Mixed concurrency control : Dealing with heterogeneity in distributed database systems

J.F. Pons and J.F. Vilarem
Centre de Recherche en Informatique de Montpellier
(Université de Montpellier II / UA-CNRS)
860, rue de Saint-Priest
34100 Montpellier, France
EARN address : CRIM@FRMOP11.BITNET

Abstract

A mixed concurrency control, which allows the two techniques - two phase locking and certification - to coexist together in the same distributed database system, proves to be advantageous in a number of situations : interconnected databases, static or dynamic heterogeneity of transactions or objects. In this paper we propose a method which seems well adapted to the majority of the forms of heterogeneity, by using dynamic calculation of a serialization order and concurrent control of all types of transactions.

1. Introduction

The two classical concurrency control approaches are the pessimistic approach, based on two phase locking (2PL), well adapted where conflicts are quite probable, and the optimistic approach whose efficiency relies upon the scarcity of conflicts. A more recent approach, mixed or heterogeneous, enables the use of the benefits of the two previous approaches. As a result of these two classic methods being together in the same system, a number of advantages occur in different situations :

- In the case of interconnected databases with static heterogeneity, a mixed control gives uniform access to the different parts of the system which apply distinct methods.
- In the case of dynamic heterogeneity, the strategy used by each site can vary in the course of time between a pessimistic technique and an optimistic one. We can therefore best exploit the dynamism of the system.
- When the transactions are typed (long/short, reading/updating), we increase the parallelism by allocating an adapted technique to each type of transaction.
- Finally, when facing the heterogeneity in terms of objects rather than in terms of transactions, we could select the right compromise between locking and rollback. That is, for conflicting objects we would use 2PL, whereas an optimistic technique would be used for less sensitive objects.

In the literature, numerous studies connected to this subject can be found, but to our knowledge there is no existing mixed method which is satisfactory in a distributed environment. The certification solutions proposed come up against the following difficulties :

- To ensure the compatibility between heterogeneous local orders, which causes useless rejection.
- The global distributed certification problem which leads to the lowering of parallelism.

In order to handle these problems more effectively, we propose in this study a general method of distributed mixed control, whose characteristics are as follows :

- No local total order is imposed. The local constraints consistent with conflicts between transactions are taken into account dynamically.
- All possible conflicts between different types of transactions are locally handled. (These types are either transmitted by the objects on the site, or inherited from transactions).
- Starting from the local orders, the method constructs a global order in a non centralized way.
- In transferring the global order into the local commitment phases - executed in parallel - the set of

This work was supported in part by the C^3 project of the Centre National de la Recherche Scientifique.

all committed transactions does not have to be explicitly considered.

This paper is organized as follows : In section 2 we present the three approaches to concurrency control and the problems caused by their distribution. Section 3 studies the different heterogeneous situations and proposes some adapted solutions. The rules of mixed concurrency control are stated in section 4 and are then applied in section 5.

2. Background

2.1. Dependency graph

Operations

The transaction model we are using was taken from [Bernstein 81]. The transaction has a private workspace. The operations applied to object x are as follows:

- **read(x)** returns the value of a copy of x, if it already exists in the workspace; otherwise it returns the original value retrieved from the database.
- **prewrite(x)** expresses the intention of writing an object x and transfers the value from the transaction's workspace into **secure storage**. Once a prewrite is accepted, the corresponding write must not be rejected.
- **write(x)** effectively executes the transfer from secure storage into the database. A transaction which has finished, after a possible write phase, or commit phase, is said to be committed.

From these primitive operations, semantically more complex operations could be constructed.

Conflicts and dependencies

Two transactions conflict if one tries to read (resp. prewrite) an object already prewritten (resp. read or prewritten) by the other. A **conflict** between two transactions induces a constraint upon their respective **serialization order** (SO), represented by a **dependency relation** [Papadimitriou 79].

Dependency graph

The execution of a set of transactions may be expressed by a dependency directed graph G, whose vertices are transactions and arcs are dependencies induced by their conflicts. This graph is the privileged theoretical tool for the study of concurrency control. Let G* be the subgraph of G restricted to all committed transactions with their dependencies. All methods try to guarantee serializability by keeping G* acyclic.

2.2. Concurrency control for distributed databases

Pessimistic approach :

Two phase locking [Eswaran 76] [Traiger 82] is widely used for historical reasons and because it is well suited to a distributed environment [Mohan 84]. It is the most typical pessimistic approach : loss of parallelism is only justified when conflicts are frequent. The 2PL method tries to construct a global SO, starting with the local orders (one per object) induced by the locking policy. When these orders are inconsistent, 2PL leads to a deadlock the prevention of which implies the predeclaration of used objects, and the avoidance of which relies on the existence of a global timestamp order, and the detection and resolution of which require a search in G [Elmagarmid 86].

Optimistic approach (or using certification) [Kung 81] [Ceri 82] [Boksenbaum 84] [Sinha 85]

These methods exclude any kind of synchronization involving read and prewrite operations; the control is delayed until the certification phase, following the access phase and preceding a possible commit phase. As these methods only use backup, their efficiency depends on a low conflict rate, contrary to pessimistic methods. Therefore they are said to be optimistic: rejection is necessary only in the worst circumstances. The major drawback of this approach lies in the difficulty of its distribution. Indeed the increase of parallelism during the access phase may well be illusory if, in order to avoid inconsistencies, the method induces an increased wait by imposing that transaction certifications either be executed in mutual exclusion or in the same order on all sites.

"Concurrent certification", where several transactions are running their certification/commitment phase on the same site relies upon:

1) Dividing the certification/commitment phase into three distinct a priori mutually exclusive phases, namely: the **local certification**, related to the control of the objects located on the site, the **global certification**, executing the global control using the local control results, and the **local commitment-reject** phase.

2) Taking into account the locally certified (or locally controlled) transactions on the site, during the local certification phase. In [Boksenbaum 85] it is shown that assimilating a locally certified transaction with a committed one is not enough to guarantee consistency when local certification phases are executed in different orders on the sites. On the contrary, as emphasized in [Schlageter 82] and in [Lai 84], handling local certification phases in the same global order - used as the SO - ensures consistency.

However traditional mechanisms, based on timestamps or circulating tokens which ensure such a global order, tend to reduce parallelism. In practice, the very restrictive "same order" hypothesis has to be removed. Proposed solutions are therefore as follows:

- To forbid concurrent certifications by rejecting a transaction in its local certification as soon as there is a conflicting transaction previously certified on the site.
- To act so that concurrent certifications of conflicting transactions, when executed in different orders, produce a deadlock [Ceri 82].
- To use a dynamic technique, based on intervals of timestamps [Bayer 82] in order to translate dependencies between transactions [Boksenbaum 87].

