アルゴリズム 12－38
(1989. 11. 22)

An On-Line Consistency Preservation Algorithm for Distributed Database Systems with Broadcast Communication Function

Jun Okui*, and **Mamoru Fujii****
* Faculty of Engineering, Nagoya Institute of Technology
** College of General Education, Osaka University

Abstract

An on-line scheduler for consistency preservation of distributed database systems with broadcast communication function is proposed. Each local subdatabase system (SDBS) has an identical scheduler. These schedulers manage only their respective local data items. Every transaction is executed by one SDBS. A scheduler communicates with other SDBS's only if data items of another SDBS are read (or written) in the transaction. Our goal is a SDBS which can act both as a local system and as a total system. In other words, a system wherein each SDBS shall work as a stand-alone database system while preserving only local information in itself.

We assume that there is no copy of the data items. We further assume the scheduler can know the sets of the data items in each transaction t which are read and written when the transaction t is started. The maximum number of communications to other SDBS's for each read (or write) operation is (SDBS's + 1). Our scheduling algorithm accepts the class of the read-write histories which includes properly the class of the CPSR.

1 Introduction

Local area networks (LAN's) or satellite communication networks which have broadcast communication functions are widely used nowadays. We propose an on-line schedule algorithm (hereinafter DSA) which utilizes the broadcast communication function for consistency preservation (through the preservation of serializability) of distributed database systems.

The problem of preserving consistency (through the preservation of serializability) is generally a NP-complete problem[1]. Many polynomial time algorithms which accept subclasses of serializable histories (the sequences of read or write commands executed by the scheduler) are known[1,2,3] (For example, CPSR proposed by P.A.Bernstein, etc.). For distributed database systems, SDD1[5] applies the time-stamp. Our algorithm DSA accepts the class of histories which include properly the class of the CPSR and also the class accepted by SDD1.

The distributed database system (hereinafter DDBS) is composed of multiple local subdatabase systems (hereinafter SDBS's) and one broadcast communication line. Our goal is a SDBS which can act both as a local system and as a total system. In other words, a system wherein each SDBS shall work as a stand-alone database system while preserving local information in itself. Each SDBS has an identical scheduler which manages its local data items. The only cost criterion is the number of broadcasts.

Database D is a set of data items of the DDBS. For our purposes, the granularity of the data items is unimportant. The set of data items D(s) is a subset of D which is managed by a SDBS "s". A transaction is a program which is managed by an SDBS.

We define transaction t as an acyclic flowchart of read(or write) commands. SDBS s(t) is a sub-database system which executes transaction t. The two subsets of set D in transaction t are defined as follows:

The readset R(t) is a set of data items which are read by transaction t, and the writeset W(t) is a set of data items which are written by transaction t. The scheduler of a SDBS s(t) can know the readset R(t) and the writeset W(t) when execution of transaction t is started. There are two types of transactions. If the readset R(t) and the writeset W(t) of transaction t are both included [D(s(t)): the set of data items in SDBS s(t)], then transaction t is a L-transaction. In any other case, transaction t is a M-transaction.

The following are assumptions about the communication model.

(1) Only one message is transmitted at a time.

(2) Each message is received by all SDBS's simultaneously in the order of the transaction.

(3) There is no communication error.

(4) Each scheduler S of the SDBS has a number SI(S) and is totally ordered by the number.

If a read (or write) command of a transaction t reads (or writes) data items other than D(s(t)), then our algorithm DSA communicates with other SDBS's $(n+1)$ times to decide the acceptance of the read (or the write) command, where n is the total number of SDBS's in the DDBS. If a read (or write) command of a transaction t reads (or writes) data items in the D(s(t)) then scheduler S of the SDBS s(t) broadcasts a message only once. Further more, it is shown that our algorithm DSA is deadlock-free.

2 Terminology

Let $D = \{x_1, \ldots, x_m\}$ be the set of all data items.

The domain of data item x_i is $dom(x_i)$, $i = 1, \ldots, m$; a database state is an element of $dom(D) = dom(x_1) \times \ldots \times dom(x_m)$. We shall define the initial database state IS as consistent. Database state transition occurs by transactions. The set CD is the consistent database state which contains all of the database states which will be translated by any serial execution of transactions starting from the state IS.

