# A METHOD FOR CONCURRENCY CONTROL IN DISTRIBUTED DBMSs:

# PERMISSION TEST METHOD

UGUR HALICI and ASUMAN DOGAC
Dept. of Electrical and Electronics Engineering, Middle East Tech. Univ. 06531, Ankara, TURKEY
Dept. of Computer Engineering, Middle East Tech. Univ. 06531, Ankara, TURKEY

Abstract:In this paper, a method for concurrency control in distributed DBMSs, called Permission Test Method is proposed. The PT method satisfies the basic requirements for concurrency control, that is, it executes the transactions in a serializable order, deadlocks do not appear and indefinite postponment is prevented by the method. In PT method , transactions, which are permitted to run, are not aborted unless a related site failure occurs. Furthermore, the complexity analysis indicates that the algorithm will work in a reasonable amount of time. The PT method provides more concurrency than the Basic Timestamp Ordering and Two-Phase Locking techniques.

## I. INTRODUCTION

The concurrency control in distributed DBMSs is an important research problem. Several concurrency control algorithms have been proposed for distributed DBMSs, and several have been and are being implemented. Most of the concurrency control algorithms are the variations of the following basic techniques: Two-Phase Locking (2PL), Timestamp Ordering (BTO) and Serialization Graph Testing (SGT or CPSR) [2,3,4,8].

In this paper, the "Permission Test (PT) Method" is suggested. The basic idea behind the PT Method is to give a serialization order to the transactions during execution and to prevent the transactions which destroy this order from running immediately. The PT method never delays a write operation because it guarantees that the write operations do not destroy the serialization order. A transaction waits only for some of the conflicting transactions that are currently running. Once a transaction obtains a run permission it runs to completion unless a related site fails.Because of these two conditions deadlocks do not appear. Since deadlocks and abortions do not appear the communication overhead is reduced. The PT Method also avoids indefinite postponement (livelocks) as explained in Section 3.1.a.

It is assumed that the basic unit of user computation is transaction. A transaction executes in three steps: first it reads a portion of the database into a local workspace; then it performs some local computation on its workspace; and finally it writes some values into the database [1].

A set of transactions executes serially if each transaction executes its write operation before the next transaction executes its read operation.That is, the transactions are in no way interleaved. A serial execution of

transactions preserves database consistency because each individual transaction preserves database consistency. If an interleaved execution of transactions produces the same effect as a serial execution of those same transactions, then the execution is called serializable. Since a serial execution preserves consistency, a serializable execution also preserves consistency [1]. In the PT method serializability is taken as the fundamental notion of concurrecy correctness.

In PT method, one of the sites is chosen as the scheduler site. The read and write requests of the transactions, which operate on the distributed data, reaches their destination by passing through the scheduler site, and the permitted read and write operations are executed atomically by the database system. Furthermore, it is assumed that the network is reliable and provides message sequencing.

In Section 2., definitions and notations are given. Section 3 describes the general structure of the PT method, and the related algorithms and data structures are given in this section. Section 4, contains the correctness proof of the algorithm. In Section 5, the space requirement of the PT method in comparison to locking, timestamp ordering and serialization graph testing is given and the amount of concurrency provided by the PT method is explained. Finally, Section 6. contains the conclusion and weaknesses of the method.

## II. DEFINITIONS AND NOTATIONS

Serializability theory models executions by logs. A log identifies the read and write operations executed on behalf of each transaction and tells the order in which those transactions are executed [4].

Let L be a log over [To,...,Tn]. Transaction Tk reads-Vm-from Tj in L if (1) Wj(Vm) and Rk(Vm) are operations in L ; (2) Wj(Vm) preceeds (which is denoted by <) Rk(Vm) ; and (3) no Wl(Vm) falls between these operations. Two logs over [To,...,Tn] are equivalent, denoted ≡ , if they have the same read-from relationships; that is, for all j,k and Vm, Tk reads-Vm-from Tj in one log iff this condition holds in the other [4].

More formally, the read-from (RF) relation induced by a log L, RF(L) is a ternary relation on VxTxT, where T is the set all transaction names in L and V is the set of all items, defined by (Vm,Tj,Tk) ∈ RF(L) iff Tk reads-Vm-from Tj in log L. Two logs La and Lb over [To,....,Tn] are said to be equivalent, denoted by La ≡ Lb, if RF(La)=RF(Lb).