Integrated approach

The aim of this recent idea is to make the previous approaches coexist in the same system or method. The different forms of this "cohabitation" are presented in the next section. Historically, the basic ideas have been defined for centralized systems in the following ways :

- In [Boral 84] the dependency graph is effectively maintained in order to obtain a common "2PL-certification" SO. Each type of conflict (read/write or write/write) is resolved by one of the two techniques (2PL or optimistic) either during the read phase, or during the certification phase.
- In [Lausen 82] locking transactions are integrated into the basic optimistic method of [Kung 81].

Distributed and integrated methods are rare. The difficulties when adapting to a distributed environment lie in the following facts :

- The use of the dependency graph presents a major drawback : Each site has a local knowledge of the graph, and a global decision produces a heavy load in communication between sites.
- The distribution of Lausen's method first requires that the concurrent certification problem be correctly answered. This is attempted in [Sheth 86]; however, in this method, the previous problem does not seem to be solved : Concurrent certifications of conflicting transactions lead to inconsistencies when executed on different sites and in reverse orders.

Integration of locking transactions in a distributed certification method - based on the acceptance of locking transactions having reached their maximum locking point - is covered in the method proposed in [Pons 88].

3. Heterogeneity

3.1. Static heterogeneity

We are interested in interconnected databases, which historically differ by their transaction management [Gligor 85]. The aim of a mixed control is therefore to allow a uniform access to the different sites, which each apply a distinct method. Within this framework, we regard a transaction as a collection of local subtransactions - one per site - scheduled by a coordinating site. The global control of a transaction, as in the "superdatabases" from [Pu 87], supposes a unique protocol - such as two phase commit - insuring global atomicity, and an explicit local SO for each subtransaction.

The question is to guarantee a global serializability of transactions from these local SOs. The resulting answers are generally unsatisfactory or even incomplete. Two examples are given below.

1) [Pu 87] proposes the following certification algorithm: Firstly, a local phase exhibits a timestamp materializing a local and **total** SO for each subtransaction. Secondly, in a global and centralized certification phase, a global SO is constructed as the product of all the SOs of the committed transactions. A transaction is committed if its tuple of local SOs can find a place in the global SO, otherwise it is rejected. The method is correct because - taking into account **all** the dependencies between the unique committing transaction and all the ones already committed - it can detect any cycle in the global graph. The method's drawbacks include a) the difficulty to forget old committed transactions, b) a bottleneck due to the mutually exclusive centralized global control and c) useless rejection due to unnecessary local ordering of non conflicting subtransactions.

2) [Elmagarmid 87] proposes to enhance concurrency over the previous algorithm, in releasing the local transactions from the global centralized control. However, in spite of this correct improvement, the method, using conflict sets, takes into account the only direct dependencies between the committing transaction and the committed ones. Thus it forgets to control the transitive dependencies; so it forgets a cycle involving the committing transaction and two or more committed transactions.

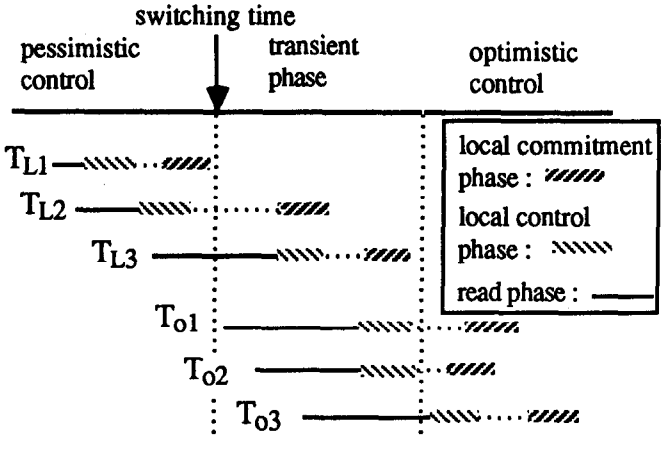
In the general proposal of section 4, we present a new distributed and mixed concurrency control applying to static heterogeneity : Instead of a total SO leading to global useless rejection, the local phase calculates the strongest local constraints affecting a subtransaction ; then these are sent to the coordinating site. Globally, but not in a centralized manner, we construct a global SO from the

local constraints. This common SO is then carried out upon the sites during local commitment phases.

3.2. Dynamic heterogeneity

In this approach, the concurrency control mechanism should be a part of the software able to adapt to a changing environment. Therefore the hypothesis of a static partition between optimistic and locking sites should be removed. As an example, in a dynamic hypothesis, if a site gets a heavy workload during the day then it must use a pessimistic algorithm ; otherwise, with decreasing activity, this site may change to an optimistic algorithm during the night.

In general, in order to best exploit the dynamism of the system, each site would use a time variant strategy which is able to switch between pessimistic and optimistic mechanisms. The transition will occur on a site at a given instant according to the degree of transaction interference [Badal 84] [Sheth 86]. The problem with this local transition concerns the non committed subtransactions living on the site at the switching time, as shown in the next example :



Some rough solutions consist of local rejection of the subtransactions concerned (as T_{L2} or T_{L3} in the above example), or waiting until they are committed before starting new subtransactions on the site (as the T_{oi}s). A more flexible solution authorizes the concurrent execution of heterogeneous subtransactions during a transient phase. In this preferred solution, on the one hand a common SO is needed on a site in order to globally control the heterogeneity, on the other hand mixed conflicts have to be locally managed in order to correctly process the transient phase. In the above proposed example, in addition to the conflicts related to committed transactions as T_{L1}, the mixed conflicts between T_{L2},T_{L3},T_{o1} and T_{o2} have to be controlled. Such a solution therefore appears as an extension of the control we defined in § 3.1. Indeed, when this solution is applied to dynamic heterogeneity, mixed conflicts are handled during the local control phase as follows : Optimistic subtransactions, by using additional control related to non committed locking subtransactions ; locking subtransactions, by using additional waiting related to locally controlled optimistic subtransactions.

However, this time variant strategy, though dynamic, implies a unique type for each subtransaction to be executed on the same site at a given instant. This is not realistic.

3.3. Per transaction heterogeneity

In a dual approach, the degree of interference is not measured in terms of conflict rate on the site, but in terms of probability of transaction conflict. Heterogeneity is then related to transactions, which are typed either optimistic or locking, according to the principles of [Lausen 82] and [Boral 84]. For example, a long updating transaction with a great probability of conflicts will be processed using 2PL, while an optimistic control is better suited for a short reading transaction. At its beginning, a type is allocated to a transaction either in a static way by using the type and the frequency of conflicts predicted by the transaction, or in a quasi dynamic way by incorporating some run-time parameters connected to the dynamism of the system [Boral 84].

