CERTIFICATION BY INTERVALS OF TIMESTAMPS IN DISTRIBUTED DATABASE SYSTEMS

Claude Boksenbaum, Michèle Cart, Jean Ferrié, Jean-François Pons

C.R.I.M. Université de Montpellier
860, rue de Saint Priest
34100 **MONTPELLIER - FRANCE**

Abstract

This paper introduces, as an optimistic concurrency control, a **new certification method** by means of intervals of timestamps, usable in a **distributed database system.** The main advantage of this method is that it allows a chronological validation order which differs from the serialization one (thus avoiding rejections or delays of transactions which occur in usual certification methods or in classical locking or timestamping ones). The use of the dependency graph permits both classifying this method among existing ones and proving it.

1. INTRODUCTION

Numerous Concurrency Control Methods (CCMs) for Distributed Data Bases (DDBs) have been proposed so far [Bernstein 81] using either Two-Phase Locking (2PL) or Timestamp Ordering (TO).

The main feature of these CCMs is that the serializability test is made for each action (Read or Prewrite) on an object of the base. For this reason, we call these CCMs **continuous** ones as opposed to **certification** (also called **optimistic)** ones, where the serializability test is only made at the end of the transaction.

Usually, continuous CCMs and certification ones are described in totally different ways. Our study offers a unifying view for this classification.

Certification methods [Kung 81], [Haerder 82], [Lausen 82] have been studied mostly for centralized systems. The few proposals adapted to distributed systems [Badal 79], [Schlageter 81], [Bhargava 82] are not convincing due to lack of formal proofs for their desired behavior. We answer that problem by proving a certification method in a distributed system.

Serialization control used in CCMs relies on the setting of a serialization order among transactions. In continuous CCMs two principles are used to construct this order. In 2PL methods [Traiger 82], it is dynamically constructed and corresponds to the order into which transactions reach their maximum locking point. The well known drawback of these methods is useless waiting sometimes imposed to transactions and deadlocks which may occur. Deadlock prevention may be applied [Rosenkrantz 78] at the cost of increasing the number of rejections. In basic or multiversions TO CCMs, the order is "static" and relies upon the giving of a unique timestamp to any transaction, thus defining the serialization order. One study [Bayer 82] refines this technique by introducing the notion of a "dynamic" timestamp which avoids useless rejection of a transaction when its first conflict arises, in a very particular CCM.

Control by certification allows a dynamic construction of the serialization

Proceedings of the Tenth International Conference on Very Large Data Bases.

Singapore, August, 1984

order by delaying it until validation. At that moment, more information on dependencies among transactions is available than in continuous methods. Nevertheless, previously described methods [Kung 81],[Vieront 82],[Schlageter 81] impose that this order should be identical to the chronological order of validations.

In this study, we propose and prove a **new certification CCM** which, by using **intervals of timestamps**, allows construction of the serialization order which may differ from the chronological order of validations. Moreover, the gradual construction of the serialization order makes detecting non-serializable actions possible without waiting until the transaction's validation. This characteristic connects our method to continuous CCMs and allows their comparison.

More precisely, the advantages and originalities of our method compared to known CCMs are the following:

1) It allows a chronological order of validations which differs from the serialization order thus decreasing the number of rejections or waitings.

The multiversions TO CCMs [Reed 78] which also do not impose identity between these two orders request a lot more information (versions and timestamps) about transactions' history than our method.

2) It allows an "old" transaction to use, in some cases, results of "younger" ones.

This property which requires dynamical construction of the serialization order without using chronological starting time of transactions is also achieved by 2PL CCMs. Intervals of timestamps permit this property to be obtained in our method.

3) It allows favoring a particular class of transactions (reading or writing ones, for instance).

Such strategies may be implemented by proper choice of the final timestamp inside the interval of timestamps associated to a given transaction.

4) It is fully integrated into a distributed validation protocol.

Our study is organized as follows.

In section 2, we present the notion of conflicts and the notion of dependency graph which is the basis of formal proof of our method. This graph maintains two types of dependencies: effective ones and potential ones. It allows classifying various CCMs as continuous or optimistic (either "backward" or "forward") according to their serialization criterion and shedding some light on their limitations.

In section 3, the principle and proof of an **original certification CCM** using intervals of timestamps are described.

Section 4 specifies the implementation in a distributed environment and details the **certification protocol** used, which is also applicable to other optimistic CCMs.

