OREGON STATE UNIVERSITY COMPUTER SCIENCE DEPARTMENT

True-Copy Token Scheme for a Distributed Database System

Toshimi Minoura
Department of Computer Science
Oregon State University
Corvallis, Oregon 97331-4602

Susan S. Owicki
Computer Systems Laboratory
Department of Electrical Engineering
Stanford University
Stanford, California 94305

Gio Wiederhold
Computer Science Department
Stanford University
Stanford, California 94305

83-60-1

True-Copy Token Scheme for a Distributed Database System

Toshimi Minoura

Department of Computer Science
Oregon State University
Corvallis, Oregon 97331-4602

Susan S. Owicki

Computer Systems Laboratory
Department of Electrical Engineering
Stanford University
Stanford, California 94305

Gio Wiederhold

Computer Science Department
Stanford University
Stanford, California 94305

This work was partially supported by the Air Force Office of Scientific Research under Contract F49620-77-C-0045, by the Joint Services Electronics Program under Contract DAAG29-79-C-0047, and by the Defence Advanced Research Projects Agency under Contract N00039-80-G-0132.

True-Copy Token Scheme for a Distributed Database System

Abstract

A concurrency and resiliency control scheme for a distributed database system with replicated data is discussed. The scheme, true-copy token scheme, uses true-copy tokens in order to designate the physical data copies (true copies) that can be identified with the current logical data that are globally unique, and then it realizes consistent execution of transactions by the locking over these true copies. If subsystem failures occur and if some true copies are lost, the scheme regenerates lost true copies so that their continuity is preserved.

In analyzing the true-copy token scheme, we establish a precise relationship between physical transactions and their corresponding logical transactions by data and time abstraction. Then we show that continuity of logical data is preserved if continuity of true copies is preserved.

Key Words and Phrases: distributed database system, concurrency control, resiliency control, replicated data, true-copy tokens, data abstraction, time abstraction

---

In the final manuscript, unusual notations will be changed to normal ones, subscripts will be properly marked, and figures of better quality will be provided.

1. Introduction

Distributed database systems, where data are stored at multiple sites, are gaining importance because of the recent advances in hardware and computer communications technologies. Among the expected advantages of the decentralization of data, more specifically, partitioning and replication of data, are more efficient local processing and higher reliability. However, it is desirable that data partitioning and replication are hidden from the users of a distributed database system. Also, when multiple transactions are processed concurrently, their effects on the users must be as if they were processed one at a time. Further, the system must provide logically continuous operation even if some subsystems fail. Without reasonable concurrency and resiliency control, application of distributed database systems will be limited.

This chapter presents a new concurrency and resiliency control scheme that handles replicated data. The scheme first establishes logical data by hiding replication of physical data. At the highest level, a distributed database system is perceived as a collection of logical data that are not bound to particular sites. These logical data are then represented by physical data that are bound to sites. The main feature of the new scheme is the use of true-copy tokens which designate the physical data that can be identified with the current logical data. (Physical data copies that can be identified with the curent logical data are called true copies.) The concept of logical data is crucial in making the new scheme resilient, since resilient system operation can be realized

if the continuity of the logical data is preserved in the face of subsystem failures.

One way of guaranteeing the uniqueness of a logical datum is to let its single copy circulate in the system like the "control token" in [LELA-78] or the "hopping permit" in [LEE-80]. Another way is to designate a primary site for each logical datum that is represented by multiple physical copies [ALSB-76, STON-79]. The new concurrency control scheme extends these ideas and allows, for each logical datum, either one exclusive copy that can be accessed for read-write purposes or multiple share copies that can be accessed for read-only purposes (multiple share copies for the same logical datum must possess the same content). An exclusive copy can be split into multiple share copies, and these multiple share copies can be revoked to create a single exclusive copy again. Then, in order to realize consistent transaction processing, two-phase locking is applied on these exclusive or share copies (true copies) after transferring them to the sites where they are accessed.

One of the correctness proofs of the new concurrency control scheme is performed by using a technique of data abstraction analysis. The approach is basically the one developed for sequential programs [GUTT-78, HOAR-72, WULF-76]. First, that logical data are correctly implemented by the true-copy token mechanism is shown. Then, the correctness of the scheme is discussed solely at the logical data level. One new aspect of our proof is that execution timings of logical (abstract) operations are pinpointed on the

time axis. (This technique is called time abstraction.) Thus, we establish a precise relationship between physical transactions and their corresponding logical transactions.

Migration of exclusive copies may allow efficient local processing of transactions in a batch. Multiple share copies are useful for data that are occasionally updated but are mostly used for read-only purposes by many users. However, the main benefit of migrating true copies is that the resiliency problem can be discussed within this framework. That is, logically continuous system operation can be realized as long as continuity of true copies are preserved.

Based on the above principle, a simple resiliency scheme that combines a reliable storage mechanism and a reliable message transmission mechanism is designed. One unique feature of the scheme is that once true copies are reliably transferred between sites, each site can employ the same reliable storage subsystem as one for a centralized database system in order to support the true copies residing at that site, including those brought in from other sites.

A model of a distributed database system is given in Section 2. The concurrency control part of the new scheme is discussed in Section 3. Section 4 establishes the relationship between physical transactions and their corresponding logical transactions. The resiliency control part of the new scheme is discussed in Section 5. Section 6 concludes this chapter.

2. Distributed Database System Model

In this section we discuss the model of a distributed database system used in this chapter. The key aspect of the model is the notion of logical objects.

2.1 Logical Objects, Physical Objects and Sites

In our model a distributed database system consists of a set of logical objects, a set of physical objects, a set of sites, and a set of transactions.

Each logical object X, to which an independent value can be assigned, is represented by multiple (replicate) physical objects x1, ..., xk for some k that assume the same value except during the transitional periods while update operations on them are in progress.

An object is a data container that can store a data value. An object can be characterized by read and write operations applied to it.

Definition (Object). The data value of an object can be changed only by a write operation, and when a write operation is applied to it, the data value specified by the write operation will become the current data value of the object. When a read operation is applied to an object, the current data value of the object will be returned.

In other words, a read operation applied to an object returns

the data value written by the latest write operation applied to the object. However, there are some subtle points about logical objects, and they will be more precisely discussed in Section 4.

Each site of a distributed database system accommodates a subset of the set of physical objects in the system, and each physical object belongs to exactly one site. An example of a distributed database system is shown in Fig. 1. The system DDBS consists of three logical objects X, Y and Z. X is fully replicated at all sites H, I and J, Y is partially replicated at sites H and I, and Z has a single physical representation at site H.

Here, the terms "logical" and "physical" are used to indicate only a relative degree of abstraction. "Physical" does not mean direct implementation by hardware; a "physical object" may be a "logical object" at another level of abstraction. The important fact, however, is that in our model logical objects are globally unique entities and they are not bound to any particular sites.

In this chapter sometimes we will denote a distributed database system by its set of logical objects, and a site by its set of physical objects.

2.2 Transactions and Operations

A transaction is a set of operations. An operation is an activity that manipulates data or that coordinates the execution of other operations. We consider four types of operations, namely, read operations, write operations, local computations, and syn-

chronization operations.

Read and write operations, either logical or physical, are used to access objects, either logical or physical. Local computations of each transaction can transform the data read by read operations and supply the transformed data to write operations. Furthermore, data may be passed in the form of messages between two physical operations that occur at different sites.

A physical read operation read(xi) returns the current data value of physical object xi, and a physical write operation write(xi) updates the current data value of physical object xi. Further, we will show in Section 4 that a logical read operation read(X) returns the current data value of logical object X, and that a logical write operation write(X) updates the current data value of logical object X.

At this point we mention the following relationship between logical operations and physical operations. A logical read operation read(X) corresponds to a physical read operation read(xi), some i, and a logical write operation write(X) corresponds to the set of physical write operations {write(x1), write(x2), ... }, each of which writes the same data value as write(X). Now the reader is cautioned that we are postponing a precise definition of the execution timings of logical operations until we introduce the true-copy token scheme in later sections.

A physical write operation applied to a physical object at a site other than the one where it is created is called a remote

update. Usually a transaction creates physical write operations at the site where most of its local computations are performed.

Execution of read and write operations may be delayed by the system scheduler in order to avoid unacceptable intermixing of read and write operations of different transactions. A concurrency control scheme is the specification of the behavior of such a system scheduler. Synchronization operations (e.g., lock-seize and lock-release operations) may be used to coordinate the execution of other operations. Note that synchronization operations are used by transactions when part of the responsibility of the system scheduler is relegated to transactions.

