An Optimistic Locking Technique For Concurrency Control in Distributed Databases

Ugur Halici, *Member, IEEE,* and Asuman Dogac, *Member, IEEE*

Abstract— **An optimistic scheme, called ODL, which uses dummy locks to test the validity of a transaction for concurrency control in distributed database systems, is suggested. The dummy locks are long-term locks; however, they do not conflict with any other lock. By the use of long-term dummy locks, the need for the information about the write sets of validated transactions is eliminated and during the validation test only the related sites are checked. Also, the transactions to be aborted are immediately recognized before the validation test, and therefore, the costs of restarts are reduced. Furthermore, usual read and write locks are used as short-term locks during the validation test. The use of short-term locks in the optimistic approach eliminates the need for the system-wide critical section and results in a distributed and parallel validation test.**

Although locking is used, the method is deadlock free; therefore, the deadlock overhead is avoided. This is achieved by preordering the data items, which is very convenient by the time of certification since the access set is known. The ODL method is very simple and easy to implement. The performance of ODL is compared with strict 2PL through simulation, and it is found out that for the low conflict cases they perform almost the same, but for the high conflicting cases ODL performs better than strict 2PL.

Index Terms— **Concurrency control, correctness of concurrency control, distributed database systems, optimistic locking, optimistic schedulers, serializability, two-phase locking.**

I. Introduction

A METHOD called optimistic method with dummy locks (ODL) is suggested for concurrency control in distributed databases. We can summarize the distinguishing features of the ODL method as follows:

- It combines locking and optimistic techniques
- By the use of dummy locks, the need for the information on the write sets of the validated transactions is eliminated and during the validation test, only the related sites are communicated with. This reduces the communication cost. Also, the transactions to be aborted are immediately recognized before the validation test. This prevents one of the major drawbacks of the previous optimistic methods, that is, the costly restarts
- By the use of short-term read and write locks during validation test, the need for system-wide critical section is eliminated and a distributed and parallel validation test is achieved. This increases the performance of the method compared to previous optimistic methods
- An invalidated transaction does not cause any other transaction to be invalidated, which increases the performance of the method
- The method is deadlock free; therefore, deadlock overhead is eliminated
- Two-phase commitment protocol is integrated into the method; in other words, certification information is exchanged during commit protocol. Therefore, the total number of messages necessary in ODL is comparable to distributed two-phase locking (2PL), if there is no cost of handling deadlock in 2PL, such as in timeout technique. However, if deadlock is handled by considering wait-for-graph, such as in Snoop, then the message cost of 2PL is considerably greater than ODL
- Contrary to some of the previous methods, ODL is very simple and easy to implement
- Finally, a performance study carried out through simulation has shown that ODL performs the same with strict 2PL using timeout for deadlock resolution in low-conflict cases; however, it is superior to 2PL in high-conflict cases. This performance achievement is the result of the improvements cited above.

Manuscript received May 8, 1990; revised March 26, 1991. Recommended by M. Jarke.

U. Halici is with the Department of Electrical and Electronics Engineering, Middle East Technical University, Ankara 06531, Turkey.

A. Dogac is with the Department of Computer Engineering, Middle East Technical University, Ankara 06531, Turkey.

IEEE Log Number 9100232.

II. Background

2PL is one of the most popular concurrency control techniques in database systems and has been realized in most of the commercial systems [6], [10], [12], [18], [28], [29], [33]. A 2PL scheduler is defined by the following rules: 1) Every transaction must lock a data item before it actually reads or writes that data item. Different transactions can not simultaneously hold conflicting locks. Two locks conflict if they belong to conflicting operations. Two operations conflict if they belong to different transactions and one of them is a write operation; and 2) once the scheduler has released a lock for a transaction, it may not subsequently obtain any more locks for that transaction. On the other hand, strictness is a very desirable property of histories and almost all implementations of 2PL are strict [5]. A strict 2PL scheduler releases all the locks held by a transaction after the transaction commits [12], [15]. In this paper, unless otherwise stated, strict 2PL is assumed.

Two-phase commitment (2PC) protocol is the most common commit protocol used in distributed database systems. In the first phase of 2PC, the values of data items in the write set of the transaction are copied into the secure storage at the related

sites. If the first phase terminates successfully, the transaction commits. Then, in the second phase, the commit message is sent to the related sites and the effects of the transactions on the database are made permanent [17], [34].

In this paper a transaction execution model where the read operations are multistep and write operations are single step [14], [20], [22] is assumed. For this purpose, the read operations are scheduled as they arrive. However, the write operations are performed on the private workspace of the transaction and are copied into the database when the transaction commits [2], [5], resulting in strict execution of the transactions.

The locking methods when used in distributed databases necessitate a distributed deadlock handling mechanism. There are several deadlock prevention and detection algorithms, such as wait-for-graph, Snoop, timeout, wound–wait, and wait–die [13], [24], [29], [30], [37]. Among the various distributed deadlock detection algorithms, the one implemented in System R seems to be more widely known [30].

Kung [26] has suggested the optimistic approach to concurrency control to solve these problems. Optimistic schedulers are also known as certifiers. A scheduler working in the certifier mode immediately schedules each operation it receives. Only at the end of the transaction, if a validation test is passed by the transaction, is the transaction committed [3], [26].

In the optimistic approach any transaction consists of two or three phases: a read phase, a validation phase, and a possible write phase. During the read phase, reads are done unrestrictedly. However, all the writes take place on the private workspace of the transaction. Then, if it can be established during the validation phase that changes the transaction made will not cause a loss of integrity, the writes are copied into the database in the write phase [23], [26], [35].

In the Kung's optimistic (KO) method, for each transaction Ti, the scheduler keeps the track of all the other transactions that have been validated after Ti has started. For this purpose, transactions are assigned monotonically increasing unique transaction numbers when they are validated. During the validation test of a transaction Ti, the intersection of the read set of Ti with the write sets of each of those transactions, which are validated after Ti has started, are tested. If all of the intersections are empty, then the transaction Ti is validated and the write phase is performed. This is serial validation. The disadvantages of this method are as follows: 1) Both validation phase and the write phase are realized in a system-wide critical section; that is, there is parallelism neither in the validation phase (serial validation) nor in the write phase, which lowers the amount of concurrency; 2) the write sets of all transactions that have been validated after Ti has started must be stored although they are terminated; and 3) furthermore, in the distributed case, either all of the sites must be checked to validate any transaction, or there should be a central scheduler at a site. Both of the schemes result in high communication cost.

To increase concurrency the parallel validation test is proposed in [26]. Here both the validation test and the write phase are taken out of the critical section. The transactions that are currently being validated are termed as active transactions. In parallel validation the active transactions are decided in a critical section. Then for a transaction Ti to be validated, the intersections of the read set of Ti with the write set of all the other transactions that have been validated after Ti has started must be empty, and also the intersection of the read set or the write set of Ti with the write sets of all active transactions must be empty. The validated transactions are assigned monotonically increasing unique transaction numbers again in a system-wide critical section.

In the parallel validation of KO method, the active transactions with conflicting write operations are aborted; however, in the serial validation, since the write operations are performed in critical section, the transactions with conflicting write operations are not aborted, but blocked.