2. CONFLICTS AND DEPENDENCY GRAPH

Conflicts and dependencies

The life of a transaction is divided into several sequential steps:
(1) a step when **reads** (R) and **prewrites** (P) of objects occur in the transaction's workspace,
(2) a **certification** step present only in optimistic CCMs, checking whether all operations of a transaction are serializable,
(3) possibly a validation step, only present when the transaction is serializable, whose effect is to turn on definitively, in an atomic manner, the prewritings of a transaction.

Two transactions **conflict** if one tries to prewrite [resp. read] an object already read or prewritten [resp. prewritten] by the other. Depending on the nature and chronological order of the operations, **PP**, **RP** and **PR** conflicts are distinguished.

A conflict between T1 and T2 induces a constraint on the serialization order of T1 and T2 which can be translated into a dependency taking one of the following two forms:

a) an **effective dependency** (T1⟶T2) which means that T1 precedes T2 in the serialization order;

b) a **potential dependency** (T1⇢T2) which means that the order is not yet determined but will be determined at last when one of the transactions validates.

Table 1 summarizes all possibilities for setting of a dependency - assuming that the operation of T1 precedes chronologically that of T2 - according to the nature of conflict and whether T1 is **validated** (then noted $T1^*$) or not (then called a **living** transaction).

Type of the conflict	Nature of the dependency when the conflict occurs depending whether: T1 is validated	T1 is living
T1 R — T2 P	$T1^* \longrightarrow T2$	$T1 \longrightarrow T2$
T1 P — T2 P	either $T1^* \longrightarrow T2$ or $T2 \longrightarrow T1^*$ and T2's writing ignored: Thomas' Rule	$T1 - - - T2$
T1 P — T2 R	$T1^* \longrightarrow T2$	either $T1 \longrightarrow T2$ and delayed reading or $T2 \longrightarrow T1$ and immediate reading

Table 1.

The dependency graph

The execution of a set of transactions may be represented by a **dependency graph G** whose nodes are transactions and edges are dependencies induced by their conflicts. G changes when a conflict arises or when a transaction is rejected or validated. In particular, validation of a transaction T transforms its potential dependencies $T - - - T'$ into effective ones either of the form $T^* \longrightarrow T'$ or of the form $T' \longrightarrow T^*$ when Thomas' rule [Thomas 79] has been chosen to be applied which means ignoring the writes of T'.

It has been proved [Papadimitriou 79] that a circuit of effective dependencies in G prevents all transactions involved in that circuit to be serialized (i.e. to be equivalent to a serial execution of the same transactions).

Let us note that, as long as a transaction is not validated, the effective and potential dependencies it has generated translate actions whose effects have not yet been registered into the base. Conversely, since a validation is irreversible, it is necessary that all validated transactions be serializable, i.e. that no circuit exists among them. In order to obtain serialization of transactions, a solution is to reject a living transaction as soon as the setting of one of its dependencies leads to a circuit in G.

In order to show limitations of the various known CCMs, it is useful to distinguish the following subgraphs:

- G^*, the validated graph, restricted to all validated transactions with their dependencies;
- G^*_T, the selfish graph of T, restricted to all validated transactions and to the living transaction T with their dependencies;
- G^+, the living graph, restricted to all living transactions with their dependencies.

A total order among transactions which is compatible with the partiel order defined by G^* is called a **serialization order**.

All CCMs apply a serialization criterion to keep G^* circuitless. In fact, depending on methods used, one or more preceding subgraphs are privileged. However, no method implements any graph to determine the serialization order. Indeed in a distributed environment exact view of the graph would be difficult to obtain on each site. Moreover it would not be compatible with a satisfying level of parallelism. Only one optimistic CCM lying on a centralized management of G in a local broadcast network has been proposed so far [Dewitt 80].

Continuous control and the dependency graph

Continuous CCMs are characterized by the fact that no checking upon dependencies is necessary at validation time. That is why they check when reading or prewriting, the dependencies'set-up in order to insure that G^*_T is circuitless for all living transactions T and stays that way, whatever transaction is validating . They do so by forcing potential dependencies to be converted into effective ones in case of a PP conflict. In such a case validation will not change any dependency. This advantage is paid by more rejections or waits for transactions.

In 2PL methods [Rosenkrantz 78], [Eswaran 76], [Traiger 82] the chronological order of operations induces dependency when a conflict occurs: if the operation of T_1 precedes chronologically that of T_2, the dependency $T_1 \longrightarrow T_2$ is set. If T_1 is already validated the dependency corresponds to table 1 (when Thomas' rule is never applied).