Dealing with this heterogeneity, a method must ensure a global distributed control, and must correctly handle mixed conflicts between subtransactions. Moreover, sufficient parallelism has to be allowed by concurrently controlling several transactions on a site. Transaction heterogeneity in some way generalizes the dynamic site heterogeneity of § 3.2 : Whereas in § 3.2 mixed conflicts must only be processed in the transient phase, subtransactions of different types are now present at any given instant on the site.

The principles of this mixed approach, further detailed in section 4, are the following : At first, existing applications which are based on homogeneous control, must behave in the same way when changing to a mixed control. Secondly, an optimistic transaction must not wait during its read phase and may be rejected in its certification phase. Finally, a locking transaction reaching its maximum locking point (mlp) must commit. Otherwise locking and optimism would make no sense.

As in § 3.2, the overhead caused by mixed conflicts consists of extra waiting for locking subtransactions, and

also of additional - possibly rejecting - control for optimistic subtransactions. Such a method will be useful in systems where the predictions of conflicts occurring on a site are easy, and the choice of each transaction type is unquestionable.

However, in general distributed systems, such predictions are not easy, and errors are expensive. Thus, following the conclusions of [Kung 81], [Lausen 82] and [Herlihy 86], in order to obtain the right compromise between waiting and rollback, 2PL or optimistic techniques are most likely to be useful when applied to individual objects rather than to transactions or entire systems.

3.4. Heterogeneity of objects

Let us first show, with an example, that efficiency of locking or optimistic techniques depends on the distribution of the conflicts in a complex manner.

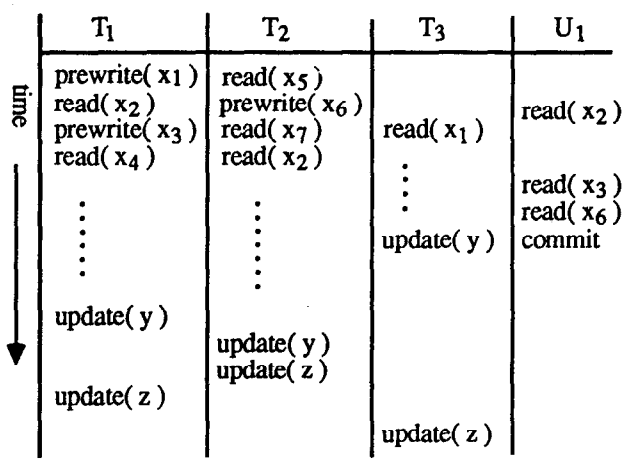
Let $T_1,...T_n$ and $U_1,...U_p$ be transactions such that:

$\forall i$ T_i reads or writes some objects $x_{ik} \in X$, then sequentially updates the objects y and z:

$y := y + \sum_k x_{ik}$; $z := z + \sum_k x_{ik}$.

$\forall j$ U_j reads objects in X.

It is supposed that all objects in X are "quiet" (with a low rate of conflicts), and that y and z are "sensitive" (with a high rate of conflicts). A concurrent execution of transactions T_i, U_j could be:



- If locking is the policy used by the whole system, then U_1 is prevented from access to objects x_{ik} previously written by non committed transactions T_1 and T_2. This causes **useless waiting** since U_1 (in general U_j) could commit before the transactions T_i, by preceding all of them in the SO.

- If an optimistic method is used by the whole system, the concurrent processing, by the transactions T_i of y and z updates, leads to rejection which could be **prevented using locks**, since y and z updates are processed in the same order. In this example any of the optimistic commitment of T_1, T_2 or T_3 would lead to the rejection of the other two.

- If locking is used for the transactions T_i, and an optimistic method is preferred for each U_j, then parallelism is reduced for the transactions T_i when having access to the x_{ik} objects. In this example, T_3 is blocked when reading x_1, until T_1 releases its exclusive lock.

- Finally, if the system uses locking on objects y and z in order to **prevent** high risk conflicts, and if it uses an optimistic method for the objects in X in order to **control** low risk conflicts, then it exploits the efficiency of each technique.

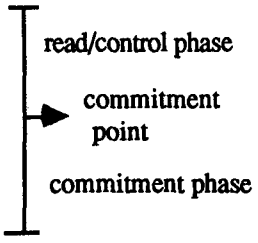
In this last case, the objects are partitioned between locking - L type - and optimistic - o type - objects. We again come across the description of static heterogeneity, but with a smaller granularity : an L type site (resp. an o type site) is here replaced by the set of objects of the same type on a site. Thus a subtransaction T_s executing on a site s, is seen as a pair of subtransactions (T_{os}, T_{Ls}). As the local partition is static, no mixed conflicts have to be handled, so the control of static heterogeneity can be extended here.

From a more general point of view, objects may dynamically change their type. In a similar way, the control used in dynamic heterogeneity of sites could be extended to this last situation. The switching of an object from one type to another needs a transient phase in order to control all the conflicts between different components of subtransactions.

4. Proposals for a distributed mixed method

4.1. Transaction model

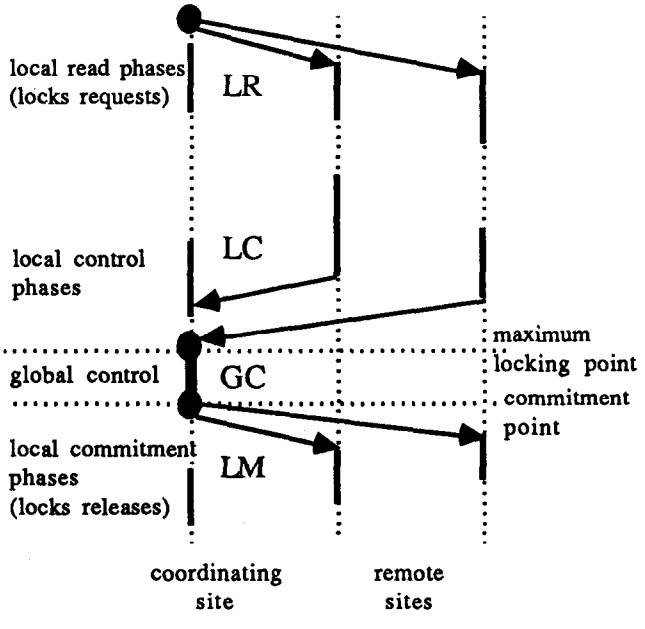
With respect to atomicity, we model the execution of a transaction with two phases: During the first one, the transaction reads objects, announces its write intentions with prewrites, and controls the correctness of its operations upon the database. During this read/control phase a transaction runs using a private workspace, its prewrites are performed within a secure storage. The second commitment phase is used to make the effects of the transaction on the database permanent.



. Before the commitment point, the transaction has no effect on the database.

