Joint Conference of 1996 International Computer Symposium
December 19~21, Kaohsiung, Taiwan, R.O.C.

Constraint Maintenance for Replicated Databases

Keith Kuan-Shing Lee[1], San-Yih Hwang[2], Y. H. Chin[1], and Shu-Chin Su Chen[3]
[1] Department of Computer Science, National Tsing Hua University, Taiwan, R.O.C.
[2] Department of Information Management, National Sun Yat-Sen University, Taiwan, R.O.C.
[3] Institute of Computer and Information Engineering, National Sun Yat-Sen University, Taiwan, R.O.C.

Abstract

In this paper we investigate the issues of maintaining consistency in a distributed environment that allows data replication but does not require strict consistency between the replicated data items. Under such an environment, a different divergence constraint can be specified between the primary copy and a replicated copy of each data item. Both full and partial replications are considered. Moreover, we address different consistency requirements imposed by system: consistency maintained by either following explicit integrity constraints or executing transactions serializably. Different alternatives on maintaining the required consistency are proposed, and the tradeoffs between them are also discussed. Finally, we point out some consistency constraint maintenance pitfalls which are usually easy to be neglected by the system administrator under a partial replication environment.

1 Introduction

Data replication has long been used in a distributed system to improve availability and performance. A number of database vendors have released various replication server products that support data replication in a distributed environment [7]. In such an environment, the same data may be replicated at multiple sites for quick data access and for high data availability.

Currently, the internet provides access to a very large number of information sources. However, there still exist some challenges for global information systems [11]. The challenges are to provide easy, efficient, robust and secure access to information of various types. We believe that replication is an effective way to deal with these challenges. If we can successfully maintain consistency constraints under replication environment, we can provide easy and correct access to the needed information globally. Because of above reasons, it is important to maintain consistency constraints in a correct manner. For example, in the real world, Oracle's symmetric replication [5] uses eventual mutual consistency as a correctness criterion. However, for some applications, this correctness criterion may be too strict, resulting in low availability and performance. In this article, we propose various correctness criteria of constraint maintenance under different replication environments.

This paper explores constraint maintenance under two different replication environments, namely *full replication* and *partial replication*. In a full replication environment, secondary sites contain the same data objects that exist at the primary site. Note that all up-to-date data are stored at the primary site, and users can specify the extent of the data stored at the secondary sites to which it can diverge from that at the primary site. So, transactions issued to a site only access local data. On the other hand, in a partial replication environment, secondary sites can contain only part of the data objects stored at the primary site. So, transactions issued to a site may have to access data stored at another site.

In addition, this paper explores various problems in maintaining both *divergence constraints* and *integrity constraints* under the two replication environments. Divergence constraints, which describe how much a replica can diverge from its up-to-date value, are specified by the user and enforced by the system. Integrity constraints, for another, are used to maintain database consistency at a site. There are various types of divergence specifications for coherence conditions proposed in the literature [1, 13, 10]. These specifications specify the allowed difference between the primary copy and some secondary copies of the same object in terms of value, time lag, and others. For example, Alonso et. al. proposed four types of coherence conditions: *delay*, *version*, *arithmetics*, and *period* [1]. A delay condition indicates when an update operation should be applied to the specified secondary copies after the primary copy is updated. A version condition specifies the maximum difference between the secondary copies and the primary copy in terms of versions. An arithmetics condition enables a user to set a maximum difference for which the values of the specified secondary copies can diverge from the value of the primary copy. With respect to period conditions, a user can specify a period upon which the values of the primary copy and the secondary copies must be synchronized.

In our previous work [9], we have shown that the method of *Replication with Divergence* constraint has better performance than the traditional *Replication with Consistency* method under almost all circumstances. In this paper, we further study the effect of integrity constraint in the replicated database system employing replication with divergence. To our best knowledge, no papers have ever considered both divergence constraints and integrity constraints in a single framework. This is an interesting and important problem since the goal is to improve the system performance and to maintain the system consistency simultaneously in a replicated database.