In case of conflict between living transactions, T_2 is then **blocked** (sometimes uselessly) until T_1 is validated in order to force the dependency $T_1 \xrightarrow{*} T_2$: parallelism is thus decreased. In these methods the serialization order corresponds to the chronological validations' order. The possibility of a deadlock, characterized by a circuit in G^+, is another drawback of these methods.

In TO CCMs [Bernstein 81] the order of the **timestamps** associated to transactions at their birth defines an "a priori" serialization order . If, when a conflict occurs, the effective dependency it generates does not correspond to the timestamps order, the transaction which triggers the conflict is rejected. In case of a PP conflict between living transactions, the dependency chosen is that of the timestamps' order, imposing validations to follow this order (the serialization one). This is done by making a transaction wait if it wants to validate earlier than other ones which precede it in the serialization order. The TO CCMs insure that G is always circuitless. A variant method, the multiversions TO [Reed 78] allows accepting reads and under certain conditions prewrites (and thus validation) even when their chronological order does not respect the order of timestamps at the high price of multiple timestamped versions of the same object. The major drawback of TO CCMs is setting "a priori" dependencies before a conflict occurs thus inducing useless rejections. Moreover, the more the timestamps' order is far from the real chronological order of the transactions' birth, the more rejections are generated. This can be avoided by resynchronizing timestamps' generators [Lamport 78] by inter-site messages. At last, transactions may suffer from starvation if repeated rejections occur. An improvement of TO CCMs consists in waiting for the transaction's first conflict before giving it a timestamp compatible with the induced dependency [Bayer 82]. This postpones the previous drawback to the second conflict when the timestamp is defined: this may induce a dependency between this transaction and those which are not and will never be in conflict with it.

Control by certification and dependency graph

Many strategies are possible depending whether certification of a transaction T is considered in G^*, G or G^+. Nevertheless in all known CCMs except ours, the serialization order corresponds to chronological order of validations.

In a **certification strategy with G^***, concurrency control uses the validated transactions' history. A transaction T is certified successfully if G^*_T is circuitless. For that reason, such a method is called "backward control". The method described in [Kung 81], suggested for a centralized system but which may be applied as well in DDBs, illustrates this type of control. During the certification of T, the conflicts it has had with all validated transactions T_i are checked to be of the form $T_i \xrightarrow{*} T$. Since PP conflicts follow validations' order (i.e. serialization order) they are ignored as well as conflicts of the type $T_i \xrightarrow{* RP} T$. In case of a PR conflict between T_i (which prewrites) and T (which reads) dependency's direction changes depending whether the read of T precedes chronologically ($T \xrightarrow{RP} T_i^*$) or follows ($T_i \xrightarrow{* PR} T$) the validation of T_i. Since the method does not distinguish between the relative order of reads and validations, it forbids $T \xrightarrow{RP} T_i^*$ and thus it also forbids $T_i \xrightarrow{* PR} T$ which might be valid. The serialization criterion of T is:
$Owriteset(T_i^*) \cap Oreadset(T) = \emptyset$, for all T_i^*, where Oreadset(T) and Owriteset(T) are the sets of objects respectively read and prewritten by T.

Another proposal [Lausen 82] aims at forbidding only dependencies of the type $T \xrightarrow{RP} T_i^*$, by timestamping the reads relatively to the validations.

A **certification strategy with G** allows more freedom because not only validated transactions are considered but also living ones under the angle of their future validations. In order to do so, all circuits in G which contain T are looked for when T certifies. If no circuit is found, T may be certified, otherwise the circuits must be eliminated by either:

i) rejecting T;
ii) rejecting another living transaction in each of the circuits for instance in order to minimize future rejections;
iii) -in the special case when each circuit contains a potential dependency T---T' which has been translated into T⟶T' - making T wait until the certification of T' which will invert the dependencies, or certifying T using Thomas' rule for T' (giving T'⟶T*).

So far no method illustrates this type of strategy.

In a **certification strategy with G^+**, control does not use the history of validated transactions but considers only living transactions. It is therefore called a "forward control" [Haerder 82]. If, during the certification of T there are $T_i \longrightarrow T$ dependencies (corresponding to RP conflicts) either all transactions which induce such dependencies are rejected or T is made to wait until T_i is validated or T is rejected. Here the serialization criterion is:

$Owriteset(T) \cap Oreadset(T_i) = \emptyset$, for all T_i.