Let $G =< N, E >$ be a directed acyclic graph. We can define a partial order $<<_G$ naturally on the set N. Let n_1 and n_2 be different nodes of graph G. We say that n_2 is *reachable* from n_1 in graph G iff there is a directed path from n_1 to n_2 in graph G. This shall be denoted $n_1 <<_G n_2$. Also, we say that a set of nodes $R(n_1)$ is *reachable* from the node n_1 in the graph G iff

$R(n_1) = \{n_i | n_i$ is a node of graph G and $n_1 <<_G n_i\}$

For simplicity, we consider that each transaction is a sequence of *commands* requested by a directed acyclic graph. Let $T(t) =< N(t), E(t) >$ be a directed acyclic graph for transaction t. Where

$N(t) = \{C_i | C_i$ is a read or write command of the transaction $t_i\}$

$E(t) = \{(C_i, C_j)|$ command C_j will be executed in t following the command $C_i\}$.

One start node and some end nodes are specified in T(t). The indegree of the start node is zero and the outdegree of each end node is zero. Each command specifies also the set of data items $d(C_i)$ which are the actual parameters of command C_i. And we denote $C_i(x, y)$ as $d(C_i) = \{x, y\}$. We represent "$C_i <_t C_j$" to denote "there is a directed edge (C_i, C_j) in the set of edge E(t)". We say that a string $S_t = C_1 \cdot C_2 \cdot \ldots \cdot C_n$ of commands in N(t) is an *execution sequence* of t iff

(1) command C_1 is the start node of T(t)

(2) command C_n is an end node of T(t)

(3) for any $C_i, C_{i+1} (1 \leq i \leq n-1)$, $C_i <_t C_{i+1}$.

Let $d(C_i)$ and $d(C_j)$ be the set of data items which are the actual parameters of commands C_i and C_j respectively. We say that write commands C_i and C_j are in *ww-conflict* iff $d(C_i) \cap d(C_j) \neq \phi$. And, we say that read command C_i and write command C_j are in *rw-conflict* iff $d(C_i) \cap d(C_j) \neq \phi$

Let ST be a set of transactions and let L be the union of the set of commands which are in N(t) for each transaction t. A *history* $H = C_1 \cdot C_2 \cdot \ldots \cdot C_n$ is a string of commands from the set L such that:

For each transaction t in the ST, the projection of the string H to the set N(t) of the commands is an execution sequence of t.

Any prefix of history H is a *subhistory* of H. We write "$C_1 <_H C_2$" to denote;

" command C_1 *proceeds* command C_2 in history (or subhistory) H".

For example, if $H = C_1 \cdot C_2 \cdot \ldots \cdot C_n$ then $C_i <_H C_j$ iff $i < j$.

Let H_s be a subhistory. We say that transaction t is an *ended* transaction if the projection of the subhistory H_s to the set of the commands N(t) is an execution sequence of t, and we say that transaction t is an *active* transaction if the projection of the subhistory H_s to the set of the commands N(t) is not an execution sequence of t.

Let $E_t = FH_t \cdot PH_t$ be an execution sequence of the transaction t_i and let H_s be a subhistory. We say that PH_t is a *rest execution sequence* of the active transaction t with subhistory H_s iff every command of FH_t appears in the subhistory H_s and no commands of PH_t appear in the subhistory H_s.

We say that a string of commands H is a *candidate history* with a subhistory H_s if H_s is a prefix of H. The set of all candidate histories with the subhistory H_s is denoted $PH(H_s)$ and, we define a subset $PSH(H_s)$ of $PH(H_s)$ as follows:

Let PH_{e_i} be a rest execution sequence of the transaction t_{e_i} with the subhistory H_s,

$PSH(H_s) = \{H | H = H_s \cdot PH_{e_1} \cdot PH_{e_2} \cdot \ldots \cdot PH_{e_q}\}$, where q is the number of active transactions in H_s.