A log is serializable if it is equivalent to a serial log. In other words, the log La is called serializable, if there is a serial log Ls such that La ≡ Ls

## III. THE GENERAL STRUCTURE OF THE SYSTEM AND THE RELATED ALGORITHMS

In PT method one of the sites is chosen as the scheduler site. A scheduler, namely PT scheduler, exists in this site. Each site of distributed DBMS runs a Transaction Manager (TM) and a Data Manager (DM) . TM forwards each read and write to the scheduler. The scheduler controls the order in which order DMs processes reads and writes. The DM executes each read and

write it receives. It should be noted that the transactions operating on the local data may be scheduled by a local scheduler making use of the same principles as described below.

When a transaction, which operates on distributed data, is submitted to a TM, the read-set and the write-set of the transaction is determined by the TM and these sets are sent to the scheduler site along with the transaction identifier. The transactions arriving at the scheduler site are placed into Waiting Transaction List in the order they arrive. At the scheduler site the Permission-Test is applied on the transaction to check whether serializability will be destroyed with the immediate execution of the transaction. For this purpose Permission-Test makes use Permission Chart and the Active List, a list containing the currently active transactions. The resulting order in the Active List (without any deletion) gives the serial order of the transactions. When a transaction succeeds the test, the read-set of the transaction is sent to the related DM. It should be noted that the TM is responsible for the control of execution but the execution does not necessarily take place at the TM's site.

After the execution of the transactions, the values obtained for the write-sets are sent to all of the related DMs, and the scheduler site is informed of this transmission. The values are stored at the secure storage of the DMs and are not committed unless a commit message is received from the scheduler site. After receiving the message informing the end of transmission, the scheduler site sends the commit messages along with the portion of the write-set which remained in the Permission Chart. Only the values for this portion of the write-set are copied into the database.

If a related site fails before a transaction is comitted and the items related to the transaction are not available at any other site, then the transaction is aborted and all the related information is deleted from the system. Later the transaction is restarted. When a scheduler site fails, all the transactions which, have not executed at least one commit, are aborted. Another site is chosen as the scheduler site and the aborted transactions are restarted. The transactions that have executed at least one commit are terminated after the DMs waiting for commit obtained this information from the DM(s) where the transaction has performed its commit.

### 3.1. Data structures

The following data structures exists at the scheduler site

a) Waiting Transaction List (WTL)
b) Permission Chart (PC)
c) Active List (AL)
d) Write Queue (WQ)

#### 3.1.a. Waiting Transaction List:

All the transactions sent from the sites are placed into Waiting Transaction List in the order that they arrive at the scheduler site. The Waiting Transaction List contains the following information :
Transaction Identifier : Each transaction has a unique identifier which is obtained by concatanating the unique site identifier and the unique

transaction identifier within that site.
Read-set : The portion of the database, a transaction reads
Write-set : The portion of the database, a transaction will update
Priority : The priority of a transaction is initially zero and is increased by one each time it fails the Permission Test. Higher priority transactions appear in the begining of the list and therefore are tested first. In order to prevent indefinite postponment, when the priority of a transaction reaches a predefined limit P, the writes of running transactions are accepted but no other transaction, except for this one, is permitted to be tested for run permission. Increasing the priority limit increases the concurrency but also increases the response time for some of the transactions. Setting a priority limit avoids indefinite postponment.

3.1.b. Permission Chart :

Permission Chart is used in conjunction with the Active List to make a decision on whether to give a run permission to a transaction or not. In this chart, there is a row for each item. The following symbols may appear on a row.

Ri : this symbol indicates that the item has been read by transaction Ti
wi : this symbol indicates that transaction Ti will write this item
Wi : this symbol indicates that Ti has written a value for this item.

