j terminates at:

$$\tau-[C_j(T_j)-C_m(T_m)]+T_j$$

The absolute value of their differences is:

$$|\{\tau-[C_i(T_i)-C_m(T_m)]+T_i\}-\{\tau-[C_j(T_j)-C_m(T_m)]+T_j\}|=$$

$$=|C_i(T_i)-C_j(T_j)+T_j-T_i|=|C_i(T_i)-C_j(T_i+\Delta t)+T_i+\Delta t-T_i|=$$

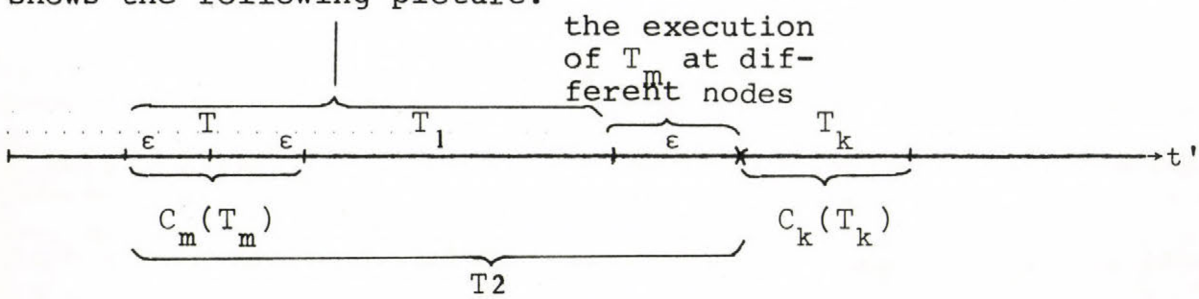
$$=|C_i(T_i)-C_j(T_i)-C_j(\Delta t)+\not{T}_i+\Delta t-\not{T}_i|<\varepsilon$$

because $|C_i(T_i)-C_j(T_i)|<\varepsilon$ and

for the time interval Δt the local clocks may be regarded as fully synchronized, therefore $C_j(\Delta t)=\Delta t$

q.e.d.

Using the theorems, the time axis representation of a log shows the following picture:



The time represented by T1 must be long enough to ensure, for every node, the reception of the transaction of the transaction of T_m and the reception of all conflicting transactions timestamped earlier than T_m. If value $\tau+\varepsilon$ is assigned to T1, then the time between the physical issue time (T_m) and the first local execution is at least τ.

In the interval succeeding T1, the database may be inconsistent. Because of this possible inconsistency, the execution of transactions conflicting with the tranaction of T_m is not permitted in this interval. Consequently, the value of T2 should be at least $\tau+2\varepsilon$.

Summing up the modifications to be done in case of real clocks:

a/ The timing should be changed from τ to $\tau+\varepsilon$.

b/ In timestamp comparison $\tau+2\varepsilon$ should be used instead of τ.

D. Weakening the condition of fully duplicated database

Instead of total duplication, the following constraint will be used: the issuing node has to contain the whole read--set of the transaction.

The execution of a transaction can be completed within a node, if the node satisfies the weakened condition with respect to that transaction. That is, a transaction can update its whole write-set on the basis of its read-set. The distribution, similarly to fully duplicated databases, serves only for introducing updates into the local databases. The nodes execute only the part of a transaction write-set that refers to its own database. Unfortunately it is impossible to reduce the number of messages since, to have the same decision results on conflictions, every node must be informed about all messages.

6. CONCLUSIONS

The suggested algorithm has some disadvantages. These are the restricted usage area and the timing that may cause implementation difficulties. On the other hand the advantages of the algorithm are that the execution of a transaction needs minimal number of messages and the amount of auxiliary information (waiting lists or the reservation of timestamps in the database etc.) is rather small.

We have some concluding remarks related to timing. The minimal time, calculated from the paramteres of the network, does not have to be used for the value of τ. With increasing τ, the number of transactions executed in time unit decreases; but on the one hand the timing can be implemented more easily using digital methods and on the other hand the short failures do not result the exceeding of τ that stops the system and needs recovery.

Other demands of the algorithm are usually provided by modern systems. In computers the preciousness of real-time clocks is in the order of 10^{-7}, 10^{-8}. That is, it is possible to hold the clock conditions for a long time without any synchronization even for an ε that is one order less than τ. Moreover, the clocks can be synchronized by messages. To determine the preciousness of synchronization the results of L. Lamport [3] can be used.

ACKNOWLEDGEMENTS

The author would like to thank L. Máté his continuous help during editing of this paper.

R E F E R E N C E S

1. P.A. Bernstein, J.B.Rothnie, N.Gooedman, C.A.Papadimitriou; The Concurrency Control Mechanism of SDD-1: A System for Distributed Databases (the Fully Redundant Case) IEEE Transactions on Software Engineering may 1978 154-167

2. P.A. Bernstein, D.W.: Shipman, J.B. Rothnie; Concurrency Control in a System for Distributed Databases (SDD-1) ACM Transactions on Database Systems march 1980 18-51

3. L.Lamport; Time, Clocks and the Ordering of Events in a Distributed System CACM july 1978 558-565

4. G.Ricart, A.K. Agrawala; An Optimal Algorithm for Mutual Exclusion in Computer Networks CACM january 1981 9-17

5. E. Chang; n-Philosophers: an Exercise in Distributed Control Computer Networks sept 1980 71-76

6. D.I. Rosenkrantz, R.E. Stearns, P.M. Lewis II; System Level Concurrency Control for Distributed Database Systems ACM transactions on Database Systems july 1978 178-198

7. J.B. Rothnie, P.A. Bernstein, et.al.; Introduction to a System for Distributed Databases (SDD-1) ACM Transactions on Database Systems march 1980 1-17

8. R.H. Thomas; A Majority Consensus Approach to Concurrency Control for Multiple Copy Databases ACM Transactions on Database Systems june 1979 180-209

Összefoglalás

EGY ÚJ IDŐZITÉSEN ALAPULÓ ALGORITMUS AZ ELOSZTOTT ADATBÁZISOK KONKURRENS FELÚJÍTÁSAINAK VEZÉRLÉSÉRE

A dolgozat röviden ismerteti a duplikált elosztott adatbázisokon alkalmazott konkurrencia vezérlő algoritmusokat, majd megad egy új eljárást, amely a felújítások szinkronizálására üzenetek helyett időzízést alkalmaz. A módszer optimális az egy felújításhoz tartozó üzenetek számára nézve.

Об одном новом алгоритме в распределенных обработках данных

В настоящей работе занимаемся распределенной обработкой данных. Даем новый алгоритм для синхронизации конкретных процессов. Наш алгоритм оптимально работает относительно мощности пакетов, которые необходимы для перепосылки информации.