. After this point, the transaction atomicity depends on the commitment phase atomicity, which is permitted by the use of secure storage for prewrites.

In order to distribute this model, the transaction has a coordinating site, which coordinates parallel executions of its subtransactions on different sites. The local read/control phase is divided into a read phase according to the subtransaction's type, and a local control phase which calculates the strongest local constraints bearing on the subtransaction and which sends this result to the coordinating site. When all these messages are received a synchronization point is reached. It is used both to execute a global control phase on the coordinating site, and to ensure that a transaction could not appear at the same time already committed on a site as well as in its read/control phase on another site. The results of the global control are broadcasted to the concerned sites which then execute a local commitment phase. The following figure illustrates this transaction model:



Distributed integrated transaction model

Remarks :

- In general a subtransaction T_s, running on the site s, has a single type, either inherited from the transaction or transmitted by the site S, depending on the concerned heterogeneity. However, in the case of the heterogeneity of objects, T_s is made of two components (T_{os} , T_{Ls}); the local controls of T_{os} and T_{Ls} and their integration are processed during the same local control phase.
- It is noted that a sequential execution of the read phase, needed for an interactive transaction, is a particular case for the parallel model presented here.
- In the case of a local transaction which executes on a single site, the GC phase merges with the LC phase, thus releasing this kind of transaction from any global control.

4.2. A method which is independent of the dependency graph

4.2.1. Sequential control

Definition 1 : A serialization order (SO) is a total order among committed transactions which is consistent with the dependencies induced by their conflicts. If we note $T_1^* \rightarrow T_2^*$ such a dependency, then SO verifies :

$$\forall T_1^*, T_2^* \in G^* \quad T_1^* \rightarrow T_2^* \Rightarrow T_1^* \underset{SO}{<} T_2^* .$$

In general, concurrency control methods apply a serialization criterion, which controls whether or not a serialization order exists.

Theorem 1 : A serialization order exists **iff** G* is an acyclic digraph.

We have seen in section 2.2 that it is not realistic to manage this graph in a distributed environment. So most of the traditional methods control the existence of a SO, which could either be a priori defined (in timestamp ordering methods) or attempted to be established during access (in 2PL methods). In our method, as in [Boksenbaum 87], the SO is dynamically constructed without using the dependency graph.

First, we present a **sequential construction** of the SO, controlling and possibly committing a single transaction T at a given time. Let G_T^* be the extension of G* obtained in adding the vertex T and the set of arcs meaning T's conflicts with committed transactions. Let SO be the total order constructed by the method and related to the acyclic G*.

Definition 2 : An extension of SO induced by T is a total order SO_T defined on G_T^* and verifying:

$$\forall T_1^*, T_2^* \in G^* :$$

(restriction) $T_1^* \underset{SO}{<} T_2^* \Rightarrow T_1^* \underset{SO_T}{<} T_2^*$

(new arcs) $T_1^* \rightarrow T \Rightarrow T_1^* \underset{SO_T}{<} T$

$T \rightarrow T_2^* \Rightarrow T \underset{SO_T}{<} T_2^* .$

Proposition 2 : If SO_T exists then G^*_T is an acyclic digraph.

The aim of this sufficient condition is to allow the SO to be **incrementally** constructed, without searching for any cycle in G^*_T while controlling a transaction T. This construction consists in inserting T in SO, if possible. To do so, we express the strongest constraints bearing on T. The resulting serialization criterion comes from the following theorem:

Theorem 3 : Let $T^- = sup_{SO}(T^* \mid T^* \longrightarrow T)$, and $T^+ = inf_{SO}(T^* \mid T \longrightarrow T^*)$.
SO_T, extension of SO exists **iff** $T^- \underset{SO}{\leq} T^+$.

Proof : Let SO_T, extension of SO, exist. Let $T_1^*, T_2^* \in G^*$ such that $T_1^* \longrightarrow T$ and $T \longrightarrow T_2^*$. Suppose $T_1^* \underset{SO}{\geq} T_2^*$. Then, using definition 2 : $T_1^* \underset{SO_T}{\leq} T$, $T \underset{SO_T}{\leq} T_2^*$ and $T_1^* \underset{SO_T}{\geq} T_2^*$ which leads to a contradiction. Thus $\forall T_1^*, T_2^* \in G^*$: $T_1^* \longrightarrow T$ and $T \longrightarrow T_2^* \Rightarrow T_1^* \underset{SO}{\leq} T_2^*$ implies $T^- \underset{SO}{\leq} T^+$. Conversely, suppose SO is a serialization order and $T^- \underset{SO}{\leq} T^+$, then SO_T constructed by inserting T in SO in any place between T^- and T^+ is obviously an extension of SO induced by T. ❑

The advantages of the method are:

- It memorizes a total order, thus excluding the need of a global use of G*.
- It may forget old transactions. For example, let x be an object written by committed transactions such that $T_1^* \longrightarrow_x T_2^* \longrightarrow_x \ldots \longrightarrow_x T_n^*$. In SO we have $T_1^* < \ldots < T_n^*$, and, through a further read access from a transaction T, only $T_n^* \longrightarrow_x T$ is needed.
- Its natural distribution : when the database is partitioned, usually in sites, the control, is easily distributed. Locally, for each site s it calculates :
  $T_s^- = sup_{SO}(T^* \mid T^* \longrightarrow_x T ; x \in s)$, and
  $T_s^+ = inf_{SO}(T^* \mid T \longrightarrow_x T^* ; x \in s)$.
  The global control is : $sup_{SO}(T_s^- ; s \in$ sites$) < inf_{SO}(T_s^+ ; s \in$ sites$)$. It assumes that each part uses the same global order SO. This local knowledge is a result of the local commitment phases of successful globally controlled transactions.

In the next section we study a way of increasing the parallelism, through permitting many transactions to be concurrently controlled.

4.2.2. Concurrent control

The case of non conflicting transactions

Suppose there is a sequential control of two non conflicting transactions T_1 and T_2, and suppose T_1 is committed before T_2 is controlled. The sequential control constructs SO_{T1}, then, in order to control T_2 it calculates $inf_{SOT1}(T^* \mid T_2 \longrightarrow T^*)$ and $sup_{SOT1}(T^* \mid T^* \longrightarrow T_2)$. As SO_{T1} is an extension of SO, and T_2 has no conflict with T_1 we deduce that : $inf_{SO}(T^* \mid T_2 \longrightarrow T^*) = inf_{SOT1}(T^* \mid T_2 \longrightarrow T^*)$ (same applies to sup).
Hence , a sequential control behaves like a parallel one when constructing SO_{T1T2}. This parallelism applies to any number of non conflicting transactions.