A row in the Permission Chart contains one of the following sequence of symbols

| | |
|---|---|
| Vm: Wi | Item Vm is lastly written by Ti and and there is no transaction that has read this value |
| Vm: Wi Rj | Ti is the last transaction that has written a value into Vm, and this value has been read at least by one transaction Tj. There could be other transactions that have read this value, but they disappear as a result of executing the Read-Squeze algorithm |
| Vm: Wi wj ... wk | The item Vm is lastly written by Ti and the transactions Tj,...,Tk will write into Vm and there is no transaction that has read the value written by Ti |
| Vm: Wi Rj wk ... wl | Item Vm is lastly written by Ti and this value has been read by at least one transaction Tj and the transactions Tk,...,Tl will write into Vm. Here Tj and Tk may be the same transaction. |

The granularity of data items could range between a tuple and a relation. The granularity of data items effect the size of the permission chart. Obviously finer granularity of data items results in larger permission chart.

3.1.c. Active List :

Active List contains the currently active transactions, that is, the transactions having at least one element in the Permission Chart. Active List together with the Permission Chart is used by the Permission Test to give run

permission to the transactions. In the mean time the Permission Test decides on the order of the transactions in the Active List. If we assume a virtual list called Transaction Order List, which is obtained from the Active List by ignoring the deletions, Transaction Order List will give a serial ordering of the transactions. The first transaction in the Transaction Order List will be the first transaction in serial execution and the last transaction in the Transaction Order List will be the last one in the serial execution.

Active List contains the following information :
Transaction Identifier: is the same as it is in the Waiting Transaction List
Number of Items (num): num(i) gives the number of items that are in the Permission Chart related with transaction Ti. This number is initially set to total number of items in the read-set plus number of items in the write-set of Ti and decreased as a result of chart squezing. When num(i) reaches to zero, transaction Ti is deleted from the Active List.
Relation (rel) : rel(i) is used for the test of the transactions for run permission and for their insertion to the Active List. During the Permission-Test, the value of rel(i) indicates the relative order of transaction Ti with respect to Tj. It may have one of the following three values :
">" (greater), "<" (less) , "-" (don't care)

3.1.d. Write Queue :

After the execution of the transactions by the related TMs the values obtained for the write-sets are sent to the related DMs and schecular site is informed of this transmission.

Write Queue contains the following information :
Transaction Identifier: The same as it is in the Waiting Transaction List
Write-Set: The write-set of the transaction