In Fig. 2 we give an example of a transaction which runs on the distributed database system shown in Fig. 1. The transaction is described in two ways, as a logical transaction and as a physical transaction.

Concurrency control problems can be discussed either at the physical object level or at the logical object level. This differentiation must be made in terms of operations. Treatment at the logical object level gives direction to the handling of replicate physical objects, especially in the presence of subsystem failures.

Data values created for logical (and hence physical) objects can be identified as versions. Each logical object must initially contain version zero, and each time a logical object value is updated, its version number must be incremented by one. This version number can be assigned to each of the updates sent to the

replicate physical objects associated with the logical object.

A write operation (typically a remote update to a replicate physical object) is redundant if the data value written by it is overwritten by another write operation without being read by any read operation. Redundant write operations may be omitted. By properly ignoring redundant physical write operations, the inter-site traffic can be reduced, and efficiency of system operation can be enhanced. Redundant physical write operations ~~is~~ are often automatically eliminated if only the latest version of each logical object is transferred between sites.

2.3 Execution Sequences and Consistency Condition

The execution of an operation A, either logical or physical, is characterized by the occurrences of its initiation event "a" and its termination event "a", which we call operation events. In fact, defining these events for a logical operation is not trivial as we discuss in Section 4; however, at this point we assume that they are given a priori.

Definition (Execution Sequence <<). An execution sequence << for a set of transactions is a total ordering on the set of operation events caused by the execution of these transactions: for two operation events a and b, a << b iff a precedes b according to global time.

Now we discuss a criterion for correct system operation. Subsystem failures are not considered until Section 5. The defini-

tions in this section are applicable either at the physical or at the logical object level. Informally, an execution sequence of transactions is serializable if the effects on the users are as if transactions were processed one at a time. Papadimitriou has given a precise condition that characterizes the maximal set of serializable execution sequences for a set of transactions [PAPA-79]. He also has shown that the serializability test of an execution sequence in general is NP-complete.

Other authors [ESWA-76, MINO-78, SCHL-78, STEA-76] have used a stricter condition (class CPSR in [BERN-79] or class DSR in [PAPA-79]) that can be tested in polynomial time. We will use this stricter condition in our proofs.

Before we proceed, we need some definitions.

Definition (CONFLICT). The binary relation CONFLICT on a set of operations, either logical or physical, is defined as

CONFLICT = { (A, B) | operations A and B belong to different transactions and access the same object, and furthermore one of them is a write operation }.

We say that operations A and B conflict over an object x, either logical or physical, when (A, B) is in CONFLICT and x is accessed by A and B. Further, we say that two transactions conflict when they contain conflicting operations. Conflicting operations will induce the following ordering on the set of transactions.

Definition (D-Precedence <<<). Given an execution sequence << for a set of transactions, the D-precedence <<< on the set of transactions is defined as

<<< = { (R, S) | (A, B) is in CONFLICT and a << b
for some operations A of transaction R
and B of transaction S }.

We now define the consistency condition used in our proofs.

Definition (Consistency Condition C). Consistency condition C is true if D-precedence <<< is acyclic.

Intuitively, this definition means that in an execution sequence that satisfies consistency condition C, conflicting operations must define a consistent (non-circular) ordering on the transactions to which they belong. The condition that conflicting operations cannot occur concurrently is implicit in the above definition. In this chapter, a consistent execution sequence is one that satisfies consistency condition C, and a concurrency control scheme is consistent if any execution sequence allowed by the scheme is consistent.

In a consistent execution sequence, conflicting operations are generally not allowed to occur concurrently. Therefore, in later sections we use the following shorthand notation: we write A << B instead of a << b for two conflicting operations A and B.

2.4 Subsystem Failures

We further assume that each site of a distributed database system consists of a mainframe and a secondary storage subsystem. Data at each site are held either on the main memory within the mainframe or in the secondary storage subsystem. The exact locations of site data are irrelevant while the site is operating normally. If the mainframe fails, the data on the main memory will be lost. The data held by the secondary storage subsystem may be damaged when the secondary storage subsystem fails. Further, the data held by the secondary storage subsystem may be damaged also when the mainframe fails. The extent of tolerable damages to site data is discussed in Section 5.

We assume that a message can be sent from one site to another only when the message-link connecting them is normally functioning. When message link failures occur to a distributed database system, the system may be divided into multiple partitions. Each partition is the largest group of sites such that bidirectional communication is still possible between any pair of sites in it. That is, bidirectional communication becomes impossible for two sites that belong to different partitions.

3. Concurrency Control

In this section we describe without considering subsystem failures the concurrency control part of the true-copy token scheme, and then we prove that any execution sequence allowed by the scheme satisfies consistency condition C at the physical object level.

3.1 General Description

We first give an intuitive description of the true-copy token scheme. The new concurrency control scheme first establishes true copies that can be identified with the logical objects, and then performs locking over these true copies.

One way of guaranteeing the uniqueness of each logical object is to designate a unique physical copy (true copy) that can be identified with the logical object. In the "primary site" scheme [STON-79, ALSB-76], a physical copy at the primary site of each logical object is the true copy of that logical object. In the "hopping permit" scheme [LEE-80], the "hopping permit" designates the current true copy of the single logical object in the system. In [LELA-78] a "control token" which itself is a globally unique entity is used to issue "tickets" that uniquely order transactions. The true-copy token scheme extends these ideas.

Consider a particular logical object. When the logical object is used for read-write purposes, consistency will be violated if more than one physical copy is independently accessed at a different site. However, when the logical object is used for read-

only purposes, multiple physical copies can be allowed at different sites as long as their contents are the same. Considering these two cases, we can switch between a single read-write copy (exclusive copy) and multiple read-only copies (share copies) according to the need.

Fig. 3 shows how an exclusive copy or multiple share copies can occur for a logical object at each given time. At times t1, t3 and t4 only one exclusive copy exists in the system; however, at time t2 two share copies and at time t5 three share copies exist in the system.

A physical object can contain either an exclusive copy, share copy, or a void copy. Exclusive copies and share copies are called true copies, and we will show in Section 4 that their data values are identical to the current data values of their associated logical objects. The content of a void copy may be obsolete. Read-write accesses are allowed to exclusive copies and read-only accesses are allowed to share copies, but void copies are not accessed for normal transaction processing. To visualize the transfer of time-dependent access rights, we assume that a true copy possesses a true-copy token, i.e., either an exclusive-copy token or a share-copy token.

True-copy tokens handle replicate physical objects that cannot be handled efficiently by a locking mechanism alone. We can conceive that exclusive-copy tokens and share-copy tokens are spatial extensions of exclusive locks and share locks.

Two types of locks, namely share locks and exclusive locks, are used over the true copies to realize consistent transaction processing. A transaction accessing a logical object needs to lock one of its replicate physical objects that contains a true copy. An exclusive copy can be exclusively locked and a share copy can be share locked; furthermore, locking must be two-phase. Although two-phase locking is used, it is not a complete locking at the physical object level. Write operations to physical objects at remote sites are performed without locking.

A transaction issues a lock-seize request in order to set a lock in a desired mode. A lock can be reset by a lock-release request. A lock once set is active until it is reset. Now two-phase locking can be defined as follows [ESWA-76].

Definition (Two-Phase Locking). A transaction is two-phase locked if no lock-release requests are issued by the transaction before all lock-seize requests of that transaction are issued.

Remote updates can be performed either

a) by carrying ~~each~~ the current data value of ~~a~~ each logical object with its true-copy token(s), or

b) by letting the exclusive-copy token issue a series of version numbers that are unique relative to each logical object and by performing remote updates according to these version numbers.

Updates for a logical object originate only from its exclusive copy, and they can be uniquely ordered by their version numbers.

Also, although a remote update to a replicate physical object formally belongs to some transaction, it is more appropriate to consider that remote updates are carried out by the system when one of the above two methods is employed. When remote updates are performed by the system, a transaction need to submit only one data value in order to update the set of replicate physical objects belonging to a logical object.

3.2 Concurrency Control Scheme

We now state the true-copy token scheme more precisely. The new scheme allows several variations, so we state only the basic rules that must be observed in any implementation. Note that we are not giving an algorithm that describes every required procedure, and that we are instead giving a set of constraints that are necessary to prevent inconsistent system operation. A positive aspect of this approach is that any implementation that satisfies the basic rules given in this section will possess the properties discussed in Subsection 3.3 and Section 4. Two different remote update mechanisms and various true-copy token transfer methods are discussed in [MINO-80].