It should be noted that by assuming standard transaction execution model, the class of histories produced by the KO method is a subset of the class of histories produced by the 2PL method. As an example, the history $Ha = R2[x]\,R1[y]\,R2[z]\,W2[x]\,R1[x]\,W1\,[x,y]$ cannot be produced by the KO method, because T2 is validated after T1 has started execution, and the intersection of the read set of T1 and the write set of T2 is not empty. On the other hand, a 2PL scheduler produces every history that can be produced by KO method. Let H be a history that can be handled by the KO method, that is, the history H is handled without any change on the order of the operations and all the transactions in H are validated. The validation of a transaction Ti implies that there were no conflicting write operations by any other transaction Tj during the time that Ti was executing its read steps. This implies that Tj (Ti) can obtain immediately all the write (read) locks it requests. Because Tj is also validated, Tj obtains all read locks immediately as it requests. Therefore, any transaction Tj can obtain any lock it requires and the history is in 2PL.

The major difficulty in locking approaches is deadlock. The optimistic approach prevents this problem but in the KO method the following disadvantages are observed even if the parallel validation is used: 1) There is a system-wide critical section; 2) all of the sites must be checked although the transaction may not have executed at all in some of these sites; and 3) the decision to abort a transaction is reached only when the transaction is terminated, and the transaction performs some unnecessary operations although it is to be aborted.

There are several methods based on the optimistic approach. In [11], the transactions are validated locally at each site according to the parallel validation rules of KO method. The algorithm is distributed by the use of happened before (HB) sets. The HB(Tij) for transaction Ti contains the identifiers of those transactions that precede Tij in the serialization order at site j. During global validation, a subtransaction is considered valid only if all global transactions that belong to its HB set have been either committed or aborted. If it is not yet known for some of those transactions whether they have been committed or aborted, the validation of Tij is suspended. This waiting can cause a deadlock; hence after a timeout, if the subtransaction cannot be validated, it is aborted. In this

method, although there is no system-wide critical section, there is blocking and deadlock.

The backward-oriented optimistic method described in [23] is the same as the KO method with serial validation. Forward-oriented optimistic concurrency control technique (FOCC) [23] checks, during the validation phase of Tj, whether its write set WS(Tj) intersects with any of the read sets RS(Ti) of all transactions Ti having not yet finished their read phases. Since the transactions to be checked during validation have not yet been committed, this approach offers a flexibility over the backward-oriented optimistic method in handling the detected conflict. For example if Tj is a long writer and Ti has just started its execution, Ti may be the victim for abortion. However, in this method, validation is made in a system-wide critical section.

In [36], a timestamp-based optimistic concurrency control algorithm is described. In this method, a read timestamp and a write timestamp is maintained for each data item. For each read, the transaction must remember the write-timestamp associated with the item when it was read. A read request is certified if: 1) the version that was read is still the current version of the item; and 2) no write with a newer timestamp has already been locally certified. A write request is certified if: 1) no later reads have been locally certified and subsequently committed; and 2) no later reads have been locally certified. This algorithm suffers partially from the disadvantages of the timestamp algorithms; that is, the timestamps assigned may cause unnecessary abortions, and the timestamps kept with each data item consumes a lot of storage space.

In [7], an optimistic concurrency control technique, which uses the time interval technique [4], is introduced. However the method is complicated and requires too many messages.

In [22] an optimistic concurrency control algorithm that uses the time interval technique in conjunction with short-term locks, called the OSN method, is described. It has been shown through log classification that this method provides more concurrency than 2PL. However the algorithm is complicated, therefore, brings difficulty to the implementation.

In optimistic methods, it is difficult to distribute the validation test and restarts takes a long time. Most of the optimistic methods use either local or system-wide critical section. The performance of some of the previous optimistic methods have been compared with locking methods [1], [2], [8], [9], [16], [27]. While in some studies 2PL is found superior [1], [27], in [16] optimistic methods are found superior. In [2], the difference among the decisions reached are explained by the different assumptions made. In [8], it is predicted that "optimistic locking," where transactions lock remote copies of data only as they enter into commit protocol may actually be the best performer in replicated databases where messages are costly.

III. The ODL Method and the Implementation

In the ODL method three types of locks—read locks, write locks, and dummy locks—are used. The read and write locks conflict in the usual manner. A dummy lock does not conflict with any other lock. In fact, a dummy lock can be interpreted as a special mark such that it is possible to check its existence, and if it exists it is possible to know which transaction has put the lock on which data item. When a transaction Ti issues a read command for a data item x, if the data item is not already in its workspace, a read lock is demanded on the data item. When the lock is granted, a dummy lock is requested on the data item by Ti. A dummy lock request is always granted because it does not conflict with any other lock. Then the value of the data item x is read and the read lock is released. A dummy lock can be released by the transaction itself during the validation test or by another transaction Tj when Tj performs an actual write operation on this data item. The write operations performed in private workspace are not actual write operations, and therefore, do not effect the dummy locks. When the dummy lock of a transaction Ti is released by another transaction Tj, transaction Ti is said to be invalidated.

If the transaction Ti terminates successfully, then the validation test is applied on Ti. During the validation test, the short-term read and write locks are requested on the base set of the transaction as explained in the following, and these locks are retained until the transaction is either committed or aborted. The validation test embodies the two-phase commit (2PC) protocol, that is, the values in the write set of transaction Ti are copied into secure storage in the related sites while the validation test is applied. During the validation test of transaction Ti all the data items in the base set of Ti, that is, the set of data item either in the read or write set of Ti, are accessed in a predefined order. Such a total ordering on the data items is obtained by ordering the sites and then the data items within the sites. Total ordering of the data items is necessary only for preventing the deadlocks. Obviously, other techniques for prevention or detection of deadlocks caused by short-term locks can be used instead of preordering the data items. The validation test is as follows.

1) If the data item is only in the read set of the transaction, then a read lock on the data item is requested. However, if the data item is in both the write set and the read set of the transaction, then a write lock is requested instead of the read lock. When the lock is granted, the existence of the dummy lock by the transaction on the data item is checked. If it has been released by some other transaction, then the test fails; otherwise, the dummy lock is released and the test continues.

2) If the data item is only in the write set of the transaction then only a write lock is demanded on the data item and the test continues when the lock is granted.

The test continues for all data items in the read set and the write set of the transaction Ti until either both of these sets are exhausted or the test fails. If the test fails, then the transaction is restarted after releasing all the locks it owned. If the test terminates successfully, that is, if the test does not fail and the base set is exhausted, then the transaction is committed and the data items in the write set of the transaction Ti are updated by obtaining the values from the secure storage, which corresponds to the second phase of the 2PC protocol. Meanwhile, the dummy locks held by other transactions on the data items in the write set of Ti are released, which means invalidation of those transactions. Also, all the locks owned by Ti are released.

The method is deadlock free, because: 1) the dummy locks do not conflict with any other lock; 2) the read locks requested during the read phase are immediately released after a dummy lock is obtained; and 3) conflicting short-term read and write locks are requested in a predefined order during the validation test. The preordering is applicable due to the fact that by the time of the certification the base set of the transaction is known.

When transaction Ti invalidates another transaction Tj by releasing a dummy lock held by Tj, the transaction Tj can be immediately informed of this invalidation. Therefore, the transactions to be aborted are immediately recognized and such transactions are restarted without performing unnecessary operations.

It should be noted that, in the ODL method, read–write conflicts are prevented by the validation test and the write–write conflicts are prevented by blocking during the write phase. This is the reason why all the histories produced by the method are serializable.