We assume that there exist some integrity constraints that must be maintained at all sites in the system at all times, even in the presence of divergence on the values of replicated copies. For the purpose of discussion, we use arithmetic inequalities as integrity constraints in our examples. It has been shown that arithmetic inequalities are sufficient for representing integrity constraints in many cases [2]. Intuitively, the constraint enforcement in a system that contains both integrity constraints and divergence constraints can be briefly described as follows:

1. *Execute an update transaction at the primary site.*
2. *Then, check the associated constraints as follows:*
   - *(a) if any integrity constraint is violated then rollback,*
   - *(b) else check divergence constraints.*
   - *(c) if any divergence constraint is violated, then propagate the update values to the specified secondary sites.*

That is, after successfully executing an update transaction at the primary site, the primary site will satisfy all integrity constraints, while the secondary site conforms to the divergence constraints. However, even though the integrity constraints hold at the primary site, it may or may not hold at the secondary sites. We describe the problems by the following example. Note that we use $Max(x) = b$ to represent that the maximum value difference between the primary copy and the secondary copy of an object x is b.

Example 1: Consider two objects x, y and the integrity constraint $x + y \leq 10$. x' is a replica of object x and y' is a replica of object y. We assume that the divergence constraint on the values of x is $Max(x) = 3$ and that of y is $Max(y) = 1$. Initially, the values of these objects are as follows:

$$x = 6 \quad x' = 6$$
$$y = 4 \quad y' = 4$$

An update transaction decreases x by 2 at the primary site. Because of $Max(x) = 3$, the system does no need to propagate this effect to the secondary site. The resultant database state is as follows:

$$x = 4 \quad x' = 6$$
$$y = 4 \quad y' = 4$$

In this case, the integrity constraints hold at both sites: $x+y \leq 10$ and $x'+y' \leq 10$. Next, a second update transaction increases y by 2. Because of $Max(y) = 1$, this change made to the database state at the primary site must be immediately reflected at the secondary site. The resultant database state is as follows:

$$x = 4 \quad x' = 6$$
$$y = 6 \quad y' = 6$$

In this case, the integrity constraint holds at the primary site, but it does not hold at the secondary site. □

It is clear from Example 1 that, without any control, the consistency at a secondary site may be violated. This paper defines various correctness criteria for consistency under different environments and proposes algorithms for enforcing them.

The contributions of the paper are as follows.

1. We investigate constraints maintenance on a replicated database environment in detail.
2. Two constraints enforcement algorithms are given under a full replication environment.
3. When integrity constraints in the system are not given explicitly, we define a correctness criterion and propose an approach for enforcing the correctness.
4. We point out some consistency constraint maintenance pitfalls which are usually easy to be neglected by the system administrator under a partial replication environment.

The rest of the paper is organized as follows. In Section 2, an architecture of data replication environment is given. We describe in detail our constraint maintenance methods under a full replication environment in Section 3. In Section 4, we represent constraint maintenance under a partial replication environment. The paper ends with some conclusions in Section 5.

2 Architecture of data replication environment

We consider a data replication environment where a set of database servers are interconnected via a network to provide aggregate database services. We assume that each server is capable of storing and manipulating a fix amount of data. In addition, we assume that each server contains local constraint

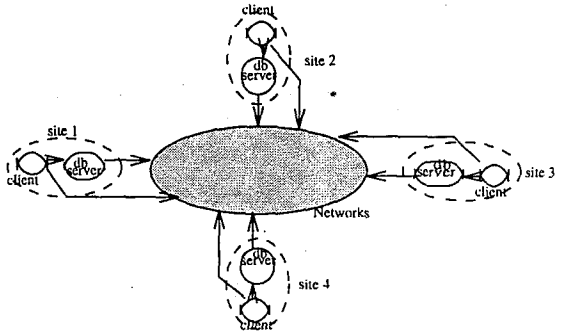

Fig. 1. Database Server Clustering Architecture

manager which is responsible for maintaining consistency constraint to ensure the correctness on the execution of transactions. However, a user on the local site can issue a query or an update request based on the entire data set to the local server. If the required data resides entirely on the local server, this request may be completely served locally. Otherwise it has to be shipped to other servers for processing. We call such a system *loosely coupled distributed database system* since each database server is able to operate independently. A loosely coupled distributed database system is typically a shared nothing environment which is an efficient scheme for providing high availability [3]. A shared nothing environment is such that each server owns its private disks and main memory, and all the servers are interconnected via a local- or wide-area network. Many commercial systems, such as Teradata's DBC/1012 [14] and Tandem's NonStop SQL [8] and research prototypes GAMMA [6] and BUBBA [4], make use of this environment. Its architecture is depicted in Figure 1.