We assume that if a deadlock occurs, one or more of the transactions causing the deadlock are preempted in a way that the preempted transactions do not leave any effect to the system. In order to avoid the "domino effect" of transaction abortion, the structure of each transaction may have to be restricted [BAYE-80, MINO-80]. Further discussions concerning deadlocks in the true-

copy token scheme can be found in [MINO-80].

As we stated in the preceding section, a true copy is assumed to possess a true-copy token that specifies the type of the true copy, either an exclusive copy or a share copy. When a true-copy token is "transferred" from a physical object xi to another physical object xj, xi ceases to contain a true copy, and then xj starts to contain a true copy specified by the true-copy token. (In an actual implementation this time precedence can be established by passing a descriptor representing the true-copy token between the sites where those two physical objects reside.) A true copy is said to be free if it is not locked by any transaction.

Now we state the basic rules of the true-copy token scheme.

T1. Each physical object contains either an exclusive copy or a share copy or a void copy of the logical object to which the physical object belongs.

T2. At the point of system creation there exists exactly one free exclusive copy for each logical object.

T3. At any given time, either one exclusive copy or only share copies exist for each logical object.

T4. When a physical object obtains a true-copy token, all remote updates created thus far for the physical object must have been completed before the content of the physical object becomes a true copy. Remote updates to each physical object must follow the order of their creation.

T5. A transaction that wants to make read-write accesses to a logical object X must exclusively lock the exclusive copy of X. An exclusive lock can be set only on a free exclusive copy. Once the exclusive copy is exclusively locked, the transaction can read from or write to it. When the transaction write to the exclusive copy of X, it must also create the remote updates for other physical objects belonging to X; the same data value must be supplied for all of the replicate physical objects of X. An exclusive-copy token cannot be transferred to another physical object or changed to a share-copy token while an exclusive copy is exclusively locked.

T6. A transaction that wants to make read-only accesses to a logical object X must share lock a share copy of X. A share lock can be set only on a share copy, which may have already been share locked. Once a share copy is share locked by a transaction, the content of the share copy can be read by the transaction. A share copy cannot be revoked until all share locks on it are released.

T7. Locking on true copies by a transaction must be two-phase.

Rule T3 may be replaced by the following three rules.

T3a. An exclusive-copy token can be transferred to another physical object belonging to the same logical object.

T3b. An exclusive-copy token can become a share-copy token. A share-copy token can create other share-copy tokens, and the newly created share-copy tokens can be granted to other

physical objects belonging to the same logical object.

T3c. When an exclusive copy of a logical object for which there currently exist multiple share copies is required, all of these share copies must be revoked except the one that becomes the exclusive copy.

Note that the invariant specified by rule T3 is not invalidated by any of the operations specified in the above three rules.

Rules T5 and T6 may be modified so that share locks can be applied to an exclusive copy. Then, an exclusive copy can be revoked only when it is neither exclusively nor share locked, and an exclusive copy that is only share locked can be changed to a share copy. This modification does not affect the set of transactions that can be executed concurrently.

Redundant remote updates may be discarded to reduce the inter-site traffic as we mentioned in Subsection 2.2. If more than one remote update occurs to the same physical object at some remote site before the physical object obtains a true copy, remote updates other than the last one are redundant, for the data values written by them will never be accessed.

Fig. 5 shows which combination of transactions shown in Fig. 4 can be processed concurrently. A transaction is "active" if it is being executed by using local data, and a transaction is "blocked" if it cannot be executed by using local data.

In Fig. 5(a), transaction P can proceed because xl contains an

exclusive copy, and y1 contains a share copy. Note that P makes read-write accesses to logical object X and a read-only access to logical object Y. The remote update to x2 by P can be discarded because it will be overwritten by the remote update by transaction Q; it is redundant.

Also in Fig. 5(a), transaction R tries to make read-write accesses to logical object Y. However, physical object y2 contains a share copy and not an exclusive copy, so R cannot exclusively lock y2 and is blocked.

Once P is completed at site H and transaction Q starts its execution using the exclusive copy in x1, the share-copy token of y1 can be released and the content of y2 can become an exclusive copy; then R can proceed. In Fig. 5(b), both Q and R are running concurrently. The remote update to x2 created by Q must be sent to site I before the content of x2 becomes an exclusive copy and is accessed by T.

In Fig. 5(c) transaction S introduced at site H is blocked because the content of y1 is not a share copy. In Fig. 5(d), however, two share copies, those in y1 and y2, exist for logical object Y at the same time, and both transactions S and T are active.

Now we briefly state how to perform remote updates so that they satisfy rule T4 given above. A straightforward method is to transfer true copies themselves among physical objects, i.e., to carry the latest data value created for each logical object within

its true-copy tokens and to update each physical object to the value specified by the true-copy token when it is visited by a true-copy token. In this method, a remote update addressed to each remote physical object is not created. Therefore, assume that remote updates are carried by true copies and that a remote update carried by a true copy is applied to its target physical object when the true copy arrives at the physical object for the first time after the remote update is created. Further assume that remote updates carried by a true copy are overwritten before they arrive at their target physical objects by their successive remote updates when newer values are created for the true copy.

The obvious disadvantage of this method is that we have to carry a lot of data when a logical object is large even when the modified part is small. Another disadvantage is that replicate physical objects are not kept up-to-date. Some advantages of this method are that it is easy to understand, that remote updates need not be addressed to individual physical objects, and that redundant remote updates are automatically ignored.

Another method to perform remote updates, "sequenced update mechanism", is described in [MINO-80]. This method can eliminate the two disadvantages associated with the previous method.

3.3 Correctness Proof

In this subsection we prove that the true-copy token scheme presented in the preceding section is consistent by directly showing that any execution sequence allowed by it satisfies consistency

condition C at the physical object level.

An immediate consequence of the two-phase locking rule (rule T7) is that all locks set by each transaction are active at some point during the execution of the transaction. We define the lock peak point Rp of a transaction R as the first point in time at which all locks set by transaction R are active.

Definition (Lock-Peak-Point Ordering <<p). The binary relation <<p on a set of transactions that use two-phase locking is defined as follows: R <<p S iff Rp << Sp, where Rp and Sp are the lock peak points of transactions R and S, respectively.

As Rp and Sp are not operation events, we have extended the execution sequence << to include lock peak points. Obviously, R <<p S means that the lock peak point of R precedes that of S. Note that <<p is acyclic because <<p is a total ordering.

We now prove that any execution sequence allowed by the true-copy token scheme satisfies consistency condition C at the physical object level. We will show that in any execution sequence D-precedence <<< defined on the set of transactions can be embedded in the lock-peak-point ordering <<p, i.e., that if R <<< S for any pair of transactions R and S, then R <<p S. In the following proof, operations are physical ones, and the execution timings of remote updates are those times when they are actually applied to physical objects. Also, a true copy is either an exclusive copy or a share copy, and an access is either a read operation or a write operation.

Theorem 1. The true-copy token scheme is consistent.

Proof. Assume that for a pair of transactions R and S, R <<< S. Then, some operation A of transaction R and some operation B of transaction S conflict over some physical object xk, and A << B. Accesses made by A and B to xk are either local accesses performed under locking while xk holds a true copy, or remote updates.

First, if xk is locally accessed by both A and B, R <<p S because local accesses are made with xk being locked.

Second, if A is a local access and B is a remote update, then the remote update B must be preceded in S by a local write operation C that writes the same data value as B to some physical object xi (i /= k) that contains an exclusive copy when C is applied to it (Fig. 6(a)). Now if C << A, then B << A because B must be applied to xk before xk obtains the true copy accessed by A (rule T4). Therefore, A << C, and hence R <<p S because xi can seize the exclusive copy accessed by C only after the true copy accessed by A is released.

Third, if A is a remote update and B is a local access, A must be preceded in R by a local write operation C that accesses xi writing the same value as A (Fig. 6(b)). As C precedes A and A precedes B, C precedes B and hence R <<p S. Note that S can exclusively or share lock xk only after the exclusive lock on xi is released by R.

Finally, if accesses by both operations A and B are remote updates (Fig. 6(c)), the ordering rule of remote updates (rule T4)

guarantees that these remote updates are performed in their order of creation. Hence, transaction R exclusively locks some physical object xi and creates the remote update A before transaction S exclusively locks some physical object xj and creates the remote update B. Consequently, R <<p S.