The case of conflicting transactions

If conflicting T_1 and T_2 are controlled in parallel, one may construct a total order which is incompatible with the dependencies induced by their conflicts, thus leading to inconsistency. We show in section 2 that a medium exists between this inapplicable parallelism and the mutual exclusion of a sequential control. This medium relies upon the notion of a locally controlled or a locally certified transaction, i.e. a transaction which has successfully checked the local criterion $T_s^- < T_s^+$. Using this notion, the method must take into account the locally controlled (not yet committed) transactions. The mutually exclusive local phases are executed in parallel on different sites.

Considering this parallelism, we define the **principles of concurrent control** :

Let T_1 be a transaction running its local control on a site s including an object x concurrently used by a transaction T_2 :

- If T_2 has already been committed, the sequential control is then applied.
- If T_2 is locally controlled, we know the results of this phase, namely : $T_{2s}^- < T_{2s}^+$. If the dependency is $T_1 \longrightarrow_x T_2$ then we add the **fictitious arc** $T_1 \longrightarrow T_{2s}^-$; otherwise we add $T_{2s}^+ \longrightarrow T_1$. Next, we run the local control with this extra arc.

Theorem 4 : Using the principles of concurrent control, the method is correct.

Proof : Indeed, a successful concurrent control with two transactions whose dependency is $T_1 \longrightarrow_x T_2$, constructs an order SO_{T1T2} which is consistent firstly with G*, secondly with the arcs between T_1 or T_2 and committed transactions, and finally with either $T_1 \longrightarrow T_{2s}^-$ or $T_{1s}^+ \longrightarrow T_2$ (depending on the order of the local controls of T_1 and T_2 on the site s). Hence, in SO_{T1T2}, either $T_1 < T_{2s}^- < T_2$ or $T_1 < T_{1s}^+ < T_2$ stands, which proves this total order is consistent with $T_1 \longrightarrow_x T_2$.❑

Remarks :

- These principles apply to any number of transactions having any number of conflicts between them.
- The accuracy of the concurrent method does not depend on the order of the local controls. This interesting result gives a solution to the problem of concurrent certifications we have seen in § 2.2.
- Where $T_1 \rightarrow_x T_2$, T_1 has been locally controlled before T_2 on the site s including x, and T_{1s}^+ does not exist, the method will reject T_2 . During T_1's local control, an "improvement" consists of forcing T_1 to precede a committed transaction. This kind of "forward control", as defined in [Haerder 84] is obtained by adding a fictitious arc, at the risk of rejecting T_1 during its global control. Another way to avoid T_2's rejection is to suspend its local control until T_1 be committed or rejected.

4.2.3. Applying the method to the static heterogeneity of sites or objects

In this case, no mixed conflicts have to be handled.

- Let T_s be a locking subtransaction of T on a locking site s. It conflicts solely with locking subtransactions. Since $\{ T^* \mid T_s \rightarrow T^* \} = \{\}$ when 2PL is used, the local control is limited in this case to the calculation of T_s^-.

- Let T_s be an optimistic subtransaction of T on an optimistic site s. It conflicts solely with optimistic subtransactions. During T_s's local control phase, the controlled dependencies are:
$T_o^* \rightarrow T_s$ or $T_s \rightarrow T_o^*$ (the latter resulting from the effects of the commitment of T_o on T_s)
Also, with respect to locally controlled $T_o^\bullet$, $T_s \rightarrow T_o^\bullet$ or $T_o^\bullet \rightarrow T_s$ are taken into account by adding the corresponding fictitious arcs $T_s \rightarrow T_{os}^-$ or $T_{os}^+ \rightarrow T_s$, according to the principles of concurrent control.
The local control calculates T_s^- and T_s^+ . If $T_s^- < T_s^+$ then T_o is considered as locally controlled on the site s.

- In the case of heterogeneity of objects, a subtransaction T_s is made of two heterogeneous subtransactions T_{os} and T_{Ls} . The local control of T_s merges the local controls of T_{os} and T_{Ls}.

The global control of a transaction T, after having received the results of the local controls of all its subtransactions calculates $T^- = \sup_{SO}(T_s^- ; s \in \text{sites})$, and $T^+ = \inf_{SO}(T_s^+ ; s \in \text{sites})$. If $T^- \geq T^+$ then T is rejected; otherwise a global total order is constructed, where $T^- < T < T^+$. The effects of this global phase take place during local commitment-rejection phases.

4.2.4. Extending the method to mixed conflicts : per transaction heterogeneity

In this case a typed transaction T_o (resp. T_L) is made of subtransactions of the same type T_{os} (resp. T_{Ls}) running on different sites s. A principle of the method is to commit a locking transaction which has reached its mlp, otherwise locking would make no sense. To do so, we keep the following sufficient proposition invariant :

Proposition 5 : For each committed transaction T*, and for each locking transaction T_L which has reached its mlp :

(T* is not conflicting with T_L **or** $T^* \rightarrow T_L$)
$\Leftrightarrow (\{T^* \mid T_L \rightarrow T^*\} = \{\})$.

Consequently, T_L may be inserted into the constructed SO, and can then commit. The additional control, due to keeping this property invariant, is transferred to optimistic subtransactions.

Controlling an optimistic transaction T_o

The controlled dependencies are similar to those in section 4.2.3 applied to conflicts with committed or locally controlled transactions. Nevertheless, a special treatment is needed to keep proposition 5 invariant :

$T_L \rightarrow T_o$ or $T_L^\bullet \rightarrow T_o$ implies the rejection of T_o.

The local control on the site s calculates T_{os}^-, T_{os}^+ and $\{T_L \notin G^* \mid T_L \rightarrow_x T_o ; x \in \text{site s}\}$. If $T_{os}^- < T_{os}^+$ and $\{T_L \notin G^* \mid T_L \rightarrow_x T_o ; x \in \text{site s}\} = \{\}$ then T_o is considered as locally controlled on the site s.

The global control calculates T_o^-, T_o^+ and $\{T_L \notin G^* \mid T_L \rightarrow T_o\}$. If $T_o^- \geq T_o^+$ or $\{T_L \notin G^* \mid T_L \rightarrow T_o\} \neq \{\}$ then T_o is rejected; otherwise a global total order where $T_o^- < T_o < T_o^+$ is constructed. The effects of this global phase take place during local commitment-rejection phases.