In the following the method is explained through an example history:

$$Ha = R2[x]\, R1[y]\, R2[z]\, W2[x]\, R1[x]\, W1[x, y].$$

First, $R2[x]$ arrives and T2 demands a read lock on data item x. When the lock is granted, a dummy lock is requested on the data item x by transaction T2 and the read lock is released by leaving the dummy lock on the data item. Similarly, when $R1[y]$ arrives it is served by leaving a dummy lock on data item y and then $R2[z]$ operation is served by leaving a dummy lock on z.

$W2[x]$ arrives and the validation test for T2 starts. The related data items are x and z. T2 demands a write lock on data item x and meanwhile, the value of data item x is written into a secure storage at the related sites, since x is in the write set of T2. When the lock is granted, the existence of dummy lock by T2 on x is checked, since x is also in the read set of T2. The dummy lock exists; therefore, the dummy lock is released and the test continues. Then T2 requests a read lock on data item z. When the lock is granted, the existence of the dummy lock on z by T2 is checked. Since the dummy lock is still there, the test does not fail. The dummy lock is released. Since all the items in the access set of T2 have been processed and all the checks are turned out to be valid, the transaction T2 is validated and also committed. Its write operation is applied into the database by copying from the secure storage. All the dummy locks held by other transactions on the data items in write set of T2 are released. However, note that in this example there is no such dummy lock. Also, the short-term read and write locks held by T2 are released.

$R1\ [x]$ arrives and the dummy lock on data item x is obtained by T1.

$W1[x, y]$ arrives and the validation test for T1 starts. T1 demands write locks first on data item x and then on y. Meanwhile, the values for data items x and y are copied into secure storage. As the locks are granted, the existence of the dummy locks on those data items that are in the read set of T1 are checked and these dummy locks are released. Since the test terminates successfully, the transaction T1 is validated. Its write operations are applied into the database and the locks are released.

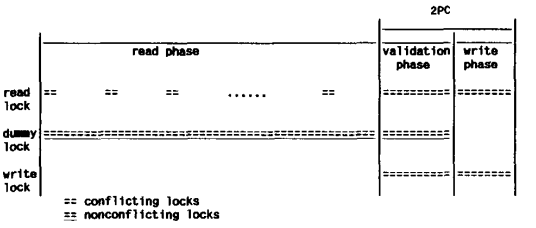

Fig. 1. The phases of a transaction and the duration of the locks.

Note that no transaction Ti can invalidate on already validated transaction Tj. Because in order for Ti to invalidate Tj, a data item x must be common to Tj's read set and Ti's write set. When Ti obtained a write lock on x, since x is in the read set of Tj and Tj is already validated, either: a) Tj already has a lock on x and this lock is conflicting with Ti's write lock, therefore, Ti cannot obtain the write lock unless Tj releases its lock; or b) Tj has terminated its validation test and released its locks; now Ti can obtain the write lock but cannot invalidate Tj because Tj's dummy locks have already been released.

Because the dummy locks can be released only by a transaction itself, or by the validated transactions, and because a transaction cannot be invalidated if it is already validated, the transactions cannot be invalidated by invalidated transactions.

Clearly, the validation test and the 2PC are integrated. A validated transaction is also a committed transaction. If a failure occurs in a related site before or when the transaction is being validated, the transaction is aborted. If a failure occurs after the transaction is validated, it is handled in the usual way as the committed transactions are handled in 2PC.

In Fig. 1, how the phases of a transaction are interleaved with the 2PC is shown. Furthermore, the duration of the locks is demonstrated in the figure.

IV. The History Classes and the Amount of Concurrency

In this paper, a transaction execution where several read operations are followed by a single write operation [22] is assumed. The quadruple $H = (n, \pi, V, S)$ is used to denote a history (log), where n denotes the number of transactions, V is the set of the data items, π is a permutation on the set of the operations in the history showing the order of the appearance of these operations in the history, and S is a function mapping the set of operations to 2^v, indicating the set of data items related to each operation. The symbol T is used to denote the set of transactions in the history. Ti is the ith transaction, Rij is the jth read operation of Ti, and Wi is the write operation of Ti. Unless otherwise stated, the histories used in this paper include two special transactions, an initial transaction Ts, which initially writes all the data items, and a final transaction Tf, which finally reads all the data items. Furthermore, in order to get rid of the extra notation needed to indicate the beginning and the end of a transaction,

we assumed that for a transaction Ti, $Ri1$ corresponds to the beginning of the transaction and Wi indicates the end of it although the read set or the write set of Ti may be empty.

Therefore, the transactions $T1$ and $T2$ of history $Ha = R2[x]R1[y]R2[z]W2[x]R1[x]W1[x,y]$, which is the one given in Section II, can be expressed as follows:

$$
\begin{aligned}
T1 &= R1[y]R1[x]W1[x,y] = R11R12W1\\
R11 &= R1[y] \text{ and } \pi(R11) = 2 \text{ and } S(R11) = \{y\}\\
R12 &= R1[x] \text{ and } \pi(R12) = 5 \text{ and } S(R12) = \{x\}\\
W1 &= W1[x,y] \text{ and } \pi(W1) = 6 \text{ and } S(W1) = \{x,y\}\\
T2 &= R2[x]R2[z]W2[x] = R21R22W2\\
R21 &= R2[x] \text{ and } \pi(R21) = 1 \text{ and } S(R21) = \{x\}\\
R22 &= R2[z] \text{ and } \pi(R22) = 3 \text{ and } S(R22) = \{z\}\\
W2 &= W2[x] \text{ and } \pi(W2) = 4 \text{ and } S(W2) = \{x\}.
\end{aligned}
$$

A history is said to be serial if each transaction in the history executes its write operation before the next transaction executes any of its read operations. A history is assumed to be correct if it is serial. A transaction Tb is said to read a data item x from transaction Ta, if: 1) Ta writes x; 2) then Tb reads x; and 3) no other transaction writes x between these two operations. If an interleaved execution of transactions produces the same effect as a serial execution of those same transactions, then the history is said to be serializable. In this paper, view serializability is chosen as the correctness criterion in which the existence of a serial history having the same read from relation on the same transaction set is required.

Fig. 2 demonstrates the amount of concurrency provided by the existing concurrency control techniques through log classification [22], [32]. In this figure, each area represents a set of histories satisfying a particular property. Area H contains the set of all histories, area VSR contains the set of all view serializable histories, and area S contains only serial histories. HD is the largest known view serializable class that is introduced in [18] and [19]. Furthermore, CPSR are the conflict preserving serializable histories, which are produced by serialization graph testing, and 2PL are the histories produced by strict 2PL, which is a subset of general 2PL. ODL are the histories produced by the ODL method, and KO are the histories produced by the Kung's optimistic method. OSN is the subset of the histories produced by the OSN scheduler introduced in [22], without the application of Thomas Write Rule (TWR) [39]. BTO is the subset of the histories produced by the basic timestamp ordering scheduler [6], without TWR.

In [32] it is proved that the set CPSR covers the general 2PL class, which is a superset of S. In [18] and [22] it is proved that OSN covers both 2PL and BTO; however, when TWR is not allowed, OSN becomes a subset of CPSR. In [25] it is proved that there is a class of histories called WRW, which covers the class CPSR; however, WRW does not have a scheduler. In [18] and [21] it is proved that HD is the largest known view serializable history through concept of ordering constraints defined on transaction sets in [18] and [19]. In the following it will be proved that all histories in class ODL are

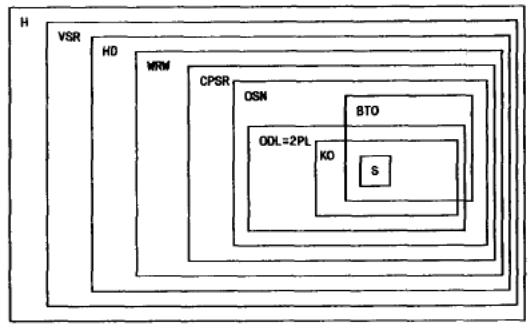

Fig. 2. The amount of concurrency provided by the existing methods.