We have shown that conflicting operations are performed according to the lock-peak-point ordering <<p defined for the transactions to which they belong, i.e., that if R <<< S for transactions R and S, then R <<p S. Since <<p is acyclic, <<< is acyclic, and hence consistency condition C is satisfied. Q.E.D.

Numbers in Fig. 6 indicate the event ordering. In Fig. 6(a) (Fig. 6(b)), the true-copy token that is transferred from xk (xi) to xi (xk) may be used at other sites after it leaves xk (xi) and before it reaches xi (xk). In Fig. 6(c), that xi = xj is allowed.

4. Logical Objects and Logical Operations

In the preceding section we have shown that the true-copy token scheme maintains consistency condition C at the physical object level. From the standpoint of system structuring, however, it is desirable to assume that logical objects themselves hold data values.

In this section we show that we can consider that in the true-copy token scheme transactions access logical objects as if they were real entities. Our discussion follows the standard approach of handling abstract data in sequential programs [HOAR-72, GUTT-78, WULF-76] except that our proof is informal.

First, a data abstraction function for the current data values of logical objects is defined. Then, we show that logical operations cause the expected effects on the logical object values. A new aspect of our proof method is that logical operations are considered instantaneous, and (abstract) execution timings are defined for them. Once logical objects and logical operations are established, that the two-phase locking rule is observed by logical operations is shown.

Before we proceed, we need a minor abstraction step. Assume that an execution sequence at the physical object level realized by the true-copy token scheme is given. In order to make our discussion simple, we reinterpret the given execution sequence in the following way.

1. Each physical operation takes place instantaneously at the point when the actual physical operation is initiated.

2. Each true-copy token transfer occurs instantaneously at the point when the new holder of the true-copy token receives it.

Note that these changes are applicable only to our perception of the system operation and not to the actual operation of the system. The first interpretation is allowable because no conflicting physical operations can occur concurrently, and the second interpretation is allowable because no transactions can access true copies while they are being transferred.

Now we can define the data abstraction function for logical object values. The following definition states that in the true-copy token scheme, logical objects and true copies can be "identified". Note that logical objects are (fictitious) entities at the logical object level whereas true copies are entities at the physical object level.

Definition (Logical Object Value). The current data value of each logical object is specified by the current content of either an exclusive copy or a share copy associated with the logical object.

Lemma 2. The current data value of each logical object is uniquely defined at any time.

Proof. When there is a single exclusive copy for a logical object, obviously its data value is uniquely defined. When a share copy creates other share copies, these share copies reflect all past

updates created for them, and hence their contents are the same. Furthermore, the contents of share copies will not change until they are revoked and an exclusive copy is created. Hence, even when there are multiple share copies, their contents are the same.

Q.E.D.

We now want to show that in the true-copy token scheme logical read and write operations access logical objects as if they were real entities. In order to make the definition of logical objects meaningful (the definition of an object was given in Subsection 2.1), an access ordering must be specified on the logical read and write operations applied to each logical object.

We now define the execution timings of logical operations. In the case of physical operations, we have conceived that they are activities that actually take place on the hardware and that their execution timings can be defined unambiguously. The definition of the execution timings of logical operations, on the other hand, requires discretion.

As we stated in Subsection 2.2, a logical read operation read(X) is represented by a physical read operation read(xi) for some xi. Therefore, it is natural to identify the execution timing of read(X) with that of read(xi).

The problem is not so simple for a logical write operation write(X) because in our model write(X) is represented by the set of physical write operations {write(x1), write(x2), ... }. However, the underlying concept of the true-copy token scheme is to identify

true copies with logical objects. Thus, it is natural to identify the execution timing of a logical write operation write(X) with that of the physical write operation write(xi) that is applied to the physical object containing the exclusive copy of logical object X. In the true-copy token scheme, only one operation in the set of physical write operations representing write(X) is applied to an exclusive copy. Therefore, the execution timing of write(X) is also uniquely defined. The reader is cautioned that this definition is applicable only when the true-copy token scheme is used. That is, different definitions must be given for other schemes.

Definition (Execution Timings of Logical Operations). Given a physical level execution sequence of transactions realized by the true-copy token scheme, the execution timings of logical operations are defined as follows.

L1. Assume that logical read operation read(X) occurs when its corresponding physical read operation read(xi), some i, occurs.

L2. Assume that logical write operation write(X) occurs when one of its corresponding physical write operations is applied to the exclusive copy of X.

L3. Assume that the execution timing of a lock-seize or lock-release operation on a logical object is that of its corresponding lock-seize or lock-release operation applied to a true copy associated with the logical object.

Now we show that the logical execution sequence thus defined

is a consistent way of viewing the physical one.

Lemma 3. When transactions are executed by using the true-copy token scheme, transactions will not see any difference even if physical objects and physical operations are replaced by their logical counterparts, i.e., logical objects and logical operations.

Proof. First, the data value returned by the physical read operation read(xi) representing a logical read operation read(X) is the data value of a true copy for X, and is by definition the current data value of logical object X. Hence, read(xi) can be replaced by read (X) without affecting the transaction issuing it.

Second, when a logical write operation write(X) is assumed to occur, one of its corresponding physical write operations is applied to the exclusive copy of X, and consequently the current data value of X changes as specified by write(X). Hence, write (X) is correctly implemented.

Third, we show that the data values of the logical objects are not changed by other physical operations. Remote updates have no effects on the current data values of logical objects because they are applied to void copies. Further, transfer of a true-copy token has no effect on the current data value of the logical object associated with the true-copy token, since two physical objects between which a true-copy token is tranferred possess the same content when the true-copy token transfer occurrs. This is because all past remote updates are applied to a physical object when it obtains a true copy (rule T4).

Consequently, in the true-copy token scheme we can assume that logical objects are accessed as if they were real objects. Q.E.D.

Lemma 4. In the true-copy token scheme, two-phase locking is realized over logical objects.

Proof. First, we show that conflicting lock instances are mutually excluded at the logical object level. If a logical object is exclusively locked by some transaction, the exclusive copy of the logical object is exclusively locked at some site. Then no other transactions can lock a true copy belonging to the same logical object, for the exclusive copy that is exclusively locked is the only true copy of the logical object. On the other hand, if a logical object is share locked, a share copy of the logical object is share locked at some site. Then, only share copies can exist for that logical object, and no transaction can exclusively lock the logical object because share copies cannot be exclusively locked.

Second, by definition the timing of the locking on logical objects is identical to the timing of the two-phase locking applied to true-copies (rule T7). Q.E.D.

It is well known that two-phase locking applied to a centralized database system preserves consistency condition C [BERN-79, ESWA-76, PAPA-79]. Hence, from Lemmas 3 and 4 we can conclude the following theorem.

Theorem 5. Any execution sequence allowed by the true-copy token scheme satisfies consistency condition C at the logical object

level.

5. Resiliency Control

In this section we first give a definition of resilient system operation. Then, we show that the true-copy token scheme can be made resilient by employing a reliable storage mechanism and a reliable message transmission mechanism. Finally, system partitioning problem is discussed. The resiliency scheme discussed in this section can handle site crashes and message link failures as long as each site can always restore its own data. However, the scheme fails when some sites cannot restore their own data.

Resiliency schemes that can tolerate complete failures of some sites in distributed database systems are described in [MINO-82, MINO-83]. Since the resiliency scheme discussed in this section is simpler and more efficient than those schemes, a sensible approach would be to employ those schemes only when the scheme discussed in this section fails.

5.1 Resilient System Operation

So far we have not considered site or message-link failures. When such subsystem failures occur, continuity of data may be lost. However, when the true-copy token scheme is employed, continuity of logical object values can be preserved by preserving continuity of true copies.

Lemma 6. Logical operations will not be affected if true copies are regenerated with the values possessed by lost true copies.

Proof. If a true copy is regenerated with the value possessed by

the lost true copy, then the proof of Lemma 3 is still valid. The combined process of the loss and the regeneration of a true copy of logical object X preserves the value of X. Note that logical objects whose true copies are lost are not accessed until those lost true copies are regenerated. Q.E.D.

When the mainframe at a site fails, some transactions being executed at that site may not be completed. Then, these transactions must be aborted, and their effects must be nullified. Transactions that are not aborted must be committed. The effect of a committed transaction must be complete.

We now give the definition of resilient system operation.

Definition (Resilient System Operation). System operation is resilient if the following three conditions are satisfied:

R1. Each transaction submitted to the system is either committed or aborted. Each logical write operation issued by a committed transaction updates the value of its target logical object, and no logical write operations issued by aborted transactions affect the logical object values.