Controlling a locking transaction T_L

If we consider the sequential method, a transaction T_L having reached its mlp is controlled with respect to committed transactions. Using the concurrent method, we must enforce additional control in order to consider locally controlled optimistic transactions. Bearing in mind that a locally controlled optimistic transaction could not be rejected, and that a locking transaction T_L, which has reached its mlp, would not be rejected, the dependencies controlled on the site s are:
- $T^* \rightarrow T_L$ (a consequence of the proposition 5 and of locking properties)
- With respect to a locally controlled optimistic subtransaction $T_o^\bullet$:
  - $T_o^\bullet \rightarrow T_L$: If T_{os}^+ exists, as a result of the local control of T_o, then $T_o^\bullet \rightarrow T_L$ is changed into

$T^+_{os} \rightarrow T_L$ according to the principles of the concurrent method. Otherwise, the local control of T_L is suspended until T_o is committed or rejected.

- $T_L \rightarrow T^{\bullet}_o$ cannot occur. Indeed, if T_L has used a conflicting object before the local control of T_o then T_o would have been rejected; otherwise, if T_o has been successfully controlled on the site s, it locks each prewritten object, thus preventing these dependencies.

The local control on site s is the calculation of T^-_s.

The global control of T, is limited to the calculation of T^- , and the construction of a SO where $T^- < T_L$. The effects of T_L on the database and the new SO are carried out during the local commitment phases.

4.2.5. Dynamic heterogeneity of sites

In this case T is made of subtransactions of different types. If the site s is in its transient phase, then the subtransaction is controlled according to the policy ruling the mixed conflicts. Otherwise, the site is in a single type phase, and we use the policy ruling the static heterogeneity. Globally we merge the different local controls.

In the case of dynamic heterogeneity of objects, each subtransaction T_S is made of two heterogeneous subtransactions T_{os}, T_{Ls} . First the local control of T_S merges the usual local controls of T_{os} and T_{Ls}, then the global control merges the different local controls.

Conclusion : Applying the sufficient serializability criterion from theorem 3, we construct a total order of committed transactions consistent with their conflicts, thus guaranteeing the accuracy of the method.

It is noted that the treatment of mixed conflicts balances the overload of the method between an additional waiting for the locking subtransactions and more rejection for the optimistic ones. In a previous paper [Pons 88], we proposed another strategy in which the additional burden was only born by optimistic subtransactions, which execute a forward control during their local phase. Locking transactions gained an advantage from this strategy, but a major drawback of this proposal was the useless rejection of optimistic transactions.

5. Implementation of the method

5.1. Timestamps and intervals

In order to forget graph G* and its partial order, a global and total order of committed transactions which is consistent with G* is computed. The rank of a committed transaction T* materializes through a positive numerical timestamp t(T*), computed during the global phase. As the timestamp order is a SO, it verifies :

$\forall T^*_1, T^*_2 \quad T^*_1 \rightarrow T^*_2 \Rightarrow \quad t(T^*_1) < t(T^*_2)$.

The serialization criterion obtained by checking the theorem 3 is as follows :

sup (t (T*) | T* → T) < inf (t (T*) | T → T*).

This criterion is naturally implemented by means of the timestamp interval technique. The global interval IG(T) associated with a transaction T is of the form [lower(T) ; upper(T)], where lower(T) = sup(t(T*) | T* → T) and upper(T) = inf(t(T*) | T → T*) represent the strongest constraints between T and already committed or concurrently controlled transactions. The serialization criterion then changes as follows : IG(T) must neither be empty nor reduced to one element, i.e. $|IG(T)| > 1$. Any timestamp in IG(T) can express T's rank in the constructed SO. The distribution of the method is expressed by the local intervals I(T,S), each of them representing the strongest local constraints affecting the subtransaction of T on site S. The calculation of the global interval is then :

$$IG(T) := \bigcap_{S \in \text{ sites used by T}} I(T,S)$$

Local interval maintenance relies upon the principles of the concurrent control. Let us call "living" a not yet locally controlled subtransaction. Let T_1 be living, $T^{\bullet}_2$ be locally controlled and T^*_3 be committed, all of them in conflict on site S of x.

-A dependency of the form $T_1 \rightarrow_x T^*_3$
(resp. $T^*_3 \rightarrow_x T_1$) must be translated into :
upper($I(T_1, S)$) ≤ t(T^*_3)
(resp. lower(I(T_1 , S)) ≥ t(T^*_3)).

-A dependency of the form $T_1 \rightarrow_x T^{\bullet}_2$
(resp. $T^{\bullet}_2 \rightarrow_x T_1$) must be translated into :
upper($I(T_1, S)$) ≤ lower(I($T^{\bullet}_2$, S))
(resp. lower(I(T_1 , S)) ≥ upper(I($T^{\bullet}_2$, S))).

Therefore, the local interval $I(T_1, S)$, initialized to [0 ;+ ∞[(no conflict), will have to be truncated to the left (resp. to the right) during the different phases of T_1 on site S. Contrary to most optimistic methods this technique permits the presence of "**old readers**" such as T_1 ; this is achieved by the handling of $T_1 \rightarrow T^*_3$ or $T_1 \rightarrow T^{\bullet}_2$ dependencies. During the local commitment phase, the global order is carried out on the objects used, by means of the timestamps W(x) and R(x), from the most "recent" - in the SO - transactions that have written or read x. We notice that, in the case of a locking transaction T_L having reached

its mlp, the proposition 4 stated in § 4.2.5 implies : $IG(T_L)$ and $\forall S$ $I(T_L,S)$ are of the form $[a , +\infty [$.

5.2. Detailed model

5.2.1. Objects and transactions

In order to simplify the notations, our model assumes that only one object is managed on a site, but it easily applies to the most general case : The set of objects managed by the site replaces the single x.

Therefore, we have $IG(T) = \bigcap_{x \in \text{ objects used by T}} I(T,x)$.

The "object-site" model is composed of :

- A global name x.
- A type - locking or optimistic - which is transmitted, in the case of the heterogeneity of objects, to the subtransaction when it first had access to the "object-site". In the case of the heterogeneity of transactions, the subtransaction's type is inherited from the transaction.
- Data structures related to the accesses : The value of x, the timestamps W(x) and R(x), the shared and exclusive locks with their queues.
- Data structures required by the local control phase : The name T, the type, the status - living or locally controlled -, the access type - read or write -, and its local interval I(T,x) are managed for each non-committed subtransaction which had access to the object.

The transaction model we use has a coordinating site and several remote object-sites ; it corresponds to the integrated figure of § 4.1, and is composed of the following phases : the local read phases LR(T,x), the local control phases LC(T,x), the global control phase GC(T,x), and the local commitment or rejection phases LM(T,x).

LR(T,x) phases : The control of locking subtransactions is limited to the acquisition of the locks ; management of their intervals is not required. Concerning optimistic subtransactions, we will see later on that only read access requires an update of the local intervals.