CPSR, and furthermore, the classes ODL and strict 2PL are the same, and they cover the class KO.

CPSR is more restrictive than the view serializability and requires the conflicting operations to appear in the same order in a serial history as they appear in the original history. The history class definition for CPSR, is given in the following.

Definition 4.1: A history $H = (n, \pi, V, S)$ is in class CPSR if and only if we can find real numbers $\{S1, \cdots, Sn\}$ such that

CPSR1) if $S(Wi) \cap S(Wm) \neq \emptyset$ and $\pi(Wi) < \pi(Wm)$, then $Si < Sm$.

CPSR2) if $S(Rij) \cap S(Wm) \neq \emptyset$ for some j, $i \neq m$ and $\pi(Rij) < \pi(Wm)$, then $Si < Sm$

CPSR3) if $S(Wi) \cap S(Rmj) \neq \emptyset$ for some j and $\pi(Wi) < \pi(Rmj)$, then $Si < Sm$. ■

Let H be a history satisfying the CPSR rules given in Definition 4.1 and let Hs be the serial history having the same set of transactions that are executed in the order of the real numbers obtained in accordance with Definition 4.1. The rule CPSR1 provides for the conflicting write operations of Hs to be in the same order as in the history H itself. The rules CPSR2 and CPSR3 provide for the ordering of the conflicting read and write operations of Hs as they appear in the history H. Therefore, there exists a serial history Hs having the conflicting operations in the same order as they appear in H, which is the requirement of the CPSR.

In order to see whether the history Ha is in the class CPSR, we will try to find real number $Si = \{S1, S2\}$ consistent with the rules defining the class CPSR:

1) $S(W1) \cap S(W2) \neq \emptyset$ and $\pi(W1) < \pi(W2)$ then $S2 < S1$ by CPSR1

2) $S(R21) \cap S(W1) \neq \emptyset$ and $\pi(R21) < \pi(W1)$ then $S2 < S1$ by CPSR2

3) $S(W2) \cap S(R11) \neq \emptyset$ and $\pi(W2) < \pi(R11)$ then $S2 < S1$ by CPSR3.

It is clear that we can find two real numbers such that $S2 < S1$; then Ha is in class CPSR and in the serial equivalent history

$$Hs = T2T1 = R2[x]R2[z]W2[x]R1[y]R1[x]W1[x,y]$$

transactions are executed in the order determined by $S1$ and $S2$.

The history class definitions of strict 2PL, KO, and the proposed method ODL, are provided in the following:

Definition 4.2: A history $H = (n, \pi, V, S)$ is in class strict 2PL if and only if we can find real numbers $\{S1, \cdots Sn\}$ such that

2PL1) $\pi(Rij) \leqq Si \leqq \pi(Wi)$ for $i = 1, \cdots, n$, $j = 1, 2, \cdots, ki$

2PL2) if $S(Rij) \cap S(Wm) \neq \varnothing$ for some j, $i \neq m$ and $\pi(Rij) < \pi(Wm)$, then $\pi(Wi) < Sm$

2PL3) if $S(Wi) n S(Wm) \neq \varnothing$ and $\pi(Wi) < \pi(Wm)$, $\pi(Wi) < Sm$. ■

Definition 4.2 is derived from the general 2PL definition in [32]. Here rule 2PL2 has been restricted to $\pi(Wi) < Sm$ instead of $Si < Sm$ as a consequence of using strict 2PL.

Definition 4.3: A history $H = (n, \pi, V, S)$ is KO if and only if we can find real numbers $\{S1, \cdots, Sn\}$ such that

KO1) $Si = \pi(Wi)$

KO2) if $S(Rij) n S(Wm) \neq \emptyset$ for some j, $i \neq m$ and $\pi(Ri1) < \pi(Wm)$, then $Si < Sm$. ■

In Definition 4.3, KO1 states that the transactions are serialized in their validation order, which is determined by the order of their write operations in the history. KO2 states that for a transaction Ti to be validated, the following condition must hold at the validation time: the intersection of the read set of Ti with the write sets of each of those transactions Tm that are validated after Ti has started must be empty. It should be noted that $Ri1$ denotes the start of transaction Ti.

In order to decide whether a given history is in KO, we will try to find real number $Si = \{S1, S2\}$ consistent with the rules defining the class KO.

1) $S1 = \pi(W1) = 6$ by $KO1$

2) $S2 = \pi(W2) = 4$ by $KO1$, therefore $S2 < S1$ by 1 and 2

3) $S(R11) \cap S(W2) \neq \varnothing$ and $\pi(R11) < \pi(W2)$, therefore, $S1 < S2$ by $KO3$

which makes it impossible to find real numbers consistent with the constraints of KO.

Definition 4.4: A history $H = (n, \pi, V, S)$ is in class ODL if and only if we can find real numbers $\{S1, \cdots, Sn\}$ such that

ODL1) $Si = \pi(Wi)$

ODL2) if $S(Rij) \cap S(Wm) \neq \varnothing$ for some j, $i \neq m$ and $\pi(Rij) < \pi(Wm)$, then $Si < Sm$. ■

In Definition 4.4, ODL1 states that the transactions are serialized in their validation order, which is determined by the order of their write operations in the history. ODL2 states that for a transaction Ti to be validated, the following condition must hold at the validation time: the data item related to any read operation Rij of Ti should not appear in the write set of any of those transaction Tm that are validated after Rij has been issued.

In order to show the correctness of a concurrency control algorithm, it is necessary and sufficient to prove that it produces only view-serializable histories. All the histories in the class CPSR are view serializable. In order to show that all the histories produced by the ODL scheduler are view serializable, we will show that the class ODL is a subset of the class CPSR. The following theorem is provided to prove this fact.

Theorem 4.1: Any history in class ODL is in Class CPSR.

Proof: Let H be a history in class ODL. In the following we show that each CPSR rule is implied by the ODL rules.

i) Let $S(Wi) \cap S(Wm) \neq \varnothing$ and $\pi(Wi) < \pi(Wm)$. By ODL1 $Si < Sm$. Hence CPSR1 is implied.

ii) CPSR2 is the same as ODL2.

iii) Let $S(Wi) \cap S(Rmj) \neq \varnothing$ for some j and $\pi(Wi) < \pi(Rmj)$. $\pi(Rmj) < \pi(Wm)$ for any j by the assumed transaction execution model. Therefore, $\pi(Wi) < \pi(Wm)$ by transitivity and then $Si < Sm$ by ODL1. Hence CPSR3 is satisfied. Therefore, by i), ii), and iii), any history in class ODL is CPSR. ■

In the following the class ODL is compared by the classes KO and Strict 2PL.

Theorem 4.2: The class ODL is a proper superset of class KO.

Proof: The rules KO1 and ODL1 are the same: however, the rule KO2 is more restrictive than the rule ODL2. The history Ha of section 2 is in class ODL, although it is not in class KO. Therefore the class ODL is a proper superset of the class KO. ■

Theorem 4.3: The class ODL is equivalent to the class strict 2PL.