3.2. THE PERMISSION PROCEDURE AND THE ALGORITHMS

The operations performed by the PT scheduler is given in the following algorithm.

```
Algorithm SCHEDULER
(Let K denote the number of elements in the Waiting Transaction List)
n=0
do
   if write queue is not empty then do
       call WRITE-INSERT,
       n=0,
   end,
   if priority(1) > P then n=1
   else do
       n=n+1,
       if n=K+1 then n=1
   end,
   call PERMISSION-TEST for Tn
   if test succeds then do
      delete Tn from the Waiting Transaction List,
      send read-set to related DMs,
```

```
      call INSERT,
      n=n-1,K=K-1
   else
      priority(n)=priority(n)+1
   end
forever
     The Algorithms called by the PT scheduler are the following

Algorithm PERMISSION-TEST
(the transaction Tn is tested for run permission)

1. fail=0
   (Let N denote the number of elements in Active List)
   do i=1 to N, rel(i)="-", end

2. For each item Vm in the read-set of transaction Tn
   if the row in the Permission Chart corresponding to the item is
   2.1. Vm: Wi   then
                 if the relation element in the Active List rel(i)
                 is ">" then do fail=1, return end
                 else set rel(i) to "<"
   2.2. Vm: Wi Rj    then
                 if rel(i) is ">" then do fail=1, return end
                 else set rel(i) to "<"
   2.3. Vm: Wi Rj wk ... wl    then
                 if rel(i) is ">" or rel(k) is "<" then do fail=1, return end
                 else set rel(i) to "<"  and rel(k) to ">"
   2.4. Vm: Wi wk ... wl    then
                 if rel(i) is ">" or rel(k) is "<" then do fail=1, return end
                 else set rel(i) to "<"  and rel(k) to ">"
3. for each item Vm in the write-set of transaction Tn
   if the row in the Permission Chart corresponding to the item is
   3.1. Vm: Wi   then
                 if rel(i) is ">" then do fail=1, return end
                 else set rel(i) to "<"
   3.2. Vm: Wi Rj    then
                 if rel(j) is ">" then do fail=1, return end
                 else set rel(j) to "<"
   3.3. Vm: Wi Rj wk ... wl    then
                 if rel(j) is ">" then do fail=1, return end
                 else set rel(j) to "<"
   3.4. Vm: Wi wk ... wl    then
                 if rel(i) is ">" then do fail=1, return end
                 else set rel(i) to "<"
4. (test of  Tn in the Active List)
   4.1. change=0, place=0,i=1
       do while  i<=N
          if rel(i)="<" and change=1 then do fail=1, return end
                                     else place=i,
          if rel(i)=">" and change=0 then change=1,
          if rel(i)="-" and change=0 then place=i,
          i=i+1
        end
```

```
5. (Insertion of Tn into the Active List)
    5.1. adjust the links such that the transaction Tn is inserted in between
         place and place+1,
         num(place+1) = number of elements in the read-set
                      + number of elements in the write-set



Algorithm INSERT
(Inserting the transaction into the Permission Chart)
(Let Ei to denote the i'th element of row Vm of Permission Chart)
(Let Ek be the last element of row Vm)
1. (Inserting reads of a transaction Tn into the Permission Chart)
   for each Vm of read-set of Tn do
       i=1 ,
       if E2 is an read element then i=2,
       insert Rn  between Ei and E(i+1),
       call READ-SQUEZE
   end
2. (Inserting writes of a transaction Tn into the Permission Chart)
   for each Vm of Write-set of Tn do
      i=2,
      found=0
      if i>k then found=1 else if Ei is a read element then i=3
      do while found=0 and i<=k
         if  in  the Active List Tn comes before Tj,  which is related to  Ei,
             then found=1
         i=i+1
      end
      insert wn in between E(i-1) and Ei
   end


Algorithm READ-SQUEZE
       if there is RjRn in the row Vm for any j then do
          convert RjRn to Rn in the row Vm of Permission Chart,
          num(j)=num(j)-1 in the Active List,
          if num(i)=0 then delete Ti from the Active List
       end


Algorithm WRITE-INSERT
for each Tp in the Write Queue do
    send  the  portion  of the write set of  Tp,  which  remains  in  the
    Permission Chart to all related DMs,
    for each item Vm in the write-set of Tp do
        convert wp to Wp in row Vm of Permission chart,
        call WRITE-SQUEZE
    end,
    delete Tp from the Write Queue
end
```

```
Algorithm WRITE-SQUEZE
        Repeat  until there is no wk for any k to the left of Wp
            convert   Vm: ... wk Wp ...   to  Vm: ... Wp ...   ,
            num(k)=num(k)-1,
            if num(k)=0 then remove Tk from the Active List
        end
        If the row for the item Vp in the Permission chart is
        1.  Vm: Wj Wp ...   then do
                convert Vm: Wj Wp ...   to    Vm: Wp ...  ,
                num(j)=num(j)-1,
                if num(j)=0 then remove Tj from the Active List
            end
        2.  Vm: Wj Rk Wp ...   then do
                convert Vm: Wj Rk Wp ...   to    Vm: Wp ...  ,
                num(j)=num(j)-1,
                if num(j)=0 then remove Tj from the Active List,
                num(k)=num(k)-1,
                if num(k)=0 then remove Tk from the Active List
             end
```

### 3.3. Complexity of the algorithms

Assuming that v is the number of the items in the set V (which is the set of all items read and written by the transactions) and N is the number of elements in the Active List, the complexity of the algorithms realizing the PT method is O(Nv).

### 3.4. The PT Method Through an Example

In the following the PT method will be explained through an example.

Before we proceed any further, we will give some more notation:

Tj[Vr/Vw] : demonstrates that transaction Tj's read-set is Vr and write-set is Vw.

La : is the output of the PT scheduler and gives the actual execution sequence.

al : shows the current order in the Active List

tol : shows the order in the Transaction Order List. The order of the transactions in tol gives the serial order of transactions for Ls which is equivalent to La.

Let the input log, the sequence of arriving user requests, of the PT scheduler be

L=R1[x] R2[y] R3[y] R4 W4[y] W2[z] W1[y,z] W3[x].