The method described in [Schlageter 81] achieves this type of forward control in order to favor transactions which read, by delaying the validations of writing transactions which conflict with them. The transactions which read and write are serialized using a backward control.

Generally no previously described CCM by certification allows $T \longrightarrow T_i^*$ dependencies: the major drawback is that it induces either waits (by delaying validations) or rejections (if validations must be immediate). At last, attention must be drawn to the fact that all CCMs by certification must process transactions' certifications in the **same** order on all sites in a distributed system. Indeed all sites must make the **same** decision based on the same sets of living transactions (i.e. G^+) or validated ones (i.e. G^*).

3. CERTIFICATION BY INTERVALS OF TIMESTAMPS

Principle of the method

The certification method we present uses a backward control: the serialization criterion for T must avoid any circuit in G^*_T.

A summary of the dependencies in the preceding subgraphs is obtained by:
- timestamps expressing dependencies in G^*,
- intervals of timestamps summarizing dependencies in G^*_T,
- sets of objects translating dependencies in G^+.

The use of each mechanism is precised by the following.

Timestamps: to each validated transaction T_i^* is associated a timestamp t_i given at certification time such that:

Property P1. The order induced by timestamps values is compatible with the partial order of G^*.

Intervals of timestamps: to each living transaction T and for each object x that it has used (read or prewritten) is associated an interval of timestamps $I(T,x)$[(1)]. The bounds of this interval express the strongest constraints between T and validated transactions which have accessed to x and must respectively precede or follow T.

More precisely, if we denote $LI(T,x)$ and $UI(T,x)$ respectively the lower bound and the upper bound of $I(T,x)$, the following must hold:

Property P2. For each living transaction T and for each object x it has used, $I(T,x)$ is compatible with the timestamps of all validated transactions which have also used x. This means that:

$\forall T$ living transaction, $\forall x$ object used by T,
$\forall T_i^*$ validated transaction having used x,
either $T_i \longrightarrow T$ and then $t_i < LI(T,x)$
or $T \longrightarrow T_i^*$ and then $UI(T,x) < t_i$

Notice that intervals of timestamps allow keeping track of $T \longrightarrow T_i^*$ dependencies. Thus it will be possible for a transaction T which precedes some validated ones in the serialization order to be certified and validated after them.

Grouping all constraints upon T can be obtained by:

$I(T) = \bigcap_{x \in X} I(T,x)$ X being the set of objects used by T.

(1) In practice, only one interval per site S, $I(T,S)$, may be used.

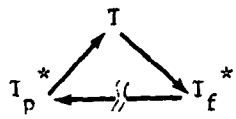
Proposition 1. Any living transaction T which introduces a circuit in G^*_T verifies I(T)=∅.

Proof: consider a transaction T which introduces such a circuit in G^*_T. In this circuit, let us call:

. T_f^* the transaction which follows T, x one object by which the dependency comes, and

. T_p^* the transaction which precedes T, y one object by which the dependency comes (we may have f=p).

This circuit induces the following relations:



From **P1**, we have: $T_f^* \rightarrow T_p^* \Leftrightarrow t_f \leq t_p$

From **P2**, we have: $T \rightarrow T_f^* \Leftrightarrow UI(T,x) < t_f$

and $T_p^* \rightarrow T \Leftrightarrow t_p < LI(T,y)$

Since $I(T) = \bigcap_{x \in X} I(T,x)$ we have:

$\forall x \in X \quad LI(T,x) \leq LI(T)$ and $UI(T,x) \geq UI(T)$

We finally have:

$LI(T) \geq LI(T,y) > t_p \geq t_f > UI(T,x) \geq UI(T)$

thus: $LI(T) > UI(T) \Leftrightarrow I(T) = \emptyset$. □

Sets of objects: to each living transaction T are associated the set of objects it has read (**Oreadset**) and the set of objects it has prewritten (**Owriteset**). As a result, between two living transactions T and T':

a) effective dependency $T \rightarrow T'$ is equivalent to:

$Oreadset(T) \cap Owriteset(T') \neq \emptyset$

b) potentiel dependency $T \dashrightarrow T'$ is equivalent to:

$Owriteset(T) \cap Owriteset(T') \neq \emptyset$

The principle of the method is to keep properties **P1** and **P2** true through all actions read, prewrite and certify:

- when T reads or prewrites x, the interval I(T,x) must be updated according to property **P2**;

- when T asks for its certification, I(T) must be computed according to its definition. Then if $I(T) \neq \emptyset$, T is certified, otherwise T is rejected.