R2. Each read operation issued by a committed transaction returns the current value of its target logical object.

R3. The execution sequence defined for committed transactions is serializable at the logical object level.

When condition R1 is satisfied, we say that atomicity of each transaction is preserved. Condition R2 does not impose any

requirement on the values read by aborted transactions.

5.2 Reliable Storage Mechanism

In implementing a reliable distributed database system, we assume that each site is equipped with a reliable storage subsystem. A site that is guarded by a reliable storage subsystem can restore its own data even if its mainframe and part of its secondary storage fail. A site equipped with a reliable storage subsystem will be called a reliable site.

We assume that the reliable storage subsystem employed by each site possesses the following properties.

S1. Data written to the reliable storage will never be lost.

S2. A set of data can be atomically written to the reliable storage at each site.

S3. An individual write operation to reliable storage is relatively expensive. Therefore, it is impractical to move to reliable storage every piece of data generated by the mainframe.

A conventional method for making a centralized database system resilient against mainframe and disk failures is to use checkpoint dumps** and log records, which effectively provide reliable storage

---

** A checkpoint dump is a complete copy of a database state. In [GRAY-81], such a copy is called an archive, and the term "checkpoint" is used for a different purpose.

[GRAY-78, GRAY-81, FOSS-74, LIND-80, STON-76, VERH-78, WIED-83]. A reliable storage subsystem can also be implemented by duplicating a storage subsystem that uses the "intentions-list" mechanism [LAMP-79, LAMP-81, MENA-80] or the shadow mechanism [LORI-77]. Various methods for implementing reliable storage subsystems are compared in [LIU-83].

5.3 Reliable Token Transfer

We now consider the problem of connecting reliable sites in order to realize a reliable distributed database system. Since reliable sites will never lose their data, a reliable distributed database system can be constructed if true copies can be reliably transferred between these sites.

However, if a message link fails, true-copy tokens being transferred over the message link may be lost. When a site crash occurs, there is a possibility of losing messages on the crashed site; these messages may have been generated by the crashed site itself, or they may have been sent from other sites. Further, as we assume that true-copy tokens and locks are kept in main memory while a site is operating normally, they may be lost or released if a mainframe failure occurs.

We can realize reliable token transfer by retrasmitting possibly lost messages. (Message retransmission is widely practiced in communication networks [EDGE-78].) In implementing a reliable token transfer mechanism, we make the following assumptions.

M1. A damaged message can be always detected as damaged.

M2. If the same message is sent repeatedly some finite number of times, it will eventually reach its destination site.

M3. Duplicate messages can be detected.

Assumption M1 is based on the fact that the probability of an occurrence of undetectable error is extremely small when an appropriate error detection code is used. Assumption M2 implies another assumption that a failed message link will eventually be restored. Since relevant messages are associated with true-copy tokens, version numbers can be used to detect duplicate messages.

In order to transfer a true-copy token from a sender site to a receiver site both of which are operating normally, the following procedures must be used by reliable sites.

X1. Before the sender site releases the true-copy token, it must write to its reliable storage a Token-Release record remembering to which site the true-copy token is being sent, and then it must delete the Token-Seize record for the true-copy token. Once these operations are complete at the sender site, the true-copy token can be sent to the receiver site by a Token-Grant message.

X2. When the receiver site receives the Token-Grant message, it must write to its stable storage a Token-Seize record before the site begins to use the true copy associated with the true-copy token. Then, the receiver site must send a Token-

ACK message to the sender site acknowledging the receipt of the true-copy token.

X3. When the sender site receives the Token-ACK message from the receiver site, the Token-Release record at the sender site must be deleted.

Fig. 7 shows the normal interaction of the sender site and the receiver site.

In the procedures above, a true-copy token held by a site is remembered by the Token-Seize record written on the stable storage at that site, and a true-copy token being transferred is remembered by the Token-Release record written on the stable storage at the sender site. Step X2 can be modified so that the Token-ACK message is effectively returned to the sender site by returning the true-copy token itself after its use. This modification can reduce the number of required messages.

Now we state the additional procedures that are required when the system is restored from subsystem failures.

X4. When a site is restored from a site failure, it must take the following action. If a Token-Seize record is found at that site and if no Token-Release record for the same true-copy token is found, then the true-copy token indicated by the Token-Seize record must be regenerated at that site.

X5. If a Token-Release record is found at some site and if the true-copy token associated with the Token-Release record may

have been affected by the subsystem failures, then the Token-Grant message for the true-copy token indicated by the Token-Release record must be sent again to the receiver site. A true-copy token sent over a message link may be affected if any of the sender site, the message link, and the receiver site fails.

X6. When a redundant Token-Grant message is received for the true-copy token that has already been received, the message must be ignored except that a Token-ACK message must be sent back to the sender site even if it has already been sent.

X7. If a Token-ACK message whose matching Token-Release record does not exist is received, the message must be ignored.

When a Token-Release record is found at a site, a true-copy token has been sent to another site as indicated by the Token-Release record, but its acknowledgment has not been returned to the sender site yet. The true-copy token may have been lost while it was being transferred to the receiver site. Therefore, the true-copy token must be sent to the receiver site again. However, if the true-copy token has already reached the receiver site, the Token-Grant message must be ignored. Also, note that if both Token-Seize record and Token-Release record for the same true-copy token are found at a site, then the Token-Release record prevails.

5.4 Transaction Processing

We showed that true copies can be reliably transferred between

sites. If every true copy required by a transaction is transferred to the site where the transaction is executed, then the transaction can be executed as if it is executed by a centralized database system, and each site can use the reliable storage subsystem in order to support the true copies at that site.

As we have mentioned, a transaction is a collection of operations, and a transaction as seen by its user is like an ordinary program with write operations intermixed with other operations. However, we restrict the transaction structure seen by the system in order to support our resiliency scheme. The main feature of our restricted transaction structure is a transaction buffer that supports the abortion of a partially executed transaction without causing any ill effect to the system.

A transaction buffer is provided for each transaction. Until a transaction issues a Transaction-End command, updates created by the transaction are kept in its transaction buffer, and they are not applied to the reliable storage. This restriction allows transaction abortion at any time before the transaction issues the Transaction-End command. A transaction can be aborted simply by discarding its transaction buffer.

Once the Transaction-End command is issued, the updates in the transaction buffer must be written into the reliable storage. Therefore, all write operations of each transaction are effectively collected at the end of the transaction. A transaction cannot be aborted once the Transaction-End command is issued.

We now state how transactions can be executed by each site.

E1. In order that a transaction can be executed at a site, all of the true-copies accessed by the transaction must be available at that site. Therefore, some true copies must be brought in from other sites. If a physical object to accommodate a true copy does not exist at a site, a tentative physical object can be created in order to accommodate the true copy. Whenever a true copy is obtained by a site, the content of the true copy must be written to reliable storage before the true copy is used by the transactions at that site.

E2. Updates** created by each transaction must be atomically applied to the reliable storage at the site where the transaction is executed.***

E3. If a site failure occurs, the reliable storage subsystem at the failed site must restore a "transaction-consistent" [GRAY-81] state.

As we discussed, continuity of true copies are preserved when they are transferred between sites. If the reliable storage subsystem at each site works correctly, true copies at each site will

---

** The output of the transaction must be included in these updates. It can be released once the updates are moved to reliable storage. On the other hand, the output must always be produced if the updates are reflected on reliable storage.

*** Updates of transactions must be moved to reliable storage according to the "U-precedence" defined for those transactions [LIU-83, MINO-83]. This requirement can be automatically satisfied when exclusive locks are held until reliable storage is updated.

be effectively manipulated only by committed transactions (rules S1, S2, E2 and E3). Further, two-phase locking applied on true copies guarantee that the resultant execution sequence is serializable if aborted transactions are excluded. Consequently, conditions R1, R2 and R3 for resilient system operation are satisfied even if subsystem failures occur.

5.5 System Partitioning

While the system is operating normally, all sites in the system belong to one partition, and a true-copy token of each logical object can be transferred to any site where it is required.

When the system is partitioned, each partition will possess a subset of the true copies that exist in the system. System partitioning can be supported without any consistency problem by letting each transaction access only those true copies that are available within the partition where the transaction is executed. Note that a transaction cannot be executed if some true copies required by the transaction are not available within the partition where the transaction is executed, and that the set of transactions that are executable within each partition is partly decided by chance. Remote updates to physical objects at the sites within other partitions must be delayed until message-links are restored.