Obviously, the input log is not a serializable one. In the following, how the PT scheduler handles the input log is explained:

Assume that items x, y and z are written by an initial transaction Ti[0/x,y,z] and will be finally read by a transaction Tf[x,y,z/0].The initial state of the Permission Chart (PC State) is

```
#               x: Wi
                y: Wi
                z: Wi
     La= Ri Wi[x,y,z]
     al= Ti and tol= Ti
* T1[x/y,z] arrives and R1[x] is served
     PC state   x: Wi R1
                y: Wi w1
                z: Wi w1
     La= Ri Wi[x,y,z] R1[x]
     al= Ti T1 and tol= Ti T1
* T2[y/z] arrives and R2[y] is served
     PC state   x: Wi R1
                y: Wi R2 w1
                z: Wi w2 w1
     La= Ri Wi[x,y,z] R1[x] R2[y]
     al= Ti T2 T1 and tol= Ti T2 T1
* T3[y/x] arrives and R3[y] is not permitted since
                x: Wi R1      --> T1 < T3  i.e. rel(1) = "<"
                y: Wi R2 w1   --> T1 > T3  i.e  rel(1) = ">"
  so T3 remains in Waiting Transaction List.
* T4[0/y] arrives and R4 is served
     PC state   x: Wi R1
                y: Wi R2 w1 w4
                z: Wi w2 w1
     La= Ri Wi[x,y,z] R1[x] R2[y] R4
     al= Ti T2 T1 T4 and tol= Ti T2 T1 T4
* W4[y] arrives and it is served
     PC state   x: Wi R1
                y: W4
                z: Wi w2 w1
     La= Ri Wi[x,y,z] R1[x] R2[y] R4 W4[y]
     al= Ti T2 T1 T4 and tol= Ti T2 T1 T4
* Now R3[y] is permitted.
     PC state   x: Wi R1 w3
                y: W4 R3
                z: Wi w2 w1
     La= Ri Wi[x,y,z] R1[x] R2[y] R4 W4[y] R3[y]
     al= Ti T2 T1 T4 T3 and tol= Ti T2 T1 T4 T3
* W2[z] arrives and it is served
     PC state   x: Wi R1 w3
                y: W4 R3
                z: W2 w1
     La= Ri Wi[x,y,z] R1[x] R2[y] R4 W4[y] R3[y] W2[z]
     al= Ti T2 T1 T4 T3 and tol= Ti T2 T1 T4 T3
*  W1[y,z] arives and it  is served as W1[y,z] .  Since there is no w1 in  the
row  of y,  write operation on item y is ignored and this fact is demonstrated
as W1[y,z].
     PC state   x: Wi R1 w3
                y: W4 R3
                z: W1
     La= Ri Wi[x,y,z] R1[x] R2[y] R4 W4[y] R3[y] W2[z] W1[y,z]
     al= Ti T1 T4 T3 and tol= Ti T2 T1 T4 T3
```

```
* W3[x] arrives and it is served
     PC state  x: W3
               y: W4 R3
               z: W1
     La= Ri Wi[x,y,z] R1[x] R2[y] R4 W4[y] R3[y] W2[z] W1[y,z] W3[x]
     al= T1 T4 T3 and tol= Ti T2 T1 T4 T3
* Assuming that Tf is the final transaction that reads all the items then
     PC state  x: W3 Rf
               y: W4 R3 Rf
               z: W1 Rf
     La= Ri Wi[x,y,z] R1[x] R2[y] R4 W4[y] R3[y] W2[z] W1[y,z] W3[x] Rf[x,y,z]
         Wf
     al= T1 T4 T3 Tf and tol= Ti T2 T1 T4 T3 Tf
```

La is equivalent to the serial log Ls which is obtained by executing the transactions in the order that they appear in tol ,that is

```
Ls=Ri Wi[x,y,z] R2[y] W2[z] R1[x] W1[y,z] R4 W4[y] R3[y] W3[x] Rf[x,y,z] Wf
```

## IV. CORRECTNESS PROOF OF THE PERMISSION TEST METHOD

Let La be any log generated by the PT method. Assume the initial values for all items in V, which is the set of all items, are written by a virtual transaction Ti whose read-set is empty and write set is V, and also assume that there is a transaction Tf, which finally reads all the items in V, that is, the read set of Tf is V and the write-set of Tf is empty. Assume that there is a list, called Transaction Order List (TOL), which is obtained from the Active List without any deletion. Hence, Ti is the first and Tf is the last element in the TOL. Let Ls be the serial log obtained by executing the transactions in the order they appear in the TOL.