We define the *rest of the readset* and *rest of the writeset* for active transactions in the subhistory H_s, respectively denoted $RER(t, H_s)$ and $REW(t, H_s)$ to be the set of data items where

$RER(t, H_s) = \{d_i | d_i$ is an element of the set $d(C_i)$, where C_i is a read command of an active transaction t in the subhistory

H_s and the read command C_i is an element of a rest execution sequence of the transaction t with the subhistory H_s}

$REW(t,H_s) = \{d_i | d_i$ is an element of the set $d(C_i)$, where command C_i is a write command of an active transaction t in the subhistory H_s and the write command C_i is an element of a rest execution sequence of the transaction t with the subhistory $H_s\}$

We define that $RER(t,H_s) = REW(t,H_s) = \phi$ for an ended transaction t.

Let a read command C_i of a transaction t_1 and a write command C_j of a transaction t_2 appear in a subhistory H_s. We say that read command C_i and $REW(t,H_s)$ are in *rw-conflict* in the subhistory H_s iff $d(C_i) \cap REW(t,H_s) \neq \phi$ and $t_1 \neq t$.

We say that write command C_j and $RER(t,H_s)$ are in *rw-conflict* in the subhistory H_s iff

$d(C_j) \cap RER(t,H_s) \neq \phi$ and $t_2 \neq t$.

And, we say that write command C_j and $REW(t,H_s)$ are in *ww-conflict* in the subhistory H_s iff $d(C_j) \cap REW(t,H_s) \neq \phi$ and $t_2 \neq t$.

Let write commands C_i and C_j appear in a subhistory H_s such that $C_i <_{H_s} C_j$ and let there be no read command C_r in the subhistory H_s such that $C_i <_{H_s} C_r$ and $d(C_i) \cap d(C_j) \cap d(C_r) \neq \phi$. We say that write commands C_i and C_j are *ww-changeable* iff

We can make a set of data items CW which covers $d(C_i) \cap d(C_j)$ and satisfies the following condition;

Condition: For any data item d_m in the set CW, there exists a write command C_w such that:

(1) The data item d_m is an element of $d(C_w)$.

(2) $C_j <_{H_s} C_w$.

(3) There is no read command C_r such that the data item d_m is an element of $d(C_r)$ and $C_j <_{H_s} C_r <_{H_s} C_w$.

Intuitively, if write commands C_i and C_j are ww-changeable then all the data items of the set $d(C_i) \cap d(C_j)$ are overwritten without being read in the subhistory H_s.

Two histories H_1 and H_2 are *equivalent* iff for every consistent database state S and for all interpretations (except read and write commands) of the transactions, H_1 and H_2 map the same final database state. Intuitively, two histories are equivalent if they have the same effect on the database for all interpretations of transactions and all initial database states.

Proposition 1 *Let $H_1 = x_1 \cdot C_i \cdot x_2 \cdot C_j \cdot x_3$ be a history, and commands C_i, C_j be ww-changeable. The history H_1 is equivalent to the history $H_2 = x_1 \cdot C_j \cdot x_2 \cdot C_i \cdot x_3$.*

3 Serialization graph GS

A *serial history* H is a concatenation of execution sequences of all transactions in a certain order.

History H is *serializable* if there is a serial history which is equivalent to history H. A subhistory H_s is *serializable* iff there exists a serializable history H and H_s is a prefix of H. The decision problem of serializability (whether or not history H is serializable) is an NP-complete problem[1]. Then, we introduce a directed graph $GS(H_s) =< V_{H_s}, E_{H_s} >$ in which acyclicity is the sufficient condition for serializability of the subhistory H_s.

Let H_s be a subhistory. We define the directed graph $GS(H_s) =< V_{H_s}, E_{H_s} >$ of the subhistory H_s, where

$V_{H_s} = \{t_i |$ transaction t_i appears in $H_s\}$

$E_{H_s} = E_{H_{s_p}} \cup E_{H_{s_t}}$

$E_{H_{s_p}} = \{(t_i,t_j) | t_i$ and t_j satisfy conditions (1) or (2) below $\}$

$E_{H_{s_t}} = \{(t_i,t_j) | t_i$ and t_j satisfy conditions (3) or (4) below $\}$.

Condition (1): Commands C_{i_k} and C_{j_l} of transaction t_i and t_j respectively appear in the subhistory H_s. Commands C_{i_k} and C_{j_l} are in rw-conflict, and $C_{i_k} <_{H_s} C_{j_l}$.