6. Conclusion

A new concurrency and resiliency control scheme, true-copy token scheme, was developed for a distributed database system with replicated data. The scheme first establishes logical objects by hiding replication of physical objects and then applies locking on these logical objects in order to preserve consistency. The behavior of the logical objects was precisely discussed by using the concept of data abstraction. In doing so, execution timings of logical operations were explicitly defined. This technique was called time abstraction.

The new concurrency control scheme supports for each logical object either one read-write copy or multiple read-only copies at one time. Multiple read-only copies that can be updated by revoking them will be useful because many files in real systems are used in this way.

We also showed that the true-copy token scheme can be made resilient by combining a reliable true-copy transfer mechanism and a reliable storage mechanism; the latter has long been used by a centralized database system. The resiliency scheme discussed in this chapter cannot handle complete site failures. Resiliency schemes that can restore a failed site by using redundatant data stored at other sites are disucssed in [MINO-82, MINO-83]. Even when those schemes are employed, The scheme discussed in this chapter can be used as the first level recovery scheme.

References

[ALSB-76] Alsberg, P.A., Belford, G.G., Day, J.D., and Grapa, E. Multi-copy resiliency techniques. Center for Advanced Computation, Univ. of Illinois, Urbana-Champaign, May 1976.

[BAYE-80] Bayer, R., Heller, H., and Reiser, A. Parallelism and recovery in database systems. ACM Tr. on Database Systems 5, 2 (June 1980), 139-156.

[BERN-79] Bernstein, P., Shipman, D., and Wong, W. Formal aspects of serializability in database concurrency control. IEEE Tr. on Software Engineering SE-5, 3 (May 1979), 203-216.

[EDGE-78] Edge, S.W. and Hinchley, A.J. A survey of end-to-end retransmission techniques. SIG Computer Communication Review 8, 4 (Oct. 1978), 1-16.

[ESWA-76] Eswaran, K., Gray, J., Lorie, R., and Traiger, I. The notions of consistency and predicate locks in a database system. CACM 19, 11 (Nov. 1976), 624-633.

[FOSS-74] Fossum, B.M. Data base integrity as provided by a particular data base management system. Data Base Management, Klimbie, J.W. and Koffeman, K.L. (eds.), North-Holland, 1974, pp. 271-288.

[GRAY-78] Gray, J. Notes on data base operating systems. In Lecture Notes in Computer Science 60, Springer-Verlag, 1978, pp. 393-481.

[GRAY-81] Gray, J., McJones, P., Blasgen, M., Lindsay, B., Lorie, R., Price, T., Putzolu, F., and Traiger, I. The recovery manager of the System R database manager. Computing Surveys 13, 2 (June 1981), 223-242.

[GUTT-78] Guttag, J.V., Horowitz, E., and Musser, D.R. Abstract data types and software validation. CACM 21, 12 (Dec. 1978), 1048-1064.

[HOAR-72] Hoare, C.A.R. Proof of correctness of data representations. Acta Informatica 1 (1972), 271-281.

[LAMP-79] Lampson, B.W. and Sturgis, H.E. Crash recovery in a distributed data storage system. Xerox Palo Alto Research Center, April 1979.

[LAMP-81] Lampson, B.W. Atomic transactions. In Distributed Systems -- Architecture and implementation, Lecture Notes in Computer Science 105, Springer-Verlag, 1981, pp. 246-265.

[LEE-80] Lee, C.H. Queueing analysis of global locking synchronization schemes for multicopy databases. IEEE Tr. on Computers C-29, 5 (May 1980), 371-384.

[LELA-78] Le Lann, G. Algorithms for distributed data-sharing systems which use tickets. Proc. 3rd Berkeley Workshop on Distributed Data Management and Computer Networks, Aug. 1978, pp. 259-272.

[LIND-80] Lindsay, B.G. Single and multi-site recovery facilities. In Distributed Data Bases, Draffan, I.W. and Poole, F. (eds.), Cambridge University Press, 1980, pp. 247-284.

[LIU-83] Liu, C. and Minoura, T. Reliable Storage Update. Submitted for publication.

[LORI-77] Lorie, R.A. Physical integrity in a large segmented database. ACM Tr. on Database Systems 2, 1 (March 1977), 91-104.

[MENA-80] Menasce, D.A. and Landes, O.E. On the design of a reliable storage component for distributed database management systems. Proc. 6th Int. Conf. on Very Large Data Bases, 1980, pp. 365-375.

[MINO-78] Minoura, T. Maximally concurrent transaction processing. Proc. 3rd Berkeley Workshop on Distributed Data Management and Computer Networks, Aug. 1978, pp. 206-214.

[MINO-80] Minoura, T. Resilient extended true-copy token algorithm for distributed database systems. Ph.D. Thesis, Stanford University, May 1980. (available from University Microfilms International, 300 N Zeeb Road, Ann Arbor, MI 48106) Also TR-197, CSL, Stanford University (microfiche only).

[MINO-82] Minoura, T. and Wiederhold, G. Resilient extended true-copy token scheme for a distributed database system. IEEE Tr. on Software Engineering SE-8, 3 (May 1982), 173-189.

[MINO-83] Minoura, T., Owicki, S. and Wiederhold, G. Consistent distributed database state maintenance. Submitted for publication.

[PAPA-79] Papadimitriou, C.H. The serializability of concurrent database updates. JACM 26, 4 (Oct. 1979), 631-653.

[SCHL-78] Schlageter, G. Process synchronization in database systems. ACM Tr. on Database Systems 3, 3 (Sept. 1978), 248-271.

[STEA-76] Stearns, R., Lewis, P., and Rosenkrantz, D. Concurrency control for database systems. Proc. IEEE Symp. on Foundations of Comp. Sci., Oct. 1976, pp. 19-32.

[STON-76] Stonebraker, M, Wong, E., and Kreps, P. The design and implementation of INGRES. ACM Tr. on Database Systems 1, 3 (Sept. 1976), 189-222.

[STON-79] Stonebraker, M. Concurrency control and consistency of multiple copies of data in distributed INGRES. IEEE Tr. on Software Engineering SE-5, 3 (May 1979), 188-194.

[VERH-78] Verhofstad, J.S.M. Recovery techniques for database systems. ACM Computing Surveys 10, 2 (June 1978), 167-195.

[WIED-83] Wiederhold, G. Database Design, 2nd edition. McGraw-Hill, 1983.

[WULF-76] Wulf, W.A., London, R.L., and Shaw, M. An introduction to the construction and verification of Alphard programs. IEEE Tr. on Software Eng. SE-2, 4 (Dec. 1976), 253-265.

DDBS = {X, Y, Z}

logical components:	sites:
X = {x1, x2, x3}	H = {x1, y1, z1}
Y = {y1, y2}	I = {x2, y2}
Z = {z1}	J = {x3}

Fig. 1. A distributed database system.