Moreover when T is certified, G^* is modified and therefore so are $G^*_{T'}$ for all other living transactions T'. The modifications are made by:

. choosing a timestamp for T which keeps on property **P1**.

. adjusting the interval I(T',x) of all transactions T' which have been in conflict with T, on any object x used by T.

The following proposition shows how a timestamp may be associated to a transaction:

Proposition 2. If a transaction T, with $I(T) \neq \emptyset$, asks for certification any $t \in I(T)$ will verify property **P1**.

Proof: I(T) denotes the strongest constraints between T and the validated transactions. More precisely:

$\forall T_k^*$ such that $T \rightarrow T_k^* \Leftrightarrow UI(T) < t_k$ and

$\forall T_j^*$ such that $T_j^* \rightarrow T \Leftrightarrow LI(T) > t_j$

Thus any $t \in I(T)$ (i.e. $LI(T) \leq t \leq UI(T)$) will be compatible with the dependencies in G^*_T. □

In 4, we shall show that the choice of t=LI(T) (or UI(T)) has a definite effect on the future of some living transactions.

When the timestamp t for transaction T is chosen, adjustment of all intervals I(T',x) of the living transactions T' will be done according to the type of conflicts between T and T' on x. Let us recall that these dependencies are recorded by means of sets of objects. The adjustment will take the forms:

if $T' \rightarrow T$ then t is the right truncation point for I(T',x),

if $T \rightarrow T'$ then t is the left truncation point for I(T',x),

if $T \dashrightarrow T'$ two translations are possible:

either $T \rightarrow T'$ with appropriate adjustment of I(T',x)

or $T' \rightarrow T$ with appropriate adjustment of I(T',x) and ignoring the writing of T' at validation time (Thomas' rule).

In order to make the method useful in practice, it is necessary to "forget" validated transactions. This is done, as in TO methods, by timestamping objects: to each object are associated the highest timestamps (noted respectively R(x) and W(x)) of the validated transactions which have read or prewritten x. With these timestamps, the actions read and prewrite may use proposition 3 to keep on property **P2.**

Proposition 3. When a transaction T reads [resp. prewrites] an object x, fixing $LI(T,x) > W(x)$ [resp. $> \max(R(x),W(x))$] will respect the dependencies with the validated transactions T_i^* under the form $T_i^* \longrightarrow T$.

Proof: when a transaction T reads x, it conflicts with all validated transactions T_k^* having prewritten x. ($T_k^* \xrightarrow{*\ PR} T$)

Since for all T_k^*, $t_k \leq W(x)$ and since we set $W(x) < LI(T,x)$ we have $t_k < UI(T,x)$.

Similarly when a transaction T prewrites x, it conflicts with all validated transactions T_j^* having read or prewritten x ($T_j^* \xrightarrow{*\ RP} T$ or $T_j^* \xrightarrow{*\ PP} T$).

Since for all T_j^*, $t_j \leq \max(W(x), R(x))$ and since we set $\max(W(x), R(x)) < LI(T,x)$ we also have $t_j < LI(T,x)$. □

Thus during a read or prewrite operation on an object x by a transaction T, only $T_k^* \longrightarrow T$ dependencies between T and validated transactions T_k^* in conflict with T are set up (Thomas' rule will never be applied in such cases).

4. IMPLEMENTATION IN A DISTRIBUTED SYSTEM

Data structures local to each site

I(T) which retains the strongest constraints between T and validated transactions might be computed upon any partition of the set of objects, since the intersection of these intervals would give the same result as the intersection of I(T,x) on all objects x accessed by T. Partition by site seems particularly attractive in a distributed system.

For each living transaction T which has accessed to the set X_i of objects on the site S_i, this site manages the interval $I(T,S_i)$:

$$I(T,S_i) = \bigcap_{x \in X_i} I(T,x) \ .$$

$I(T,S_i)$ is created when T makes its first access to an object on site S_i, and will be locally updated by site S_i either during further reads and prewrites of the transaction T on objects located on S_i or independently of T during adjustment following certification of a transaction T' having been in conflict with T on some objects of site S_i.

All effective dependencies due to conflicts between T and validated transactions on all sites are expressed by:

$$\bigcap_{site_j} I(T,Sj) = \bigcap_{site_j} \bigcap_{x \in X_j} I(T,x) = I(T) \ .$$

In order to make the adjustment mentionned hereabove, each site S_i also manages the sets **Oreadset**(T,S_i) and **Owriteset**(T,S_i) which are respectively the set of objects read by T on S_i and the set of objects prewritten by T on S_i. Each site S_i also maintains the set **Tlivingset**(S_i) of all living transactions known by it.