We consider two types of replication under such an environment, namely full replication and partial replication. Each database server contains two part of data, namely local data that is only used at a local site and shared data that can be replicated to others sites. In a full replication environment, the shared data is completely replicated to the other database servers. However, in a partial replication environment, only part of share data is replicated to the other database servers.

There are two ways to maintain the consistency between the secondary copies and the primary copy of an object: synchronous and asynchronous. For synchronous approach, changes made to the database state at the primary site are immediately reflected at the secondary site. For asynchronous propagation, the changes made to the database state at the primary site are propagated to the secondary sites asynchronously. That is, the data at the primary site is firstly updated; the update of the data at secondary sites, if any, is conducted offline. In general, synchronous approach usually incurs more communication overhead than asynchronous approach, because it needs to send more synchronization information among different sites. That is, synchronous approach take more start up time to negotiate among sites before transmission. It has been shown that asynchronous approaches improves performance as well as system availability [12, 3, 9]. For this reason, we choose asynchronous propagation method, which can batch many requests before transmission, to further improve the system performance.

3 Constraint maintenance under a full replication environment

In this section, we study various policies for managing constraints on the replicated data. For each policy, we also discuss the trade-offs between constraint checking overhead at the secondary site and the amount of propagation data at the primary site in a full replication environment. We will describe three distinct approaches. First two explicitly maintain consistency constraints, while the last one implicitly maintains consistency constraints by the system.

For the purpose of the exposition, we assume a system consists of two sites, one is the primary site , and the other is the secondary site, with respect to some set of data. The primary site stores a set of up to date data, and the secondary site stores a set of replicated data. we use a lower case alphabet (e.g., x) to denote a data object at the primary site and the alphabet plus a prime (e.g., x') to represent the corresponding data object at the secondary site.

3.1 Maintaining consistency constraints explicitly

In this section, we consider an environment where the consistency constraints at a site are given explicitly. As described in Section 1, arithmetic inequalities are used for illustration.

We propose two approaches to maintaining explicitly consistency constraints: method A, which maintain consistency constraints at both sites, and method B, which maintain consistency constraints only at the primary site. Before we describe these two methods in detail, we define the correctness on the database state.

Definition [correctness of database state]. A *database state* refers to the values of all data objects in the system at a particular point in time. If a *database state* satisfies (1) divergence constraints between the primary site and the secondary site and (2) integrity constraints at the primary site, and (3) integrity constraints at the secondary site. We call the *database state*, after executing a series of transactions, to be *correct.*

Although the values of objects stored at the primary site and their replicas at the secondary site may be different at any given moment of time, they are considered correct (and satisfied by the users) if the above-mentioned three conditions hold.

3.1.1 Method A

For method A, integrity constraints are maintained at both sites. For example, the system maintains two identical consistency constraints $x + y \leq 10$ and $x' + y' \leq 10$, at the primary site and the secondary site, respectively. That is, constraint $x + y \leq 10$ is for checking the integrity constraint at the primary site and constraint $x' + y' \leq 10$ is for checking the integrity constraint at the secondary site. If the execution of an update transaction violates the integrity constraint at the primary site, then the system rolls back the current database state to its original consistent state. Further, if the execution of an update transaction violates the divergence constraint between the primary data and the secondary data , then the system need to propagate the effect to the secondary site. After the execution of the propagated update transaction, if the integrity constraint at the secondary site is violated, the system need to propagate more object data from the primary site to the secondary site. That is, at the secondary site, when some integrity constraints are violated, some missing updates must be further propagated to restore the consistency. The algorithm of constraint maintenance for method A is listed in Figure 2.