Lemma 1. If (Vm,Tj,Tk) $\in$ RF(Ls) then (Vm,Tj,Tk) $\in$ RF(La)

Proof: At first we will show that If (Vm,Tj,Tk) $\in$ RF(Ls) then Vm: Wj Rk ... appears in row of item Vm of the Permission Chart.

Assume that it does not appear. That is , Vm: Wl Rk ... such that l=j appears in the Permission Chart. This implies that Vm $\in$ Wl and Vm $\in$ Rk and Tl $<$ Tk in TOL. Since, Vm: ... wj Rk ... is not permitted by the PT method, there can not be a Tj such that Vm $\in$ Wj and Tl $<$ Tj $<$ Tk in TOL. If Vm $\notin$ Wj then (Vm,Tj,Tk) $\notin$ RF(Ls) obviously, which is a contradiction.Therefore if Vm $\in$ Wj then possible orders for Tj, Tk and Tl in TOL is either Tj $<$ Tl $<$ Tk or Tl $<$ Tk $<$ Tj. If Tj $<$ Tl $<$ Tk is the order of the transactions in TOL then this implies that Wj $<$ Wl $<$ Rk in the serial log and this in turn implies that (Vm,Tj,Tk) $\notin$ RF(Ls). This is a contradiction. If Tl $<$ Tk $<$ Tj is the order in TOL then this implies that Rk $<$ Wj in the serial log which in turn implies that (Vm,Tj,Tk) $\notin$ RF(Ls) . Again contradiction.

At this point, it is shown that if (Vm,Tj,Tk) $\in$ RF(Ls) then Vm: Wj Rk... appears in the Permission Chart for the item Vm. Since Vm: ... wl Rk ... is not allowed by the PT method, the appearance of Vm: Wj Rk ... in the Permission Chart implies that Vm $\in$ Wj and Vm $\in$ Rk and Wj $<$ Rk and also implies

that there is no Wl such that Vm ∈ Wl and Wj < Wl < Rk in La. That is, (Vm,Tj,Tk) ∈ RF(La). □

Lemma 2. If (Vm,Tj,Tk) ∈ RF(La) then (Vm,Tj,Tk) ∈ RF(Ls)

Proof: If (Vm,Tj,Tk) ∈ RF(La) then Vm ∈ Wj and Vm ∈ Rk and Wj < Rk in La and there is no Vm ∈ Wl such that Wj < Wl < Rk, that is, Vm: Wj Rk ... appears in the Permission Chart. This implies that, Tj < Tk in TOL and there is no Tl such that Vm ∈ Wl and Tj < Tl < Tk in TOL. Since the serial log Ls is created by using TOL, Wj < Rk and there is no Wj such that Vm ∈ Wj and Wj < Wl < Rk in Ls. That is, (Vm,Tj,Tk) ∈ RF(Ls). □

Lemma 3. RF(La)=RF(Ls)

Proof : Obtained by using Lemma 1. and Lemma 2. □

Theorem PT : Any log La generated by the Permission Test method is serializable.

Proof: By Lemma 4., RF(La)=RF(Ls). Then by the definition of log equivalence La ≡ Ls. Since Ls is a serial log, by definition of serializability, La is serializable. □

## V. THE SPACE REQUIREMENTS AND THE AMOUNT OF CONCURRENCY PROVIDED BY THE PERMISSION TEST METHOD

The PT method is the linearization of the Serialization Graph Testing technique, in the sense that, the graph in SGT the technique is represented through lists in the PT method. The partial order preseved in the Serialization Graph is converted to a total order in the PT method. For this reason PT method provides less concurrency than SGT method. However the SGT method stores the read and the write sets of all active transactions along with the serialization graph. Therefore the amount of data to be stored in the PT method is far less than that of the SGT method.

In Basic Timestamp Ordering (BTO) technique the serialization order is selected a priori and the execution of the conflicting transactions is forced to obey this total order [4]. However, in PT method there is no priori serialization order; the transaction order is decided during execution. It is clear that this increases the amount of concurrency. In the BTO technique, the scheduler maintains for every data item x, the maximum timestamp of Reads and writes on x [4]. In the PT method, the scheduler maintains for every data item x, the transaction identifier of the transaction that has the highest order in the active list and that has performed a read and/or write operation on x. This correspons to the timestamps on data items and requires the same amount of space. In the PT method, the information about the future writes (wi in the permission chart) is also maintained, however this information is preserved for a short duration of time.