Access to the objects

In order to detect as soon as possible that an operation requested by a transaction is not serializable - and so have a kind of continuous control - to each transaction T is associated its **current interval** $I_c(T)$. At the beginning of T, $I_c(T)$ is initialized as $[o,\omega[$ (the whole set of allowed timestamps). $I_c(T)$ is transmitted during each read or prewrite made by T to the various sites accessed by T. These sites modify the interval according to dependency induced by the operation and send it back to the transaction T. Thus each site gets a more precise view on the dependencies involving T which have been set on other sites.

Notice that now $I(T,S_i)$ not only carries information about the dependencies set on site S_i but also has some knowledge of the preceding constraints put on $I_c(T)$. The final intersection of all $I(T,S_i)$ will still give the same I(T) since although some contraints will be counted many times, the intersection will not be affected.

The advantage of using $I_c(T)$ which can be considered as an approximation of I(T), is that it conveys more information on the constraints put on T, and therefore may signal rapidly that a transaction T must be rejected

when $I_c(T) = \emptyset$. Taking this shortcut into account, the semantics of operations **read**(x,T,$I_c(T)$) and **prewrite**(x,T,$I_c(T)$) which preserves proposition 3 are given below. Each algorithm is executed on the site S_i of the object x. For these algorithms it is assumed that the intervals contain only integers.

```
read(x,T,I_c(T))
    =(if T∈Tlivingset(S_i)
          then I_c(T):=I_c(T)∩I(T,S_i)
          else Tlivingset(S_i):= Tlivingset(S_i)∪{T};
      I_c(T):=I_c(T)∩[E(x)+1,+∞[;
      I(T,S_i):=I_c(T);
      Treadset(x):=Treadset(x)∪{T};
      Oreadset(T,S_i):=Oreadset(T,S_i)∪{x};
      M:=(val(x),T,I_c(T));
      send-message(M,S_i,S(T))
    );
```

```
prewrite(x,T,I_c(T))
    =(if T∈Tlivingset(S_i)
          then I_c(T):=I_c(T)∩I(T,S_i)
          else Tlivingset(S_i):= Tlivingset(S_i)∪{T};
      I_c(T):=I_c(T)∩[E(x)+1,+∞[∩[L(x)+1,+∞[;
      I(T,S_i):=I_c(T);write(copy(x,T));
      Twriteset(x):=Twriteset(x)∪{T};
      Owriteset(T,S_i):=Owriteset(T,S_i)∪{x};
      M:=(T,I_c(T));
      send-message(M,S_i,S(T)))
    );
```

Local adjustment of the intervals

In order to fulfil property **P2**, certification of a transaction T implies adjustment of intervals of living transactions T' in conflict with T, on all sites S_i where T is known as having accessed to some objects. In order to speed up the search for all transactions T' in conflict with T, each site associates to each object x, the sets **Treadset**(x) and **Twriteset**(x) of living transactions which have respectively read or prewritten x.

The semantics of the procedure **adjust**(T,t_T) which preserves property **P2** through the certification of T with timestamp t_T, is described below. It must be executed on all sites S_i accessed by T.

We have included in this procedure , by means of the procedure **swap** , the validation phase of T which corresponds to the copy of the new values of objects prewritten by T into the part of base managed by S_i.

On each site the adjustment of intervals following the certification of T must not interfere with processing of reads or prewrites issued by other transactions T' : for that reason, the procedure adjust must be considered, on each site, as a **critical section** which must be executed in mutual exclusion with read and prewrite operations .

```
procedure adjust(T,t_T)
    =(for x∈Oreadset(T,S_i) do
          (for T'∈Twriteset(x) do
               I(T',S_i):=I(T',S_i)∩[t_T+1,+∞[;
           L(x):=max(L(x),t_T);
           Treadset(x):=Treadset(x)-{T});
      for x∈Owriteset(T,S_i) do
          (for T'∈Treadset(x) do
               I(T',S_i):=I(T',S_i)∩[0,t_T-1];
           for T'∈Twriteset(x) do
               I(T',S_i):=I(T',S_i)∩[t_T+1,+∞[;
           E(x):=t_T;
           Twriteset(x):=Twriteset(x)-{T};
           swap(x,copy(x,T)));
      delete(I(T,S_i),Oreadset(T,S_i),Owriteset(T,S_i));
      Tlivingset(S_i):=Tlivingset(S_i)-{T}
    );
```

Remark. This critical section may be shortened by decomposing the procedure in many critical sections in order to exclude selectively the processing of reads and of prewrites depending on the part of the adjustment which is made.