The enforcement of consistency constraints for method A can be accounted in three steps. Firstly, the database server at the primary site verifies the legibility of an update request.

```
Method A
{Let P stand for the primary site.}
{Let S stand for the secondary site.}
{Let IC stand for the set of integrity constraints.}
{Let DC stand for the set of divergence constraints.}
Begin
 While (1) do
  Begin
   Accept an update O_i at P;
   Execute O_i at P;
   If O_i violates IC after the update
   Then Rollback and Exit;
   If O_i violates DC after the update
   Then
    Begin
     Propagate the updated data to S;
     Repeat
      Let VIC' be IC violated after the
       propagation at S;
      Let X be the set of data that are in
       VIC' but yet to be propagated;
      If X ≠ ∅
      Then Propagate X from P to S;
     Until VIC' = ∅;
    End
  End
End
```

Fig. 2. Constraint Enforcement Algorithm for Method A

If, after update, some integrity constraints at the primary site are violated then this update must be rolled back and this algorithm stops. Secondly, the database server at the primary site checks the related divergence constraint. If any divergence constraint is violated then the updated data is propagated from the primary site to the secondary site (i.e., refresh the data at the secondary site). Thirdly, after executing the propagated update operations, the database server at the secondary site confirms the integrity constraints at the site. If any integrity constraint is violated then propagate the updated object's data involved in the violated integrity constraints from the primary site to the secondary site. If the propagated objects cause more integrity constraints being violated at the secondary site, then propagate more updated object's data from the primary site to the secondary site. This procedure continues until no integrity constraint is violated.

Example 2: Consider four objects w, x, y, and z, and three integrity constraints $z - y < 5$, $y - x < 5$, and $x - w < 5$. The integrity constraints are maintained at both sites. We assume that the divergence constraint of w, x, y, and z is $Max(w) = 5$, $Max(x) = 5$, $Max(y) = 5$, and $Max(z) = 3$ respectively. Initially, the database state is as follows:

$$w = 0 \quad w' = 0$$
$$x = 0 \quad x' = 0$$
$$y = 0 \quad y' = 0$$
$$z = 0 \quad z' = 0$$

Suppose a series of incoming update transactions increase w by 5, increase x by 5, increase y by 5, and increase z by 5. For the first three transactions, because $Max(w) = 5$, $Max(x) = 5$, and $Max(y) = 5$, the system does not need to propagate the effect to the secondary site. But, because $Max(z) = 3$, the system needs to propagate the value of object z to the secondary site. The resultant database state is as follows:

$$w = 5 \quad w' = 0$$
$$x = 5 \quad x' = 0$$
$$y = 5 \quad y' = 0$$
$$z = 5 \quad z' = 0$$

Since the integrity constraint $z - y < 5$ is violated at the

secondary site, the object y needs to be propagated from the primary site to the secondary site. After that, the integrity constraint $y - x < 5$ is then violated at the secondary site, and the object x needs to be propagated from the primary site to the secondary site, too. After the propagation of x, another integrity constraint $x - w < 5$ is in turn violated at the secondary site. Thus, the object w needs to be propagated from the primary site to the secondary site, too. The resultant database state is as follows:

$$w = 5 \quad w' = 5$$
$$x = 5 \quad x' = 5$$
$$y = 5 \quad y' = 5$$
$$z = 5 \quad z' = 5$$ □

In this example, the final database states at both sites are correct. However, at the secondary site, we must check integrity constraints and perform data propagation three times. If there are many secondary sites, the checking overhead and the propagation traffic may be heavy. Suppose there are M secondary sites and each secondary site maintains the same set of N integrity constraints, the total checking overhead has to be MN in the worst case. Moreover, the data propagation requests needed to restore the consistency of the database state at a secondary site may be a lot. In the next section, we will describe an approach that only maintain consistency constraints at the primary site, which reduces a large amount of checking overhead at secondary site, and lessen the propagation traffic. This approach is called method B.