LC(T,x) phases : Each phase includes :

- Controls connected with committed transactions : Strongest constraints between T and committed transactions are handled by truncating I(T,x) to the left with respect to W(x) or R(x) according to T's access.
- Controls connected with living or locally controlled subtransactions :

If optimistic T sees a locking and non committed T_L (or T_L^*) such that $T_L \rightarrow T$ or $T_L^* \rightarrow T$, then T must be rejected by setting I(T,x) to the empty interval; other conflicts with locally controlled subtransactions are taken into account by truncating I(T,x).

If locking T sees an optimistic locally controlled T_O^* such that $T_O^* \rightarrow T$, then :

If $I(T_O^*, x)$ is bounded then I(T,x) is truncated to the left according to upper($I(T_O^*, x)$) ; otherwise - upper($I(T_O^*, x)$) = $+\infty$ - LC(T,x) is suspended until T_O is rejected or committed . Then I(T,x) may be truncated to the left according to $t(T_O^*)$).

At the end of this phase, if $|I(T,x)| > 1$ then T is considered as locally controlled (let us recall that x is locked when prewritten by an optimistic locally controlled transaction).

GC(T) phase : The control is processed on the coordinating site, after receiving all the I(T,x) from the concerned sites. The global decision leads to a rejection or a calculation of a timestamp in IG(T). This result is then broadcasted to all sites concerned.

LM(T,x) phases : During this phase the following actions take place : In case of commitment, the timestamps W(x) and R(x) are updated and write operations are performed, the intervals of old optimistic readers - T_O such that $T_O \rightarrow T^*$ - are truncated to the right. In any case, commitment or rejection, locks are released and local data structures related to T are deleted.

Parallelism on the site

Each of the LC or LM phases, as well as each read or prewrite operation of the LR phase is considered a priori as atomic. Since critical data structures are involved, each phase or operation of a transaction must be executed in mutual exclusion of any other phase or operation of a different transaction. We notice that a more precise study would allow increased parallelism on the site by dividing LC and LM phases. For example, a site may execute in parallel the LM phase of a writing transaction and the prewrites - in LR phases - of another one ; on the contrary, read operations must not interfere with an LM phase. The global GC phase, which only depends on a fixed set of intervals, and does not modify any of the critical data structures, may be processed in parallel with any other phase or operation.

5.2.2. Conflicts and control

In the following, each dependency is labelled with the related conflict.

Write conflicts (PP or WP) The processing of these conflicts is taken from [Pons 86] : When timestamps are used to express the SO, it is possible to constrain the

timestamped write operations, and thus the transactions, to conform to the SO. This is done by forgetting a "late" write operation : If $t(T_1) < t(T_2)$, and if T_1 writes after T_2, then T_1's write is ignored, providing that the concerned read operations are involved (**Thomas' rule**).Using a common SO for all the transactions, this permits the existence of "**old writers**" whose serialization order is different from their writing order in the local commitment phase. This application includes the conflicts between locking transactions.

Implementation : The PP conflicts between living or locally controlled transactions are not considered. Thus, write locks only conflict with read locks, and no deadlock can result from PP conflicts. During the local certification of T which has prewritten x, the method considers the dependencies $T_k^* \xrightarrow{WP}_x T$ related to all the committed transactions which have written x, using the truncation to the left : $I(T,x) := I(T,x) \cap [\, W(x) ; +\infty [$. During the LM(T,x) phase, if $t(T) > W(x)$ then T effectively writes x and $W(x) := t(T)$; otherwise, the writing is ignored.

Remark : This is not a strict application of Thomas' rule. Using the strict application, the method would ignore the WP conflicts too. Thus it would put old transactions at an advantage, nevertheless it would lead to systematically ignoring acceptable write operations.

Read-write conflicts (RP or WR or RW)

- Between living transactions : The only executed control uses locks. It takes place during the LR phase for locking transactions.
- Between a living transaction T and a committed transaction T_k^* : If T has prewritten x, $T_k^* \xrightarrow{RP}_x T$ is considered using a truncation to the left (as in WP conflicts) during the LC phase; otherwise, if T has read x then :
  - When T is optimistic, $T_k^* \xrightarrow{WR}_x T$ is controlled using a truncation to the left during the LR phase. Furthermore, when a writing transaction T_k is committed after T has read x, the dependency $T \xrightarrow{RW}_x T_k^*$ is carried back to T using the truncation to the right $I(T,x) := I(T,x) \cap [\, 0 ; t(T_k^*)\,]$ during the $LM(T_k,x)$ phase. This technique controls old optimistic readers, without necessarily rejecting them.
  - When T is locking, the control prevents $T \xrightarrow{RP}_x T_k^*$ dependencies. If T_k is a locking transaction, such a conflict is prevented by the locking policy ; otherwise, if T_k is optimistic, this conflict cannot occur, either because T_k has been rejected during its LC phase, or T_k has been successfully controlled, and T is blocked until T_k is committed or rejected (§ 4.2.5).
- Between certifying transactions : During its LC phase, a transaction T (noted T_L or T_o), which has ended its LR phase, sees that a transaction $T^\bullet$ (noted $T_L^\bullet$ or $T_o^\bullet$) is already locally controlled, but not yet committed.
  - $T_L \xrightarrow{RP} T_o^\bullet$ cannot occur.
  - $T_L \xrightarrow{RP} T_L^\bullet$ and $T_L^\bullet \xrightarrow{RP} T_L$ are prevented using locking policy.
  - $T_L^\bullet \xrightarrow{RP} T_o$ causes T_o to be rejected.
  - $T_o^\bullet \xrightarrow{RP} T_L$ and $T_o^\bullet \xrightarrow{RP} T_o$ are controlled through a truncation to the left of the local interval of T_L or T_o referring to the upper boundary of $I(T_o^\bullet, x)$:
    $I(T,x) := I(T,x) \cap [\, \text{upper}(I(T_o^\bullet, x)) ; +\infty [$. The particular case where $I(T_o^\bullet, x)$ is unbounded leads to the rejection of T_o or to the delay of the local control of T_L until $T_o^\bullet$ is committed or rejected.
  - $T_o \xrightarrow{RP} T_L^\bullet$ and $T_o \xrightarrow{RP} T_o^\bullet$ are controlled through a truncation to the right of T_o's local interval referring to the lower boundary of $I(T^\bullet, x)$:
    $I(T_o,x) := I(T_o,x) \cap [\, 0 ; \text{lower}(I(T^\bullet, x))\,]$.

6. Conclusions

In a distributed system, when a mixed concurrency control is applied, this poses a difficult problem : How to guarantee global serializability from a set of subtransactions controlled by different techniques. The practical solutions proposed in the literature are often limited :

- On a site there is only one type of technique. This permits interconnection of existing databases to be dealt with, but this is not sufficient to manage a more dynamic heterogeneity.
- Each site uses a total SO, thus leading to useless rejection.
- The compatibility of the local SOs is realized during a global centralized control phase, thus limiting the parallelism.