The protocol of distributed certification

The certification of a transaction T is triggered at the end of its read/prewrite phase by the broadcasting of the message **certify**(T,$I_c(T)$) to all sites managing objects used by T. The processing of this message on each site starts the certification protocol. Let us notice that it is necessary to insure that all concerned sites process the certifications in the **same** order in case of simultaneous ends of several conflicting transactions.

When communication between sites uses an unique broadcasting medium (Ethernet for instance) reception order of messages is the same on all sites. In this case, it is sufficient that certification messages are processed in the order of their reception to fulfil the previous condition. On the other hand, when the sites are linked together by a general network, the global ordering of certification messages must be done using a special mechanism like synchronization by timestamps [Lamport 78] or by circulating token [LeLann 78] not described here.

In the proposed protocol, no site plays a privileged role. On each site S_i concerned by the certification of T, protocol consists in 3 phases:

- **a proposal phase** when S_i broadcasts its $I(T,S_i)$ to all other sites and waits for their proposals.
- **a choice phase** when S_i computes $I(T)=\bigcap_j I(T,S_j)$ which will thus be the same on all sites, and then applies the serialization criterion (cf. proposition 1): if $I(T)=\emptyset$ then T is rejected, otherwise a timestamp value t_T is chosen in I(T) to be associated to T (cf. proposition 2). The protocol must ensure that all the sites will choose the **same** value t_T. Choice of $t_T=UI(T)$ fulfils this condition ,for instance. Other choices will be discussed later.
- **an adjustment phase** for the intervals of other living transactions T' which includes also the validation of T.

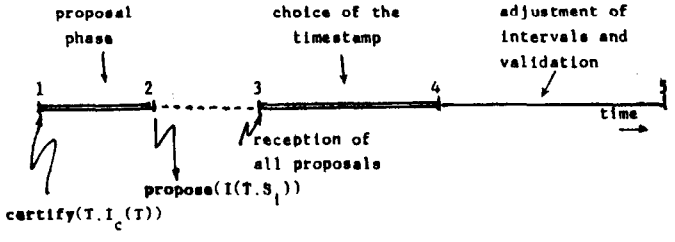


Figure. The different local steps of the certification.

Between the instants (1) and (4) the site S_i may process read and prewrite messages issued by other transactions but no other certification message. An optimization would allow the simultaneous processing of transactions'certifications which did not conflict between them on a given site.

The processing of the message **certify**$(T,I_c(T))$ received by site S_i is described below. Each site S_i is assumed to know all the other ones involved in certification of T. the critical section is bracketed with '<' and '>'.

```
certify (T,I_c(T))
    =(I(T,S_i):=I_c(T)∩I(T,S_i);
    broadcast-message(propose(T,I(T,S_i)));
    wait for the proposals from S_1,S_2,...,S_n;
                    •
                    •
    {when all proposals have been received}
    I(T):=∩_j I(T,S_j);
    if I(T)=∅
        then reject(T)
        else (choose(t_T);
            <adjust(T,t_T)>));
```

This protocol may be improved by not letting wait a site which has detected that $I(T,S_i)=\emptyset$, i.e. T must be rejected. Later on, all other sites will take the same decision in their choice phase.

Choice of the certification timestamp

During the certification of T, each site S_i must apply an identical strategy for choosing t_T in I(T) , for instance $t_T=LI(T)$. This choice may have identifiable effects on other living transactions T' in conflict with T. If we note T'_R those which have only read objects and T'_P those which have prewritten, the value t_T will truncate the intervals of T'_P on the left and those of T'_R on the right.

Choosing $t_T=LI(T)$ leaves both the greatest interval for each T'_P and the smallest for each T'_R. This tends to decrease risk of rejection for transactions T'_P and simultaneously increase that for T'_R. Choosing $t_T=UI(T)$ would have the reverse effect.

Such a definite effect can be considered as a forward control. With more information about living transactions, other choices could be devised for minimizing the number of rejections for instance.

Starvation

The method may be adapted to avoid starvation. A transaction which wants to avoid starvation must broadcast to all sites a **prioritary-certify** message at start time. On each site, processing of this message will start a protection phase ending with the certification of T. During this protection phase all certifications -prioritary or not - will be delayed; the reads and prewrites will be done as usual.