Condition (2): Command C_{i_k} of transaction t_i appears in the subhistory H_s and, C_{i_k} and $RER(t,H_s)$ or C_{i_k} and $REW(t,H_s)$ are in rw-conflict.

Condition (3): Write commands C_{i_k} and C_{j_l} of transactions t_i and t_j respectively appear in the subhistory H_s. Commands C_{i_k} and C_{j_l} are in ww-conflict and are not ww-changeable.

Condition (4): The write command C_{i_k} of transaction t_i appears in the subhistory H_s, and the write command C_{i_k} and $REW(t,H_s)$ are in ww-conflict.

Proposition 2 *Let $H_1 = x_1 \cdot C_i \cdot C_j \cdot x_2$ be a history, and let commands C_i and C_j respectively be commands of transaction t_i and t_j. If transaction t_j is not reachable from transaction t_i in the graph $GS(H_s)$, then the history H_1 is equivalent to the history $H_2 = x_1 \cdot C_j \cdot C_i \cdot x_2$.*

Theorem 1 *If the directed graph $GS(H_s) =< V_{H_s}, E_{H_s} >$ of the subhistory H_s is acyclic, then subhistory H_s is serializable.*

Proof We can define the total order $<<_E$ on the set of the transactions V_{H_s}. This total order is consistent with the partial order $<<_G S$ defined by the acyclic directed graph $GS(H_s) =< V_{H_s}, E_{H_s} >$. Let S_h be the serial history of this total order $<<_E$ and let H be a history which is an element of the set $PSH(H_s)$ where;

$H = H_s \cdot H_{e_1} \cdot H_{e_2} \dots \cdot H_{e_q}$, and where if $i < j$ then $e_i <<_E e_j$. Using proposition 2, we can make the serial history S_h by

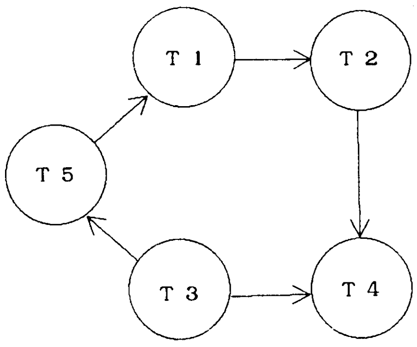

Figure 1: The graph GS(H) of the Example 1.

rearranging the commands in H so as to become equivalent to the serial history S_h. □

Example1 Let $t_1, \ldots, t_5$ be five straight line transactions such that $t_1 = R_1\{x\} \cdot W_1\{u\}$,

$t_2 = R_2\{a\} \cdot W_2\{x,y\}$,

$t_3 = R_3\{z\} \cdot W_3\{y,c\}$,

$t_4 = R_4\{b\} \cdot W_4\{y\}$,

$t_5 = R_5\{u\} \cdot W_5\{z\}$,

and

$H = R_1\{x\} \cdot R_2\{a\} \cdot W_2\{x,y\} \cdot R_3\{z\} \cdot R_4\{b\} \cdot W_3\{y,c\} \cdot W_4\{y\} \cdot R_5\{u\} \cdot W_5\{z\} \cdot W_1\{u\}$.

Figure 1 shows the graph GS(H).

In this case, command W_2 and W_3 are ww-changeable and the history H is equivalent to the serial history

$S_h = R_3\{z\} \cdot W_3\{y,c\} \cdot R_5\{u\} \cdot W_5\{z\} \cdot R_1\{x\} \cdot W_1\{u\} \cdot R_2\{a\} \cdot W_2\{x,y\} \cdot R_4\{b\} \cdot W_4\{y\}$.

History H is not in the class CPSR.

Let H be a history and $GS(H) =< V_H, E_H >$. A directed graph $CGS(H) =< V_H, CE_H >$ is defined as follows:

$CE_H = E_H \cup \{(t_i, t_j)$ There exist two write commands (command C_i of transaction t_i and command C_j of transaction t_j) in the history H such that $C_i <_H C_j$ and are ww-changeable in the history H $\}$.

For history H of the class CPSR, the graph $CGS(H) =< V_H, CE_H >$ is acyclic. If graph CGS is acyclic, then graph GS is also acyclic. From this, we get the following proposition.