In Two-Phase Locking technique, when there is a write-lock on an item, no other transaction is allowed to have any lock on this item even if such a request would not result in any inconsistency. However, in PT method such requests can be accepted and this results in more concurrency. In 2PL

implementations, a lock table is maintained which contains the identifier of the transaction requesting the lock, the name of the data item to be locked and the mode of the lock, "read" or "write". A queue of waiting lock requests is also maintained. Obviously, the amount of information maintained in a lock table is more than the information maintained in the permission chart. However, the lock table maintains this information only for currently executing transactions. In the PT method, the rows of the permission chart which have not been used for a while can be discarded. Then, when a transaction requires a data item that does not exist in the permission chart, it is assumed that this data item has been read and written by the highest order transaction in the active list.

Consider the following log L which can be the output of the SGT scheduler but which can not be produced by the BTO or 2PL schedulers.

L= R3[x] R1 W1[x] R2[y] W2 W3[y] R4[x] R5 W5[x,y] W4[z] R6 W6[y,z]

L is the log h10 in [7]. In the following, how the PT scheduler handles the input log L is explained:

```
* The initial state of the Permission Chart (PC state) is
                    x: Wi
                    y: Wi
                    z: Wi
     La= Ri Wi[x,y,z]
     al= Ti   and  tol= Ti
* T3[x/y] arrives and R3[x] is served:
     PC state      x: Wi R3
                   y: Wi w3
                   z: Wi
     La= Ri Wi[x,y,z] R3[x]
     al= Ti T3    and   tol= Ti T3
* T1[0/x] arrives and R1 is served:
     PC state      x: Wi R3 w1
                   y: Wi w3
                   z: Wi
     La= Ri Wi[x,y,z] R3[x] R1
     al= Ti T3 T1    and   tol= Ti T3 T1
* W1[x] arrives and it is served:
     PC state      x: W1
                   y: Wi w3
                   z: Wi
     La= Ri Wi[x,y,z] R3[x] R1 W1[x]
     al= Ti T3 T1    and   tol= Ti T3 T1
* T2[y/0] arrives and R2[y] is served:
     PC state      x: W1
                   y: Wi R2 w3
                   z: Wi
     La= Ri Wi[x,y,z] R3[x] R1 W1[x] R2[y]
     al= Ti T2 T3 T1    and   tol= Ti T2 T3 T1
* W2 arrives and it is served:
     PC state      x: W1
                   y: Wi R2 w3
                   z: Wi
```

```
     La= Ri Wi[x,y,z] R3[x] R1 W1[x] R2[y] W2
     al= Ti T2 T3 T1    and   tol= Ti T2 T3 T1
* W3[y] arrives and it is served:
     PC state      x: W1
                   y: W3
                   z: Wi
     La= Ri Wi[x,y,z] R3[x] R1 W1[x] R2[y] W2 W3[y]
     al= Ti T3 T1    and   tol= Ti T2 T3 T1
* T4[x/z] arrives and R4[x]  is served:
     PC state      x: W1 R4
                   y: W3
                   z: Wi w4
     La= Ri Wi[x,y] R3[x] R1 W1[x] R2[y] W2 W3[y] R4[x]
     al= Ti T3 T1 T4    and   tol= Ti T2 T3 T1 T4
* T5[0/x,y] arrives and R5 is served:
     PC state      x: W1 R4 w5
                   y: W3 w5
                   z: Wi w4
     La= Ri Wi[x,y] R3[x] R1 W1[x] R2[y] W2 W3[y] R4[x] R5
     al= Ti T3 T1 T4 T5   and   tol= Ti T2 T3 T1 T4 T5
* W5[x,y] arrives and it is served:
     PC state      x: W5
                   y: W5
                   z: Wi w4
     La= Ri Wi[x,y] R3[x] R1 W1[x] R2[y] W2 W3[y] R4[x] R5 W5[x,y]
     al= Ti T4 T5   and   tol= Ti T2 T3 T1 T4 T5
* W4[z] arrives and it is served:
     PC state      x: W5
                   y: W5
                   z: W4
     La= Ri Wi[x,y] R3[x] R1 W1[x] R2[y] W2 W3[y] R4[x] R5 W5[x,y] W4[z]
     al= T4 T5   and   tol= Ti T2 T3 T1 T4 T5
* T6[0/y,z] arrives and R6 is served:
     PC state      x: W5
                   y: W5 w6
                   z: W4 w6
     La= Ri Wi[x,y] R3[x] R1 W1[x] R2[y] W2 W3[y] R4[x] R5 W5[x,y] W4[z] R6
     al= T4 T5 T6   and   tol= Ti T2 T3 T1 T4 T5 T6
* W6[y,z] arrives and it is served:
     PC state      x: W5
                   y: W6
                   z: W6
     La= Ri Wi[x,y] R3[x] R1 W1[x] R2[y] W2 W3[y] R4[x] R5 W5[x,y] W4[z] R6
         W6[y,z]
     al= T5 T6   and   tol= Ti T2 T3 T1 T4 T5 T6

* Assuming that Tf is the final transaction that reads all the items then:
     PC state      x: W5 Rf
                   y: W6 Rf
                   z: W6 Rf
     La= Ri Wi[x,y] R3[x] R1 W1[x] R2[y] W2 W3[y] R4[x] R5 W5[x,y] W4[z] R6
         W6[y,z] Rf[x,y,z] Wf
     al= T5 T6 Tf  and   tol= Ti T2 T3 T1 T4 T5 T6 Tf
```