In other words, the method forces all other transactions to be certified after the complete execution of T. Therefore only dependencies of type $T_k \xrightarrow{*} T$ will be set during T's life, since dependencies of type $T \longrightarrow T_k$ are only possible if T_k is certified after the beginning of T, which has been avoided during the protection phase. Consequently no circuit will occur in G^*_T and T will be certified sucessfully.

5. CONCLUSION

By classifying various CCMs according to the type of dependencies they take into account, we have been able to make their limitations clear. Such an approach shows that there exists a continuum of methods between continuous control and control by

certification.

In our method, use of intervals of timestamps allows us to summarize dependencies in $G^*_{T_*}$ without washing out those of type $T \longrightarrow T_i$. It becomes then possible to accept a chronological order of certifications different from the serialization order, which avoids arbitrary rejections.

Since the intervals of timestamps are broken into locally managed intervals on each site, without needing any synchronization messages before the certification step, our method is particularly well suited to a distributed environment.

REFERENCES

[Badal 79] Badal D.Z.
Correctness of concurrency control and implications in distributed databases.
Proc. COMPSAC 79, Chicago, 1979.

[Bayer 82] Bayer R., Elhardt K., Heigert J., Reiser A.
Dynamic timestamp allocation for transactions in database systems.
Proc. of the 2nd International Symposium on Distributed Data Bases, Berlin, sept.1982.

[Bernstein 81] Bernstein Ph., Goodman N.
Concurrency control in distributed database systems.
Computing Surveys, vol. 13, n 2, june 1981.

[Bhargava 82] Bhargava B.
Resiliency features of the optimistic concurrency control approach for distributed database systems.
Proc. 2nd Symposium on Reliability in Distributed Software and DBS. Pittsburgh, july 1982.

[Cornafion 81] Cornafion, nom collectif
Systèmes informatiques répartis: concepts et techniques.
Ed. Dunod, 1981.

[Dewitt 80] Dewitt D.J., Wilkinson W.K.
Database concurrency control in local broadcast networks.
Computer Sciences TR 396, Univ. of Wisconsin, Madison, august 1980.

[Eswaran 76] Eswaran K.P., Gray J.N., Lorie R.A., Traiger I.L.
The notions of consistency and predicate locks in a database system.
Com. ACM 19,11, nov. 1976, pp 624-633.

[Haerder 82] Haerder T.
Observations on optimistic concurrency control schemas.
RR 3645, IBM Research Laboratory, San Jose, oct. 1982.

[Kung 81] Kung H.T., Robinson J.T.
On optimistic methods for concurrency control.
ACM Transactions on Database Systems, vol 6, n 2, june 1981, pp 213-226.

[Lamport 78] Lamport L.
Time, clocks and the ordering of events in a distributed system.
Com. ACM, vol 21, 7, july 1978.

[Lausen 82] Lausen G.
Concurrency control in database systems: a step towards the integration of optimistic methods and locking.
Proc. ACM Conf., Dallas, Oct. 1982.

[Le Lann 78] Le Lann G.
Algorithms for distributed data sharing systems which use tickets.
Proc. of the 3rd workshop on Distributed Data Management and Computer Network, Berkeley, aug. 1978.

[Papadimitriou 79] Papadimitriou C.H.
Serializability of concurrent updates.
J. ACM 26,4, oct. 1979, pp 631-653.

[Reed 78] Reed D.P.
Naming and synchronization in a decentralized computer system.
Ph.D dissertation, Dept. of Electrical Engineering, MIT Cambridge, Mass, sept. 1978.

[Rosenkrantz 78] Rosenkrantz D.J., Stearns R.E., Lewis P.M.
System level concurrency control for distributed database system.
ACM Transactions on Database Systems, vol 3, 2, june 1978, pp 178-198.

[Schlageter 81] Schlageter G.
Optimistic methods for concurrency control in distributed database systems.
Proc. 7th Conf. on Very Large Databases, Cannes, sept. 81

[Thomas 79] Thomas R.H.
A solution to the concurrency control problem for multiple copy databases.
Proc. 1978 COMPCON Conf. (IEEE) New-York.

[Traiger 82] Traiger I., Gray J., Galtieri, Lindsay B.
Transactions and consistency in distributed database systems.
ACM Transactions on Database Systems, vol. 7, n 3, sept. 82.

[Viemont 82] Viemont Y., Gardarin G.
A distributed concurrency control algorithm based on transaction commit ordering.
Proc. FTCS - 12, Los Angeles, 1982.