Proof: Let H be a history. We will prove that the history H is in class strict 2PL if and only if H is in class ODL.

(if): In the following we show that each 2PL rule is implied by the ODL rules.

i) Rule ODL1 is more restrictive than 2PL1.

ii) Let $S(Rij) \cap S(Wm) \neq \varnothing$ for some $j, i \neq m$ and $\pi(Rij) < \pi(Wm)$. Then $Si < Sm$ by ODL2, and then $\pi(Wi) < Sm$ by ODL1. Therefore, 2PL2 is implied.

iii) Rule ODL1 is more restrictive then 2PL3. Therefore, any history in class ODL is also in class 2PL.

(only if): Assume H is not in class ODL. This implies that there is at least one invalidated transaction in the history H. A transaction Ti is invalidated in the proposed method only when the dummy lock on some data item x held by Ti is released by some other transaction Tm, and this in return implies that $x \in S(Rij) \cap S(Wm)$ for some j, and $\pi(Rij) < \pi(Wm) < \pi(Wi)$. However, this implies that $Sm \leqq \pi(Wm) < \pi(Wi)$ by the rule 2PL1 and $\pi(Wi) < Sm$

TABLE I
PERFORMANCE COMPARISON OF ODL AND 2PL FOR LOW-CONFLICT CASE

			ODL				2PL			
DBS	MIAT	MTBS	RS	AB%	TC%	MRT	TO (ms)	AB%	TC%	MRT
1500	10 000	5	0	0	100	1848	1250–10 000	0	100	1848
1000	10 000	5	0	0	100	1841	1250–10 000	0	100	1852
500	10 000	5	0	0	100	1841	1250–10 000	0	100	1852
100	10 000	5	0	0	100	1841	1250–10 000	0	100	1837
1000	10 000	2	0	0	100	934	1250–10 000	0	100	845
1000	10 000	4	0	0	100	1564	1250–10 000	0	100	1504
1000	10 000	5	0	0	100	1841	1250–10 000	0	100	1852
1000	10 000	10	0	0	100	3196	1250–10 000	0	100	3382
1000	20 000	5	0	0	100	1841	1250–10 000	0	100	1852
1000	10 000	5	0	0	100	1841	1250–10 000	0	100	1852
1000	5000	5	0	0	100	1841	1250–10 000	0	100	1852
1000	1000	5	3	2	100	1945	1250	1	100	1959
							2500	1	100	1916
							5000	1	100	1966
							10000	1	100	2066

DBS: database size; MTBS: mean transaction base set size; AB%: Percentage of aborted transactions; TC%: Percentage of throughput capacity; MRT: Mean response time; TO: Timeout duration.

by the rule 2PL2, which is a contradiction. Therefore, H is not in class 2PL. ■

V. PERFORMANCE OF THE ODL METHOD THROUGH SIMULATION

The performance of ODL method is compared to the performance of the strict 2PL technique, which is the most common concurrency control technique, through simulation in [38]. The simulation study is carried out using GPSS, a discrete simulation language. In the 2PL model, the deadlocks are handled with the timeout technique, which is a widely used deadlock resolution technique. However, the timeout is a parameter that has to be tuned well. For this reason the 2PL model is experimented with a set of timeout values, and then the best values of the evaluated performance metrics are chosen for comparison with ODL. These experiments revealed that the performance of 2PL is very sensitive to the fine tuning of the timeout parameter. This is because the timeout technique may abort a transaction that is not really part of the deadlock, but is just waiting for a lock owned by another transaction that is taking a long time to finish. There is certainly a performance penalty to the transaction that was unfairly aborted, although the overall effect may be to improve transaction throughput [6].

In ODL method deadlocks are prevented by using dummy locks and also by ordering the data items during the validation test.

Both of the simulation models include the 2PC protocol, which is a necessity for reliability. It should be noted that in the ODL method, validation test is embedded in the first phase of the 2PC, as explained in Section III.

The values of the parameters used during simulation runs were chosen from [31]. These values are common to the most of the similar studies [1], [2]:

Mean interarrival time of transactions (MIAT)	: 1000–2000 ms
Mean base set size of transactions (MTBS)	: 2–10 data items
The database size (DBS)	: 100–1500 data items
Timeout period for 2PL (TO)	: 1250–10000 ms
Message transmission time	: 100 ms
I/O time to process an item	: 25 ms
Number of sites	: 5 sites.

The network topologies, transmission media, communication protocols, network traffic density, and message size vary to a great extent. Therefore, the ranges for transmission time vary widely. In [31] the range is given as 50–250 ms, and 100 ms is chosen as the typical value. Furthermore, in [31] CPU time to process an item is taken as 1 ms. We have neglected this in comparison to I/O time to process an item (25 ms) and message transmission time (100 ms).

The performance metrics evaluated are the throughput capacity (TC%) that can be supported by the method, which is the percentage of already terminated transactions to all the transactions created, and the mean response time (MRT), which is the time elapsed between the start and the termination of a transaction by taking restarts into account. In our simulation models, aborted transactions are immediately restarted.

The performance results for TC% and MRT (ms) are summarized in Table I, for low conflict cases, in Table II for medium conflict cases, and in Table III for high conflict cases. In these tables, AB% corresponds to the percentage of transactions that are restarted at least once. Notice that timeout TO is a parameter only for the 2PL method.

The graphs obtained through the experiments are given in Figs. 3–8. In each figure, the behavior of either TC or MRT is demonstrated for the following cases:

a) *low conflict case, where* database size = 1000 data items, mean base set size = 5 data items, and mean interarrival time = 10 000 ms;

TABLE II
PERFORMANCE COMPARISON OF ODL AND 2PL FOR MEDIUM-CONFLICT CASE

			ODL				2PL			
DBS	MIAT	MTBS	RS	AB%	TC%	MRT	TO (ms)	AB%	TC%	MRT
1500	5000	10	1	1	100	3245	1250	1	100	3515
							2500	1	100	3540
							5000	1	100	3590
							10 000	1	100	3690
1000	5000	10	1	1	100	3252	1250	1	100	3505
							2500	1	100	3530
							5000	1	100	3580
							10 000	1	100	3680
500	5000	10	3	3	100	3367	1250	5	100	3865
							2500	3	100	3709
							5000	3	100	3517
							10 000	2	100	4041
100	5000	10	12	10	100	3792	1250	42	75	9121
							2500	42	70	5319
							5000	32	79	5946
							10 000	42	63	5836
500	1000	2	0	0	100	937	1250–10 000	0	100	845
500	1000	4	0	0	100	1555	1250–10 000	0	100	1530
500	1000	5	5	4	100	2013	1250	6	100	2185
							2500	4	100	2161
							5000	4	100	2369
							10 000	4	98	2552
500	1000	10	21	13	93	3419	1250	26	80	3946
							2500	16	84	3795
							5000	15	86	5088
							10 000	15	77	5207
500	20 000	10	1	1	100	3238	1250	2	100	3496
							2500	1	100	3448
							5000	1	100	3498
							10 000	1	100	3598
500	10 000	10	2	2	100	3308	1250	3	100	3628
							2500	2	100	3591
							5000	1	100	3517
							10 000	1	100	3617
500	5000	10	3	3	100	3367	1250	5	100	3865
							2500	3	100	3709
							5000	3	100	3517
							10 000	2	100	4041
500	1000	10	21	13	93	3419	1250	26	80	3946
							2500	16	84	3795
							5000	15	86	5088
							10 000	15	77	5207

b) *medium conflict case, where* database size is 500 data items, mean base set size is 10 data items, and mean interarrival time is 1000 ms or 5000 ms;
c) *high conflict case, where* database size is 100 data items, mean base set size is 10 data items, and mean interarrival time is 1000 ms.