Proposition 3 *The class of history H whose graph GS(H) is acyclic includes properly the class of the CPSR.*

4 The scheduler and protocols

Database D is partitioned into the data item sets D(s) for each SDBS s. Let $H_s = C_1 \cdot C_2 \cdot \ldots \cdot C_m$ be a subhistory and $C_i(s)$ be the restricted command to the D(s) of the command C_i such that $d(C_i(s)) = D(s) \cap d(C_i)$. We define the subhistory of SDBS s denoted $H_s(s)$ as follows:

$HH_s(s) = C_1(s) \cdot C_2(s) \cdot \ldots \cdot C_m(s)$, $d(C_i(s)) = d(C_i) \cap D(s)$ $(1 \leq i \leq m)$. From $HH_s(s)$, we get the subhistory $H_s(s)$ of SDBS s to remove the commands such that $d(C_i(s)) = \phi$.

All schedulers in the DDBS memorize the following three data:

(1) A subhistory $H_s(s)$ of the SDBS s.

(2) A directed graph $SGS(H_s(s)) = GS(H_s(s))$.

(3) List LR(s) of reachable sets R(t) from each transaction t in the graph $SGS(H_s(s))$.

For each read (or write) command, on-line schedulers check the acyclicity of the graph $GS(H_s)$.

Let H_s be the subhistory which was executed in the system DDBS. When a command C is requested by a transaction t of a SDBS s, each scheduler shall make a temporary graph $SGS(H_s(s_i) \cdot C(s_i))$. The schedulers check the acyclicity of the graph $GS(H_s \cdot C)$. If it is acyclic, then command C is accepted and,

(1). Graph $SGS(H_s(s))$ is updated to the graph $SGS(H_s(s) \cdot C(s))$.

(2). The subhistory $H_s(s)$ is updated to the subhistory $H_s(s) \cdot C(s)$.

(3). Every reachable set R(t) is updated.

If the graph is acyclic, command C is rejected and placed in a queue.

Graph $SGS(H_s, s)$, subhistory $H_s(s)$ of the SDBS s and all reachable sets R(t) are preserved in their former states (the scheduler S(s(t)) does not update any data).

For all queued commands C_w, every time a command is accepted, the scheduler tries to accept command C_w in the same manner as in which command C_w was previously requested. During the acyclic check of command C in a M-transaction (called the *check phase* of command C), all other commands are queued.

The schedulers use the following three messages for the preservation of serializability:

(1). Message $M1(t, Ci, RER(t), REW(t))$: Where command C_i of a M-transaction t, and parameters RER(t) and REW(t) are defined when command C_i is accepted.

(2). Message $M2(LRT, SI)$: Where LRT is the list of reachable sets from each M-transaction in the graph $GS(H_s \cdot C_i)$, and SI is the number of the scheduler which requested the broadcasting of this message M2.

(3). Message $M3$: The information that a cycle has been detected.

After a message $M1(t, C_i, RER(t), REW(t))$ has been broadcast, thereafter all schedulers can send only messages M2 or M3 to check the acyclicity of command C_i. All other messages are queued. The command C check phase ends when message M3 is broadcast or when all schedulers broadcast message M2 (there is no cycle in the graph $GS(H_s \cdot C_i)$).

Every scheduler S is assigned a number SI(S). These schedulers are totally ordered by their number. We denote this order $<_S$. In the check phase of a command C, every scheduler S_i sends a request for message M2(or M3) to be broadcast. These broadcast requests are sent in ascending order by scheduler number.

5 The algorithms DSA and analysis

5.1 The algorithms DSA for a command of a M-transaction

For a command C of a M-transaction t, the scheduler S of a SDBS s(t) sends a request for a message $M1(t, C, RER(t), REW(t))$ to be broadcast if there are any data items in the set d(C) which are not included in D(s(t)). If the set of data items d(C) is included in D(s(t)), then our algorithms DSA treats it as a command of a L-transaction.

Let TLR, BLR be lists of transaction sets. These lists exist in each scheduler for working areas. Initially list BLR is an empty list. The elements of each list for each transaction t are respectively denoted TRT(t) and BLR(t). The element TRT(t) of the list TRT is the subset of transactions reachable from transaction t in the graph $GS(H_s \cdot C)$. The element BLR(t) of the list BLR is a set of already broadcast transactions reachable from transaction t in the graph $GS(H_s \cdot C)$.