La is equivalent to the serial log Ls, which is obtained by executing the transactions in the order that they appear in tol, that is

Ls=Ri Wi[x,y,z] R2[y] W2 R3[x] W3[y] R1 W1[x] R4[x] W4[z] R5 W5[x,y] R6 W6[y,z] Rf[x,y,z] Wf

## VI. DISCUSSIONS

The method presented in this paper, introduces some basic principles for dynamic ordering of transactions for concurrency control in distributed databases. In the PT method the deadlocks do not appear and the livelocks are prevented by the method. Transactions are not aborted unless a related site failure occurs. The method provides less concurrency than SGT method but more concurrency than BTO and 2PL methods.

A weakness of the PT method is that the scheduler is centralized, that is one of the sites is chosen as a scheduler site. In the case of a scheduler site failure, all running transactions must be aborted and furthermore, another site must be chosen as the scheduler site. This problem arises from the fact that, the permission chart and the active list can not be distributed efficiently. Distributed permission test will result in higher communication cost as in the case of distributed SGT method. Instead of keeping an active list, serialization numbers can be given to the transactions such that the order of these numbers correspond to the order in the active list [5]. Once the need for an active list is eliminated, the permission chart can be distributed by using the serialization numbers in the permission chart instead of the transaction identifiers. In [6] such an algorithm is presented, based on the principles given in this paper, however, read and write sets are not predeclared. It should be noted that in such a case abortions become unavoidable.

## REFERENCES


[1] P. A. Bernstein, D.W. Shipman and W.S. Wong, "Formal Aspects of Serializability in Database Concurrency Control", IEEE Trans Software Eng., vol. SE-5, pp. 203-215, May 1979.

[2] P.A. Bernstein and N. Goodman, "Fundamental Algorithms for Concurrency Control in Distributed Database Systems", Technical Report CCA-80-05, Feb. 1980.

[3] P. A. Bernstein and N. Goodman, "Concurrency Control in Distributed Systems", ACM Comput. Surveys, vol. 13, no. 2, pp. 185-222, June 1981.

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

[5] A. Dogac and U. Halici, "Timestamp Transformation Method for Concurrency Control in Distributed DBMSs", in the Proc. of the first International Symposium on Computer and Information Sciences, Ankara, Turkey, Oct. 1986

[6] U. Halici and A. Dogac, "Concurrency Control in Distributed Databases through Time Intervals and Short Term Locks", Technical Report, Dept. of Computer Engineering, METU, August 1986 (Submitted for publication).

[7] C. H. Papadimitriou, "The Serializability of Concurrent Database Updates", J. Ass. Comput. March., Vol. 26, pp. 631-653, Oct. 1979.

[8] C. H. Papadimitriou, Theory of Database Concurrency Control, Computer Science Press, 1986.