In Fig. 3, throughput capacity is plotted against database size. Fig. 3(a) depicts the low conflict cases where throughput capacity is about 100% for both of the methods, which implies that almost no transaction remains in the system. The corresponding MRT's are given in Fig. 4(a) which shows that both methods performs almost the same.

Fig. 3(b) is the medium conflict case, and when database size drops below about 500 items, which implies an increase in the number of conflicts, ODL starts performing better than 2PL, both in terms of throughput and in terms of MRT (Fig. 4(b)). This behavior becomes much apparent in Fig. 3(c) which depicts the high conflict case. From Fig. 3(c) it is clear that when database size drops below about 500 data items, 2PL trashes for high-conflict cases, whereas ODL's performance is quite reasonable.

When 2PL trashes, it terminates very few transactions. Although MRT is low (Fig. 4(c)) for 2PL in this region, standard deviation is very close to MRT; therefore, the MRT is not very meaningful.

Fig. 5 shows the throughput as a function of mean interarrival time for the: a) low-, b) medium-, and c) high-conflict cases. The graphs of the MRT's corresponding to these experiments are given in Fig. 6. The graphs of Figs. 5 and 6 confirm the conclusions we have obtained from the graphs of Figs. 3 and 4, that is, ODL and 2PL perform the same in the low-conflict case; however, as the number of conflicts increases,

TABLE III
PERFORMANCE COMPARISON OF ODL AND 2PL FOR HIGH-CONFLICT CASE

			ODL				2PL			
DBS	MIAT	MTBS	RS	AB%	TC%	MRT	TO (ms)	AB%	TC%	MRT
100	20 000	10	4	4	100	3355	1250	5	100	3960
							2500	6	100	4263
							5000	6	100	6077
							10 000	8	100	9165
100	10 000	10	5	5	100	3461	1250	9	100	4038
							2500	6	100	3938
							5000	6	100	4238
							10 000	6	100	4903
100	5000	10	12	10	100	3792	1250	42	75	9121
							2500	42	70	5319
							5000	32	79	5946
							10 000	42	63	5836
100	1000	10	78	29	87	5313	1250	65	38	3160
							2500	58	40	2983
							5000	60	35	4045
							10 000	59	31	2464
100	1000	2	2	2	100	982	1250	2	100	992
							2500	3	98	895
							5000	3	97	804
							10 000	2	95	854
100	1000	4	10	7	100	1851	1250	11	93	1756
							2500	9	91	1649
							5000	8	88	1443
							10 000	6	88	1646
100	1000	5	17	10	98	2336	1250	19	89	2019
							2500	15	86	1985
							5000	15	84	2342
							10 000	21	74	4001
100	1000	10	78	29	87	5313	1250	65	38	3160
							2500	58	40	2983
							5000	60	35	4045
							10 000	59	31	2464
1500	1000	10	6	4	97	3419	1250	5	96	3554
							2500	4	96	3700
							5000	5	93	3723
							10 000	4	93	3976
1000	1000	10	10	6	96	3319	1250	10	90	3279
							2500	9	91	3655
							5000	8	92	4041
							10 000	6	87	4184
500	1000	10	21	13	93	3419	1250	26	80	3946
							2500	16	84	3795
							5000	15	86	5088
							10 000	15	77	5207
100	1000	10	78	29	87	5313	1250	65	38	3160
							2500	58	40	2983
							5000	60	35	4045
							10 000	59	31	2464

ODL starts performing better than 2PL, and even when 2PL trashes, ODL still has a reasonable performance. Figs. 7 and 8, which show the throughput capacity and mean response times as a function of transaction base set size, provide further support to our conclusions. One can observe from Fig. 7(c) that throughput of 2PL reduces drastically as the mean base set size of transactions increases for the high-conflict case. The MRT corresponding to this region is given in Fig. 8(c). Here the MRT of 2PL seems to be better than ODL; however, this is the region where 2PL trashes as throughput indicates, and the MRT is not very meaningful, as explained above.

In the simulation the CPU time is neglected. Notice that even if the CPU time is significant, this will not change the conclusions about the simulation results with infinite resources assumption. ODL does not incur any extra CPU cost; therefore, the CPU cost will be the same for all the models. The values obtained for performance metrics will change but the conclusions will be the same when it comes to comparison of the techniques.

As a summary, in low conflict cases, almost no transaction is restarted and also no transaction remains in the system resulting in 100% throughput capacity both for the ODL and 2PL methods. In higher conflict cases, ODL performs better

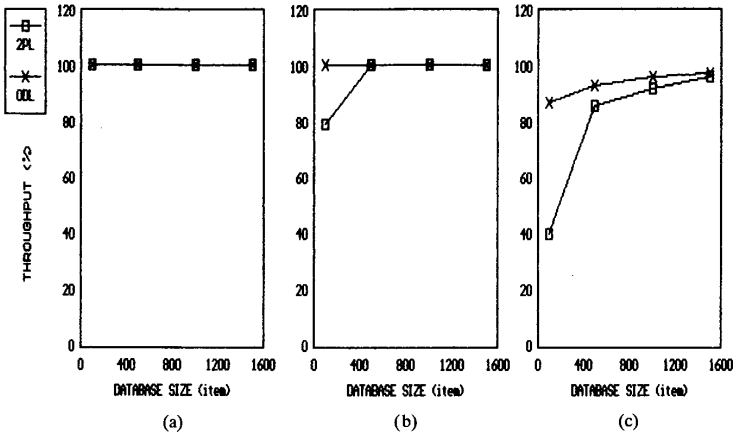

Fig. 3. Effect of database size on throughput capacity. (a) MIAT: 10 000 ms, BS: 5 items (low conflict). (b) MIAT: 5000 ms, BS: 10 items (medium conflict). (c) MIAT: 1000 ms, BS: 10 items (high conflict).

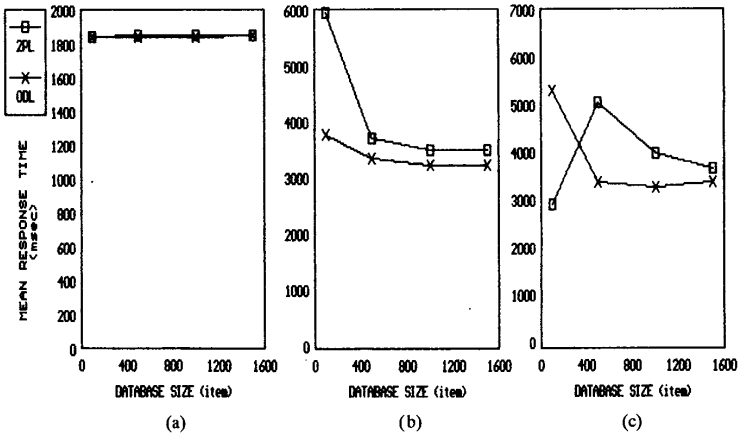

Fig. 4. Effect of database size on MRT. (a) MIAT: 10 000 ms, BS: 5 items (low conflict). (b) MIAT: 5000 ms, BS: 10 items (medium conflict). (c) MIAT: 1000 ms, BS: 10 items (high conflict).