Each scheduler executes the following algorithms according to the message received. (M1, M2, or M3)

(Algorithm 1): When message $M1(t, C, RER(t), REW(t))$ broadcast by scheduler S_r of the SDBS s_r is received:

(1). Every scheduler S_i such that $d(C) \cap D(s_i) \neq \phi$, temporarily makes;

(a). the subhistory $TH(s_i) = H_s(s_i) \cdot C(s_i)$ of SDBS s_i,

(b). the subgraph $TGS(s_i) = GS(H_s \cdot C(s_i))$ and

(c). the list TLR of reachable sets R(t) from each transaction t in the set TV of the subgraph $TSG = (TV, TE)$.

(2). If $SI(S_j)$ is the smallest number of all the schedulers, then scheduler S_j sends a request for the message $M2(TLR, SI(S_j))$ to be broadcast.

(Algorithm 2): When message $M2(LRT, SI(S_r))$ broadcast by scheduler S_r of the SDBS s_r is received :

```
Let $LRT = R1(t_1) \cdot R1(t_2) \cdot \ldots \cdot R1(t_k)$.
For every scheduler $S_i$ except scheduler $S_r$,
begin
if the number $SI(S_r)$ is not largest number of all the schedulers,
then
  (step1) for every M-transaction $t_j$ $(1 \leq j \leq k)$
    begin
      $BLR(t_j) := BLR(t_j) \cup R1(t_j)$;
      if $R1(t_j) - TLR(t_j) \neq \phi$
        then $TLR(t_j) := R1(t_j) \cup TLR(t_j)$
    end;
  while there exists a M-transaction $t_h$ such that
    the M-transaction $t_h$ is an element of $TLR(t_p)$
      and $TLR(t_p) - TLR(t_h) \neq \phi$ do
    for all M-transactions do
      $TLR(t_p) := TLR(t_p) \cup TLR(t_h)$;
  (step2) if the set TLR(t) includes the transaction t itself
    then scheduler $S_i$ sends a request for message M3 to
         be broadcast when there is no other scheduler $S_j$
         such that scheduler $S_j$ has not broadcast message
         M2 and the number $SI(S_j) <_S SI(S_i)$
    else
      begin
        if there is no other scheduler $S_j$ such that
          scheduler $S_j$ has not broadcast message M2
          and the number $SI(S_j) <_S SI(S_i)$
          then
            begin
              for every M-transaction $t_h$
                $TR2(t_h) := TLR(t_h) - BLR(t_h)$;
                the scheduler $S_i$ sends a request
                  for the message $M2(TR2, SI(S_i))$
                          to be broadcast
            end;
      if the number $SI(S_r)$ is the largest number of
            the all schedulers {no cycle detected}
        then
          begin
```

```
                $H_s(s_i) := TH(s_i)$;
                        $\{TH(s_i) = H_s(s_i) \cdot C(s_i)\}$
                $SGS(H_s(s_i))$:=$TGS(s_i)$;
                    $\{TGS(s_i) = GS(H_s(s_i) \cdot C(s_i))\}$
                $LR(s_i) := TLR$;
                for every M-transaction $t_h$
                    $BLR(t_h) := \phi$;
                ends the check phase of the command C
            end
        end
end;
```

(Algorithm 3): When message M3 broadcast by a scheduler S_r of the SDBS s_r is received :

```
begin
    for every M-transaction $t_h$
        $BLR(t_h) := \phi$;
    ends the check phase of the command C
end;
```

5.2 The algorithms DSA for a command of a L-transaction

For a command C of a transaction t, if set d(C) of data items is included in set D(s(t)), then scheduler S of SDBS s(t) temporarily makes

(a). the subhistory $TH(s) = H_s(s) \cdot C$,

(b). the subgraph $TGS(s) = SGS(H_s(s) \cdot C)$ and

(c). the list TLR of reachable sets $R(t_h)$ from each transaction t_h in the graph $TGS(s)$. And the scheduler S shall execute the following algorithm:

(Algorithm 4):

```
begin
    while there exists a transaction $t_h$ such that
        transaction $t_h$ is an element of $LR(t_p)$
        and $LR(t_p) - TLR(t_h) \neq \phi$ do
          for all M-transactions do
            $TLR(t_p) := LR(t_p) \cup TLR(t_h)$;
    if transaction t is not an element of the set TLR(t)
        then        no cycle detected
          begin
            $H_s(s) := TH(s)$;        $\{TH(s) = H_s(s) \cdot C\}$
            $SGS(H_s(s))$:=$TGS(s)$;
                $\{TGS(s) = GS(H_2(s) \cdot C)\}$
            $LR(s) := TLR$;
          end
        else the command C is queued;
end.
```

5.3 The analyses

5.3.1 Analysis 1

Let C be a command of a transaction t which is executed in the SDBS s(t) and let n be the number of SDBS in the DDBS. If $d(C) - D(s(t)) \neq \phi$ then every scheduler shall broadcast message M2 or M3 only once. So the maximum number of broadcasts in the DDBS is $n+1$ times for each command of a M-transaction such that $d(C) - D(s(t)) \neq \phi$.

5.3.2 Analysis 2

Let $REWW(t, H_s)$ be the set of data items for active transaction t in subhistory H_s and be defined as follows :

$REWW(t, H_s) = \{d_i | d_i$ is an element of the set $d(C_i)$, where command C_i is a write command of an active transaction t in the subhistory H_s, and write command C_i is an element of each remaining execution sequence of transaction t with the subhistory H_s.$\}$

A data item in the set $REWW(t, H_s)$ will always be overwritten in the rest of subhistory H_s unless a new transaction occurs.

Let write commands C_i and C_j appear in subhistory H_s such that $C_i <_{H_s} C_j$ and let there be no read command C_r in subhistory H_s such that $C_i <_{H_s} C_r$ and $d(C_i) \cap d(C_j) \cap d(C_r) \neq \phi$.

By extending the definition of "ww-changeable" to "eww-changeable",

we can expand the class of the histories which will be accepted by our algorithms DSA. But in the case of$d(C) - D(s(t)) = \phi$, message M2 must be broadcast.

We say that write commands C_i and C_j are *ww-changeable* iff

The write commands C_i and C_j are ww-changeable or satisfy all of the following three conditions:

Condition (1). There is no read command C_k in the subhistory H_s such that $d(C_i) \cap d(C_j) \cap d(C_k) \neq \phi$

Condition (2). For any active transaction t in the subhistory H_s, $d(C_i) \cap d(C_j) \cap RER(t, H_s) = \phi$

Condition (3). Each data item in the set $d(C_i) \cap d(C_j)$ is an element of set $REWW(t, H_s)$ for active transaction t with the subhistory H_s such that $C_j <_{H_s} C_w$, or is an element in set $d(C_w)$ for a write command in the subhistory H_s such that $C_j <_{H_s} C_w$.

5.3.3 Analysis 3

We assumed that messages M2 are broadcast according to a predefined order, but this assumption can be changed to :

"Any message which has already been requested but has not yet been broadcast can be cancelled just before it is broadcast."

If for example, scheduler S has received a request to broadcast message M2a but has not yet broadcast this message, and message M2b is received from another scheduler, then scheduler S will cancel the request for message M2a to be broadcast and will make a new message M2c or M3 by using the algorithm A2. Thus, each scheduler will broadcast message M2 only once to cycle check a command.

5.3.4 Analysis 4

If all of the above assumptions are removed, then in the worst case, the schedulers may be required to broadcast messages M2 $n(n-1)/2$ times. For example:

In the check phase of command C, each message M2 has an integer. These integers are assigned in the following manner:

(1). The first message M2 broadcast during a check phase is "1".

(2). Each scheduler ignores the previously broadcast message M2(LRT1,N1) if message M2(LRT2,N2) has already been received such that $N2 \geq N1$.

(3). When each scheduler sends a request for message M2(LRT,N) to be broadcast such that $N = N1 + 1$ where N1 is the message with the greatest number in the current check phase.

If scheduler S sends a request for message M2 to be broadcast, and M2 is subsequently broadcast and is not ignored in the current check phase, thereafter scheduler S will not request any more messages to be broadcast.