In order to rectify these limitations, we have proposed a new approach, relying on the following ideas :

- Principles dealing with distributed concurrent certification.
- Extending these principles to the local management of mixed conflicts, and defining a general distributed mixed method.

The most significant features of the method are :

- The global transactions do not need centralized control. A global - multisite - transaction certification

involves only the concerned sites. The set of all committed transactions does not have to be explicitly considered. Therefore, the monosite transactions are only controlled locally.

- An increase in parallelism obtained from the processing of local certification with several subtransactions being controlled on the same site.
- A decrease in rejection rate obtained by replacing the local total order by weaker constraints.

Finally this method supports the dynamism needed in general purpose distributed systems :

- A "per transaction" heterogeneity allowing the applications to select, in an adaptive way, the type allocated to their transactions.
- A "per site" heterogeneity, which could even be "per objects", permitting either pessimistic or optimistic techniques to be applied, in a time variant strategy, depending on both when and where they will be most effective.

References

[Badal 84] Badal D.Z. and McElyea W.,"A robust adaptive concurrency control for distributed databases", in *Proc. IEEE INFOCOM 84*, April 1984, pp. 382-391.

[Bayer 82] Bayer R., Elhardt K., Heigert J. and Reiser A. , "Dynamic timestamp allocation for transactions in database systems", in *Proc. 2nd Int. Symp. Distributed Databases*, 1982, pp. 9-20.

[Bernstein 81] Bernstein P. and Goodman N. , "Concurrency control in distributed database systems", *ACM Comput. Surveys*, vol. 13, no. 2, pp. 185-221, June 1981.

[Boksenbaum 84] Boksenbaum C., Cart M., Ferrié J. and Pons J.F., "Certifications by intervals of timestamps in distributed database systems", in *Proc. 10th Int. Conf. Very Large Data Bases*, 1984, pp. 377-387.

[Boksenbaum 85] Boksenbaum C., Cart M., Ferrié J. and Pons J.F., "Concurrent certifications in distributed database systems", in *Proc. 8th Int. Comput. Symp.*, 1985, pp. 11-19.

[Boksenbaum 87] Boksenbaum C., Cart M., Ferrié J. and Pons J.F., "Concurrent certifications by intervals of timestamps in distributed database systems", *IEEE Trans. on Soft. Eng.*, vol 13, no. 4, April 1987.

[Boral 84] Boral H. and Gold I., "Towards a self-adapting centralized concurrency control algorithm", *ACM Sigmod Rec.*, vol. 14, no. 2, pp. 18-32, June 1984.

[Carey 83] Carey M.J., "An abstract model of database concurrency control algorithms", in *Proc. ACM-SIGMOD Int. Conf. Management of Data*, 1983, pp. 97-107.

[Ceri 82] Ceri S. and Owicki S., "On the use of optimistic methods for concurrency control in distributed database systems", in *Proc. 6th Berkeley Workshop Distributed Data Management and Computers Networks*, 1982, pp. 117-129.

[Elmagarmid 86] Elmagarmid A.K., "A survey of distributed deadlock detection algorithms", *SIGMOD RECORD*, vol. 15, no. 3, pp. 37-45, Sep. 1986.

[Elmagarmid 87] Elmagarmid A.K. and Leu Y., "An optimistic concurrency control algorithm for heterogeneous distributed database systems", *Data Engineering, IEEE bulletin*, vol. 10, no. 3, pp. 26-32, Sept. 1987.

[Eswaran 76] Eswaran K.P., Gray J.N., Lorie R.A. and Traiger I.L., "The notions of consistency and predicate locks in a database system", *Commun. ACM*, vol. 19, no. 11, pp. 624-633, Nov. 1976.

[Gligor 85] Gligor V.D. and Popescu-Zeletin R., "Concurrency control issues in distributed heterogeneous database management systems", *Distributed Data Sharing Systems, Elsevier science Publishers (North-Holland)*, pp. 43-56, 1985.

[Haerder 84] Haerder T., "Observations on optimistic concurrency control schemes", *Inform. Syst.*, vol. 9, no. 2, pp. 11-120, 1984.

[Herlihy 87] Herlihy M., "Optimistic concurrency control for abstract data types", *ACM Operating Systems Review*, vol. 21, no. 2, April 1987.

[Kung 81] Kung H.T. and Robinson J.T.,"On optimistic methods for concurrency control", *ACM Trans. Database Syst.*, vol.6, no. 2, pp. 213-226, June 1981.

[Lai 84] Lai M.Y. and Wilkinson W.K., "Distributed transaction management in Jasmin", in *Proc. 10th Int. Conf. Very Large Data Bases*, 1984, pp. 466-470.

[Lausen 82] Lausen G., "Concurrency control in database systems: a step towards the integration of optimistic methods and locking", in *Proc. ACM Conf.*, Oct. 1982, pp. 64-68.

[Mohan 84] Mohan C., "Recent and future trends in distributed data base management", in *Proc. of NYU Symp. on New Directions in Data Base Systems*, 1984.

[Papadimitriou 79] Papadimitriou C.H., "Serializability of concurrent updates", *J. ACM*, vol. 26, no. 4, pp. 631-653, Oct. 1979.

[Pons 86] Pons J.F., "Contrôle de la cohérence des accès aux objets dans les systèmes répartis: Application des règles d'écriture recouverte.", Thèse de Doctorat, Montpellier 1986.

[Pons 88] Pons J.F., Vilarem J.F., "A dynamic and integrated concurrency control for distributed databases", in *Proc. 3rd Int. Conf. on Data and Knowledge Bases*, 1988.

[Pu 87] Pu C., "Superdatabases : transactions across database boundary", *Data Engineering, IEEE bulletin*, vol. 10, no. 3, pp. 19-25, Sept. 1987.

[Schlageter 82] Schlageter G.,"Problems of optimistic concurrency control in distributed database systems", *ACM SIGMOD Rec.*, vol. 12, no. 3, pp. 62-66, Apr. 1982.

[Sheth 86] Sheth A.P. and Liu M.T., "Integrating locking and optimistic concurrency control in distributed database systems", in *Proc. 6th Int. Conf. on Distributed Computing Systems*, 1986, pp. 89-99.

[Sinha 85] Sinha M.K., Nanadikar P.D. and Mehndiratta S.L., "Timestamp based certification scheme for transactions in distributed database systems", in *Proc. ACM SIGMOD Int. Conf. on Management of Data*, 1985, pp. 402-411.

[Traiger 82] Traiger I., Gray J., Galtieri C. and Lindsay B., "Transactions and consistency in distributed database systems", *ACM Trans. Database Syst.*, vol. 7, no. 3, pp. 323-342, Sept. 1982.