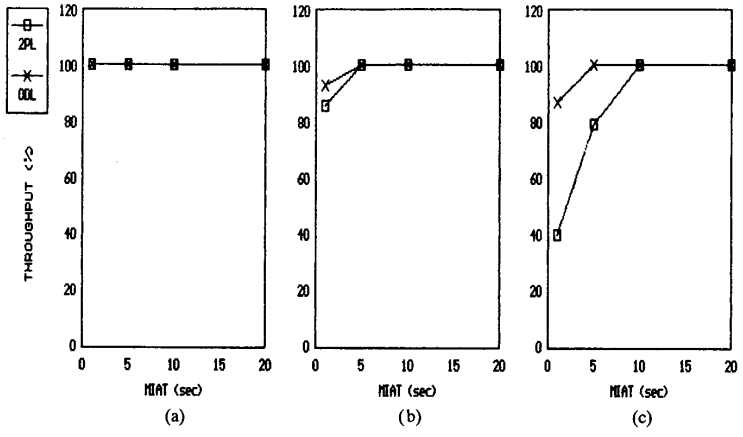

Fig. 5. Effect of mean interarrival time on throughput capacity. (a) DBS: 1000 items, BS: 5 items (low conflict). (b) DBS: 500 items, BS: 10 items (medium conflict). (c) DBS: 100 items, BS: 10 items (high conflict).

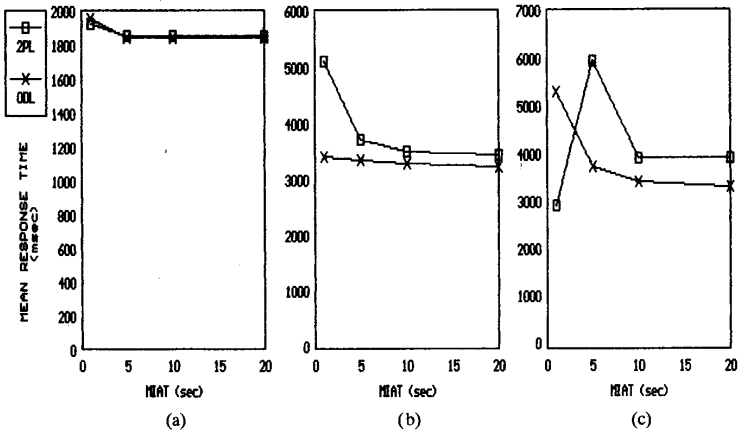

Fig. 6. Effect of mean interarrival time on MRT. (a) DBS: 1000 items, BS: 5 items (low conflict). (b) DBS: 500 items, BS: 10 items (medium conflict). (c) DBS: 100 items, BS: 10 items (high conflict).

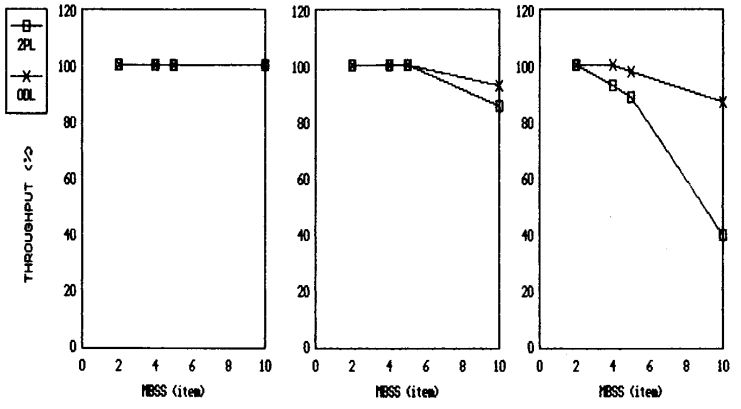

Fig. 7. Effect of mean base set size on throughput capacity. (a) MIAT: 10 000 ms, DBS: 1000 items (low conflict). (b) MIAT: 1000 ms, DBS: 500 items (medium conflict). (c) MIAT: 1000 ms, DBS: 100 items (high conflict).

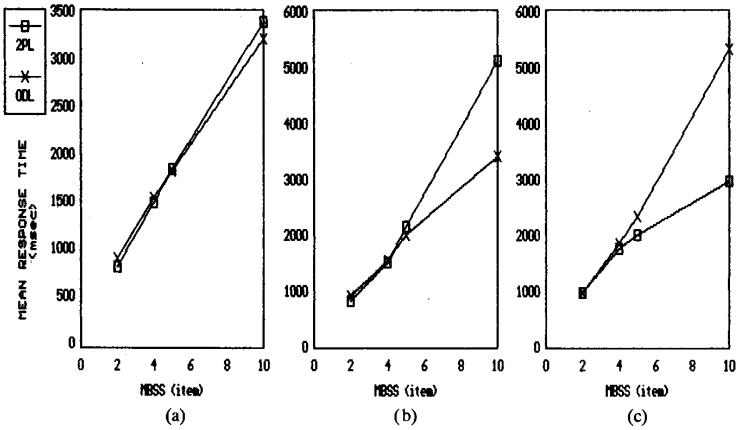

Fig. 8. Effect of mean base set size on MRT. (a) MIAT: 10 000 ms, DBS: 1000 items (low conflict). (b) MIAT: 1000 ms, DBS: 500 items (medium conflict). (c) MIAT: 1000 ms, DBS: 100 items (high conflict).

than 2PL. This is because the long-term locks used in 2PL rise blocking beyond an unacceptable level. These results are in agreement with our expectations. The use of dummy locks and short-term locks in ODL works very much in advantage of the method, by avoiding blocking and deadlock overhead, since there is no deadlock in ODL. Furthermore, the certification information is exchanged during 2PC protocol; therefore, certification is almost free in terms of communication cost. The most important feature of the ODL is the following: the invalidated transactions are immediately recognized, and therefore, the cost of restart is not high as in the previous optimistic methods.

VI. Conclusion

In this paper, an optimistic scheme is introduced. The synchronization of read–write conflicts are achieved through dummy locks. The advantage of using dummy locks is that although they are long-term locks, they do not block the execution of transactions in any way. The nonexistence of an expected dummy lock informs a transaction that it has been invalidated. During the validation test, it is decided on whether a transaction is invalidated. However, a transaction may be invalidated before it enters the validation test and such a transaction is immediately recognized and aborted.

2PC is integrated into the validation test and parallelism is achieved by the use of short-term read and write locks during the validation test. The validation test is distributed. Although locking is used the method is deadlock free.

The proposed method provides as much concurrency as strict 2PL, which is more than the KO method.

The performance of ODL is compared with strict 2PL using timeout technique through simulation, and it is found out that except for the low-conflict cases, the ODL performs better than strict 2PL. In low-conflict cases, the throughput of the two methods are almost the same.

Acknowledgment

The authors would like to express their deepest thanks to M. Sungur and K. Tuzer for coding and experimenting the simulation models.

References

[1] R. Agrawal and D. J. DeWitt, "Integrated concurrency control and recovery mechanisms: Design and performance evaluation," *ACM Trans. Database Syst.*, vol. 10, pp. 529–564, Dec. 1985.

[2] R. Agrawal, M. J. Carey, and M. Linvy, "Concurrency control performance modelling: Alternatives and implications," *ACM Trans. Database Syst.*, vol. 12, pp. 609–654, Dec. 1987.

[3] D. Z. Badal, "Correctness of concurrency control and implication in distributed databases," in *Proc. COMPSAC '79*, 1979.