If message M2(LRT,N) is broadcast and is not ignored, then in the worst case, there may exist $N-1$ M2 messages which were previously requested by the same scheduler. However, these previously requested messages will be ignored.

5.3.5 Analysis 5

The information to be memorized in a SDBS increases in proportion to the number of transactions. But if the in-degree of a transaction t is zero in the graph $SGS(s) =< V(s), E(s) >$ and the transaction t is terminated, then we can remove the transaction from graph SGS. The increase in information is resolved by periodically broadcasting the set of ended transactions.

6 Summary

In concurrency control, the polynomial classes of the history may be improved by another ww-changeable condition. But the algorithms may become more complicated. Our scheduler protocol for a distributed database system with broadcast communication function may be applicable to the serializability preservation algorithm for distributed database systems with broadcast communication function if an acyclicity check of the directed graph G is executed where Graph G is the union of the local subgraphs. One such application would be the creation of a more simple algorithm for the class CPSR.

References

[1] P.A.Bernstein, D.W.Shipman, W.S.Wong,"Formal Aspects of Serializability in Database Concurrency Control", IEEE Trans. Software Eng., Vol.SE-5, No.3, pp.203-216, May 1979.

[2] M.A.Casanova, "The Concurrency Control Problem for Database Systems", Lecture Notes in Computer Science: Springer-Verlag 1981.

[3] K.P.Eswaran, J.M.Gray, R.A.Lorie, I.L.Traiger, "The Notions of Consistency and Predicate Locks in a Database System", CACM Vol.19, No.11, pp.624-633, Nov. 1976.

[4] H.T.Kung, C.H.Papadimitriou, "An Optimality Theory of Concurrency Control for Databases", Acta Informatica, Vol.19, pp.1-11 1983.

[5] P.A.Bernstein, J.B.Rothnie, N.Goodman, C.A.Papadimitriou, "The Concurrency Control Mechanism of SDD-1: A System for Distributed Databases (The Fully Redundant Case)", IEEE Trans. Software Eng.,Vol.SE-4, No.3, pp.154-168, May 1978.

[6] Z.Kedem, A.Silberschatz, "Controlling Concurrency Using Locking Protocols", in Proc. 20th IEEE Symp. Found. Com. Sci. pp.274-285, Oct. 1979.

[7] P.L.Lehman, S.B.Yao, "Efficient Locking for Concurrent Operations on B-Trees", ACM Trans. Database System Vol.6, No.4, pp.650-670, Dec. 1981

[8] C.H.Papadimitriou, "The Serializability of Concurrent Database Updates", JACM Vol.26, No.4, pp.631-653, Oct. 1979

[9] C.H.Papadimitriou, "A Theorem in Database Concurrency Control", JACM Vol.29, No.4, pp.998-1006, Oct. 1982

[10] C.H.Papadimitriou, P.A.Bernstein, J.B.Rothnie, "Some Computational Problems Related to Database Concurrency Control" in Proc. Conf. Theoretical Comp. Sci., pp.275-282, Aug. 1977

[11] A.Silberschatz, Z.M.Kedem, "A Family of Locking Protocols for Database Systems that are Modeled by Directed Graphs", IEEE Trans. Software Eng. Vol.SE-8, No.6, pp.558-562, Nov. 1982

[12] A.Silberschatz, Z.M.Kedem, "Consistency in Hierarchical Database Systems" JACM Vol.27, No.1, pp.72-80, Jan. 1980

[13] M.Yannakakis, "Issues of Correctness in Database Concurrency Control by Locking", STOC pp.363-367, 1981

[14] M.Yannakakis, "A Theory of Safe Locking Policies in Database Systems", JACM Vol.29, No.3, pp.718-740, July 1982

[15] M.Yannakakis, C.H.Papadimitriou, H.T.Kung, "Locking Policies: Safety and Freedom from Deadlock", in Proc. Conf. Theoretical Computer Science, pp.286-297, Aug. 1977

[16] S. Harashima, T. Ibaraki, "Concurrency Control of Distributed Database Systems by Cautious Schedulers", The Transactions of the IEICE, Vol.J70-J, No.6, pp.1140-1148, (1987)