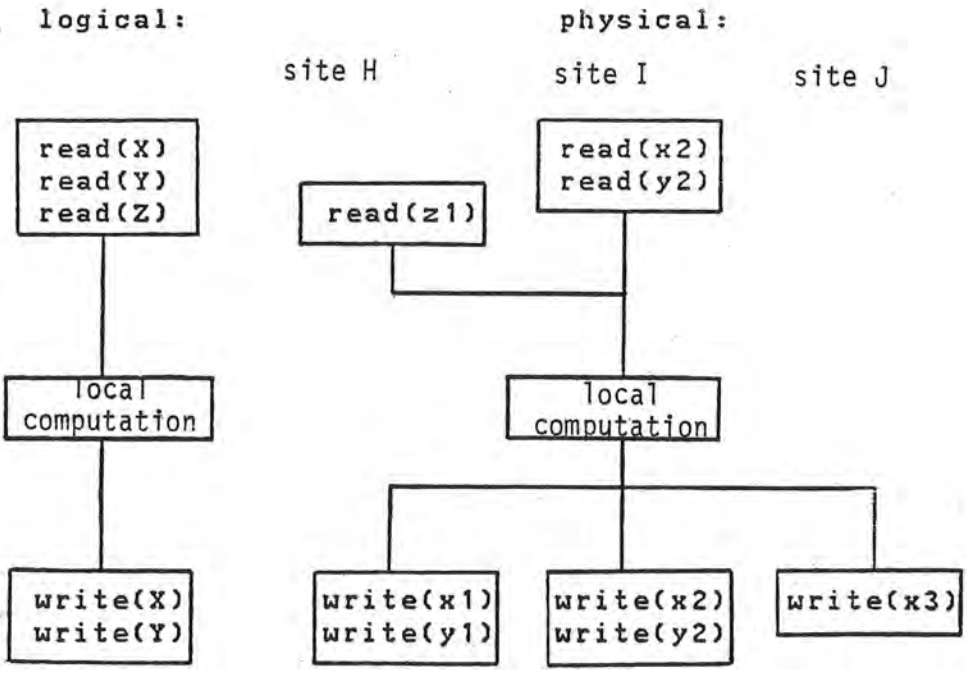
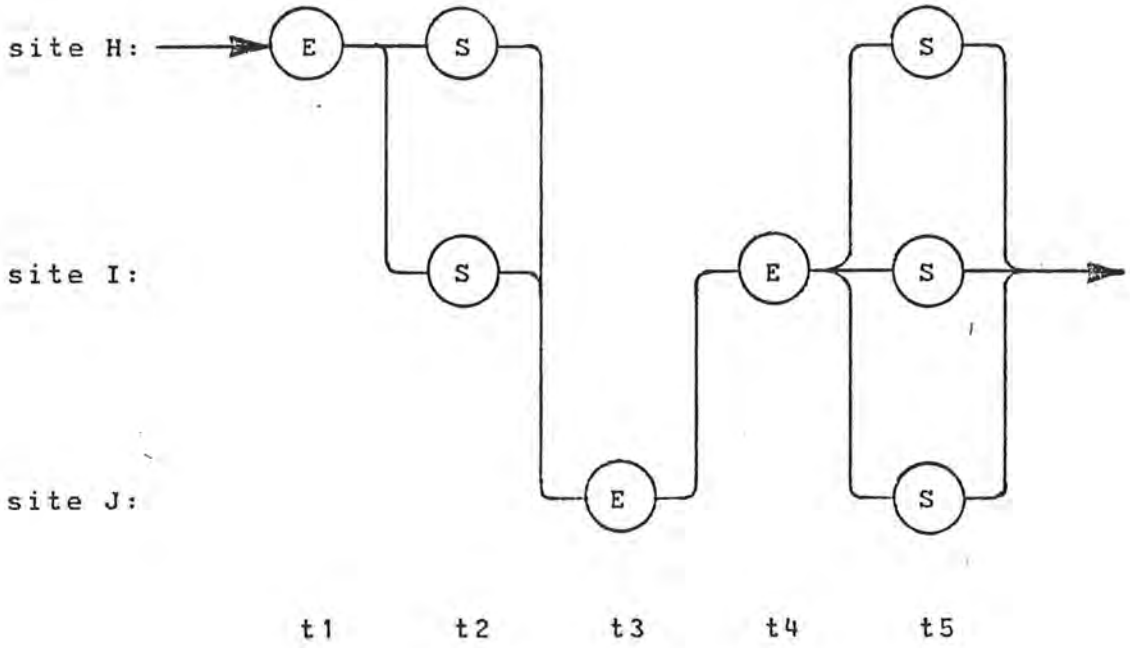


Fig. 2. Transaction.



E : exclusive copy
S : share copy

Fig. 3. True copies.

logical components:

X = {x1, x2}
Y = {y1, y2}

sites:

H = {x1, y1}
I = {x2, y2}

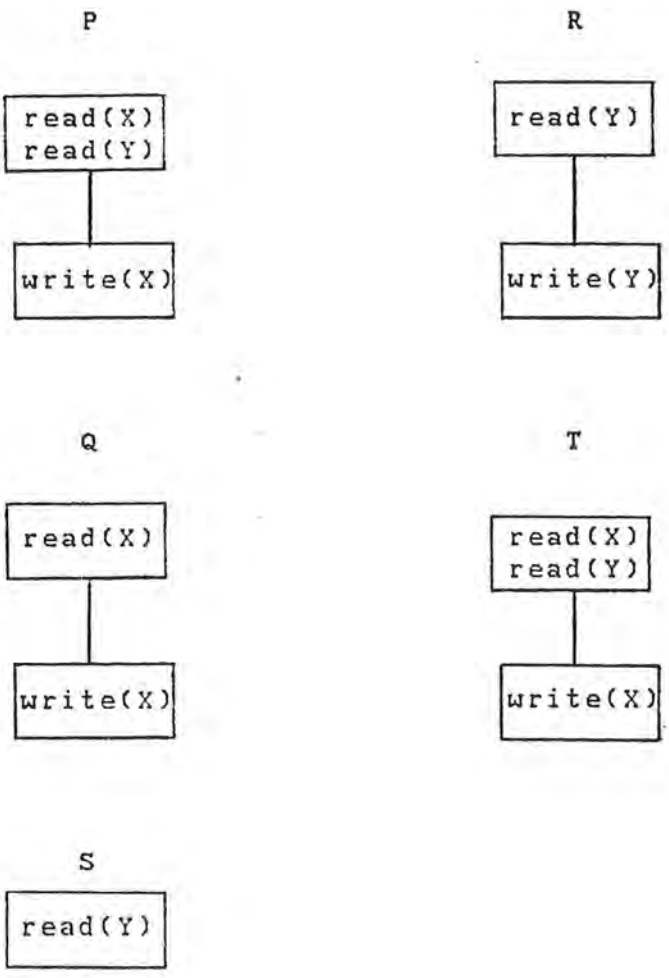


Fig. 4. Sites and transactions.

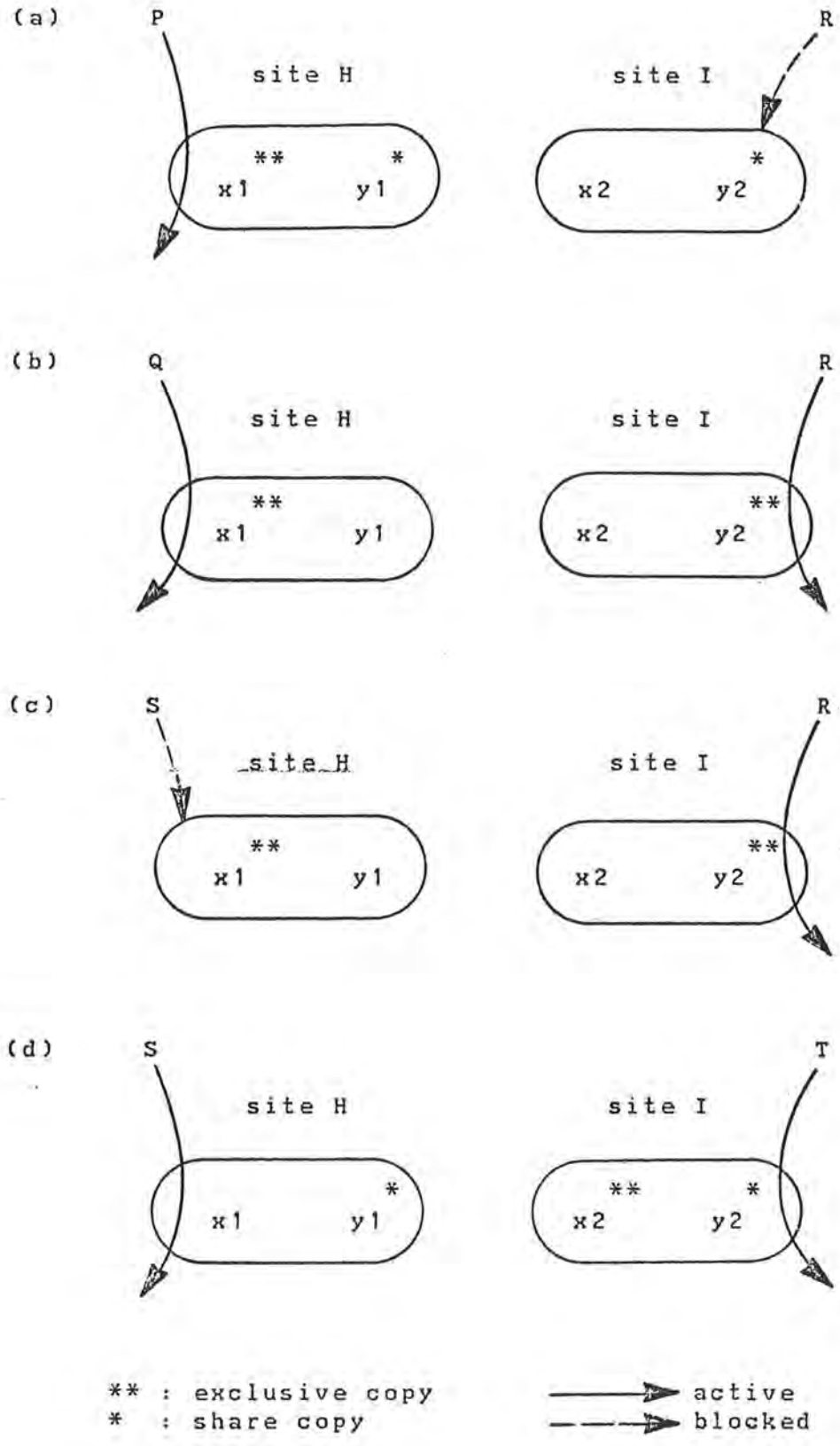
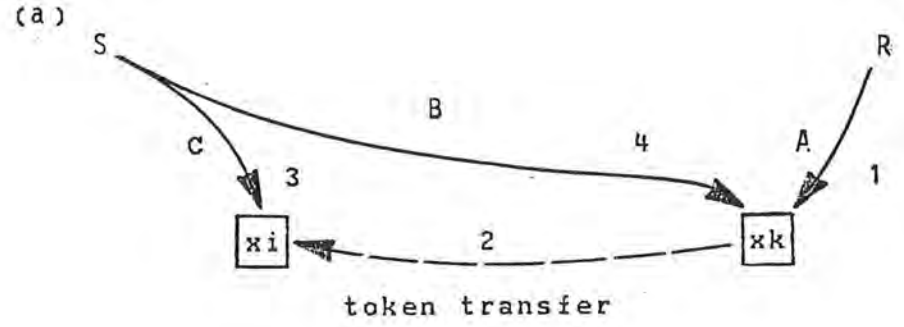
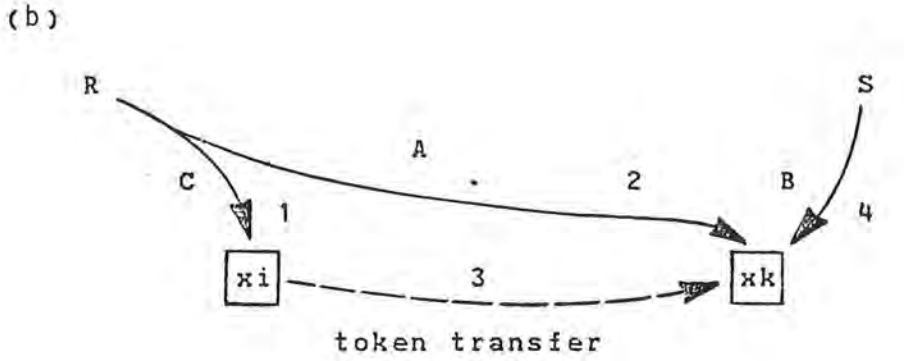