[4] R. Bayer, K. Eldhart, J. Heigert, and A. Reiser, "Dynamic timestamp allocation and its application to the BEHR method," Tech. Univ. Munich, Tech. Rep. TUM 18107, July 1981.

[5] P. A. Bernstein and N. Goodman, "Fundamental algorithms for concurrency control in distributed database systems," Computer Corp. of America, Tech. Rep., Feb. 1980.

[6] P. A. Bernstein, V. Hadzilacos, and N. Goddman, *Concurrency Control and Recovery in Database Systems*. Reading, MA: Addison–Wesley, 1986.

[7] C. Boksenbaum, M. Cart, J. Ferrie, and J. F. Pons, "Concurrent certifications by intervals of time stamps in distributed database systems," *IEEE Trans. Software Eng.*, vol. SE-13, pp. 409–419, Apr. 1987.

[8] M. Carey and M. Linvy, "Distributed concurrency control performance: A study of algorithms, distribution and replication," in *Proc. Int. Conf. Very Large Databases*, Aug. 1988.

[9] ——, "Parallelism and concurrency control performance in distributed database machines," in *Proc. SIGMOD Int. Conf. Management of Data*, 1989.

[10] W. Cellary, E. Gelenbe, and T. Morzy, *Concurrency Control in Distributed Database Systems*. Amsterdam:North Holland, 1988.

[11] S. Ceri and S. Owicki, "On the use of optimistic methods for concurrency control in distributed databases," in *Proc. 6th Berkeley Workshop on Distributed Data Management and Computer Networks*, 1982.

[12] S. Ceri and G. Pelagatti, *Distributed Databases Principles and Systems*. New York: McGraw-Hill, 1984.

[13] K. M. Chandy, L. M. Haas, and J. Misra, "Distributed deadlock detection," *ACM Trans. Computer Systems*, vol. 1, 1983.

[14] A. Dogac and U. Halici, "Timestamp transformation method for concurrency control in distributed DBMSs," in *Proc. Bilkent Symp. Computer and Information Sciences*, 1986.

[15] K. P. Eswaran, J. N. Gray, R. A. Lorie, and I. L. Traiger, "The notions of consistency and predicate locks in a database system," *Commun. ACM, Comput. Mach.*, vol. 19, pp. 624–633, Nov. 1976.

[16] P. Franaszek and J. T. Robinson, "Limitations on concurrency in transaction processing," *ACM Trans. Database Systems*, vol. 10, Mar. 1985.

[17] J. N. Gray, *Notes on Database Operating System, Operating Systems: An Advanced Course, Lecture Notes in Computer Science*. Berlin: Springer-Verlag, 1978.

[18] U. Halici, "Contributions to the theory of database concurrency control," Ph.D. dissertation, Dep. Electrical and Electronics Eng., Middle East Tech. Univ., Ankara, Turkey, Feb. 1988.

[19] U. Halici and A. Dogac, "Serializability of a set with a given ordering constraint and serializability criteria for concurrency control in database systems as constraints on transaction sets," Dep. Electrical and Electronics Eng., Middle East Tech. Univ., Ankara, Turkey, Tech. Rep. TR 87 4, Sept. 1987.

[20] ——, "A new method for concurrency control in distributed DBMSs: Permission test method," *ACM SIGMOD Record*, vol. 16, Sept. 1987.

[21] ——, "Class HD covers class WRW," , Dep. Electrical and Electronics Eng., Middle East Tech. Univ., Ankara, Tech. Rep. TR 87 6, Nov. 1987.

[22] ——, "Concurrency control in distributed databases through time intervals and short term locks," *IEEE Trans. Software Eng.*, vol. 15, pp. 994–1003, Aug. 1989.

[23] T. Hearder, "Observations on optimistic concurrency control schemes," *Inform. Syst.*, vol. 9, pp. 111–120, 1984.

[24] R. C. Holt, "Some deadlock properties of computer systems," *Computing Surveys*, vol. 4, pp. 179–196, Sept. 1972.

[25] T. Ibaraki, T. Kameda, and T. Minoura, "Serializability with constraints," *ACM Trans. Database Syst.*, vol. 12, pp. 429–452, Sept. 1987.

[26] H. T. Kung and J. T. Robinson, "On optimistic methods for concurrency control," *ACM Trans. Database Syst.*, vol. 6, pp. 213–226, June 1981.

[27] D. A. Menasce and T. Nakanishi, "Optimistic versus pessimistic concurrency control mechanism in database management systems," *Inform. Syst.*, vol. 7, pp. 13–27, 1982.

[28] C. Mohan, D. Haderle, B. Lindsay, H. Pirahesh, and P. Schwarz, "ARIES: A Transaction Recover Method Supporting Fine-Granularity Locking and Partial Rollbacks Using Write-Ahead Logging," IBM RJ 6649, Jan. 1989.

[29] C. Mohan, B. Lindsay, and R. Obermarck, "Transaction management in the R* distributed database management system," *ACM Trans. Database Syst.*, vol. 11, Dec. 1986.

[30] R. Obermarck, "Distributed deadlock detection algorithm," *ACM Trans. Database Syst.*, vol. 7, June 1982.

[31] T. Ozsu, "Modelling and analysis of distributed database concurrency algorithms using an extended petri formalism," *IEEE Trans. Software Eng.*, vol. SE-11, pp. 1225–1239, Oct. 1985.

[32] C. H. Papadimitriou, "The serializability of concurrent database updates," *J. ACM*, vol. 26, pp. 631–653, Oct. 1979.

[33] ——, *The Theory of Database Concurrency Control*. Rockville, MD: Computer Science Press, 1986.

[34] D. J. Rosenkrantz, R. E. Stearns, and P. M. Lewis, II, "System level concurrency control for distributed database systems," *ACM Trans. Database Syst.*, vol. 3, pp. 178–198, June 1978.

[35] B. Schlageter, "Optimistic methods for concurrency control in distributed database systems," in *Proc. Int. Conf. Very Large Databases*, pp. 125–130, Sept. 1981.

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

[37] M. Stonebreaker, "Concurrency control and consistency of multiple copies of data in distributed INGRES," *IEEE Trans Software Eng.*, vol. SE-5, May 1979.
[38] M. Sungur, K. Tuzer, and U. Halici, "Performance evaluation of database concurrency control techniques through simulation," Dep. Electrical and Electronics Eng., Middle East Tech. Univ., Ankara, Turkey, Tech. Rep. TR-90-1, Jan. 1990.
[39] R. H. Thomas, "A majority concencus approach for multiple copy databases," *ACM Trans. Database Syst.*, vol. 4, pp. 180–209, June 1979.

Ugur Halici (M'89) received the B.Sc., M.Sc., and Ph.D. degrees in electrical and electronics engineering from the Middle East Technical University, Ankara, Turkey, in 1980, 1983, and 1988, respectively.

She is presently an Associate Professor at the Department of Electrical and Electronics Engineering of the Middle East Technical University. Her research interests include concurrency control, fault detection, neural networks, and computational complexity.

Dr. Halici has worked as the Organizing Chairperson for the establishment of the IEEE Computer Society Chapter of the IEEE Turkey Section.

Asuman Dogac (S'85–M'86) received the B.Sc. degree in electrical and electronics engineering and the M.Sc. and Ph.D. in computer engineering from the Middle East Technical University, Ankara, Turkey, in 1973, 1975, and 1980, respectively.

She is presently a Full Professor at the Department of Computer Engineering of the Middle East Technical University. Her research interests include object oriented programming languages and database systems, expert systems, and distributed databases.