Fig. 5. Extended true-copy token algorithm.


(a)
S
R
B
C
4
A
3
1
xi
2
xk
token transfer


(b)
R
S
A
C
2
B
1
4
xi
3
xk
token transfer

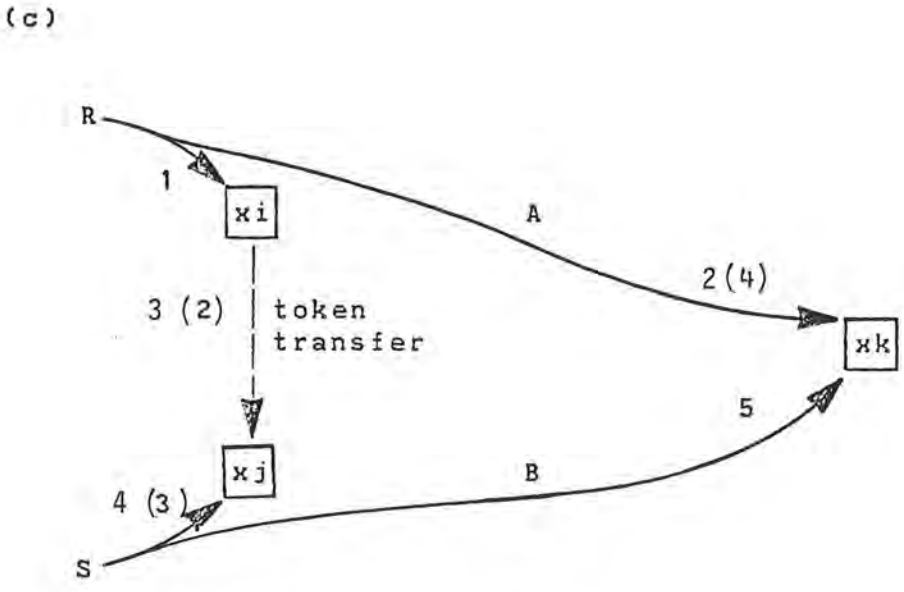

(c)
R
1
xi
A
2(4)
3 (2)
token
transfer
xk
5
xj
B
4 (3)
S

Fig. 6. ":<<<" and "<<p".

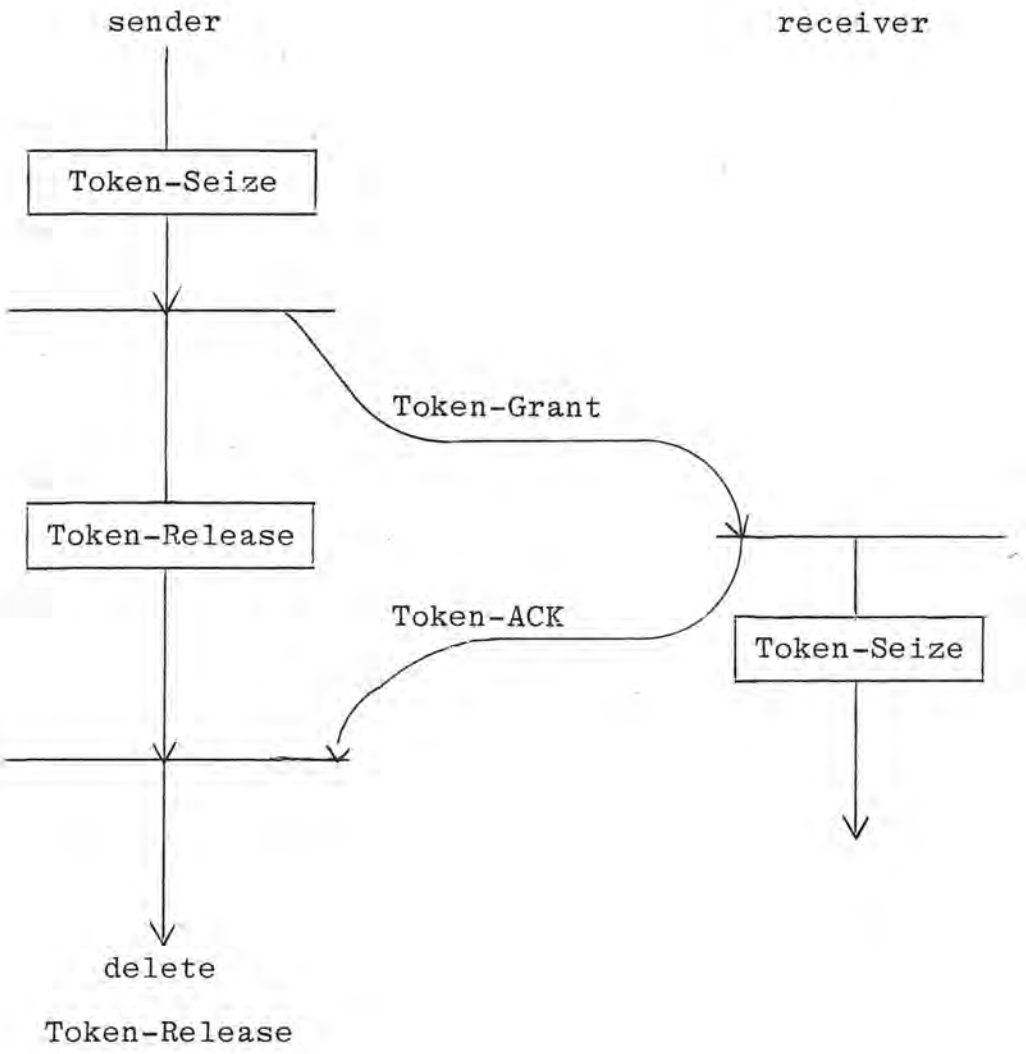


Fig. 7. Reliable token transfer.