3.1.2 Method B

For method B, integrity constraints are maintained at the primary site only. If an update transaction violates some integrity constraints at the primary site, then database state is rolled back to the original consistent state before the execution of the transaction. If the update transaction causes the divergence constraint of some replicas being violated, then the values of some objects need to be propagated to the secondary site. Note that the set of propagated objects contains all the objects that violated the divergence constraints, among others. The algorithm of constraint maintenance for method B is listed in Figure 3.

The enforcement of consistency constraints for method B can be accounted in three steps. Firstly, the algorithm executes an update operation at the primary site. If some integrity constraints are violated after the update, the update operation is rolled back, and this algorithm stops. Secondly, the algorithm checks the related divergence constraints. If any divergence constraint is violated, then the updated data must be propagated from the primary site to the secondary site. In addition to the newly updated data, some other data objects may have to be propagated together. In order to determine which objects need to be propagated, the algorithm checks the integrity constraints at the primary site by using transitive closure rules to find all the related objects which are related to the violated divergence constraints' object. Note that any updated objects that are yet to be propagated and related to the to be propagated object directly or indirectly via some integrity constraints have to be propagated. The reason why we need to propagate all the objects that are related to the violated divergence constraints' object at the same time is to avoid the potential violation of the integrity constraints at the secondary site. Note that we can use Depth_First_Search to replace Find_Transitive_Closure procedure. The time complexity of Depth_First_Search procedure is $O(e)$, where e is the number of connected edges in a graph.

Example 3: Consider four objects w, x, y, and z, and three integrity constraints $z - y < 5$, $y - x < 5$, and $x - w < 5$.

Method B

```
{Let P stand for the primary site.}
{Let S stand for the secondary site.}
{Let IC stand for the set of integrity constraints.}
{Let DC stand for the set of divergence constraints.}
Begin
 While (1) do
  Begin
   Accept an update O_i at P;
   Execute O_i at P;
   If O_i violates IC after the update
   Then Rollback and Exit;
   If O_i violates DC after the update
   Then.
    Begin
     Let T be the set of data involved in O_i;
     Find_Transitive_Closure(T);
     If T ≠ ∅
     Then Propagate the current value of the
      updated objects in T from P to S;
    End
  End
End

Find_Transitive_Closure(X)
Begin
 Let VIC' be IC that involve some data in X;
 Repeat
    VIC = VIC';
    X = the set of data involved in VIC;
    VIC' = IC that involve some data in X;
 Until VIC = VIC';
End
```

Fig. 3. Constraint Enforcement Algorithm for Method B

Integrity constraints are maintained at the primary site only. We assume that the divergence constraints on w, x, y, and z are $Max(w) = 5$, $Max(x) = 5$, $Max(y) = 5$, and $Max(z) = 3$ respectively. Initially, the database state is as follows:

$$w = 0 \quad w' = 0$$
$$x = 0 \quad x' = 0$$
$$y = 0 \quad y' = 0$$
$$z = 0 \quad z' = 0$$

Suppose a series of incoming update transactions increase w by 5 , increase x by 5, increase y by 5, and increase z by 5. For the first three transactions, because $Max(w) = 5$, $Max(x) = 5$, and $Max(y) = 5$, the system does not need to propagate the effect to the secondary site. But, because $Max(z) = 3$, the system needs to propagate the value of object z to the secondary site. Before propagating, Find_Transitive_Closure($\{z\}$) is invoked and return $\{x, y, z, w\}$. The reasons are that object z is involved with the integrity constraint $z - y < 5$ and object y is involved with the integrity constraint $y - x < 5$ and finally object x is involved with the integrity constraint $x - w < 5$. Thus, the database state of these four objects are propagated. So, the resultant database state is as follows:

$$w = 5 \quad w' = 5$$
$$x = 5 \quad x' = 5$$
$$y = 5 \quad y' = 5$$
$$z = 5 \quad z' = 5$$ □

In the above example only one round of data propagation is performed, in contrast to Example 2, where four rounds of data propagation are needed by using Method A. However, Method B may potentially introduce more (unnecessary) data to be propagated, as illustrated by Example 4.

Example 4: Consider four objects w, x, y, and z. We assume that the initial settings of the integrity constraints and

the divergence constraints are the same as the above example. So, we have the following initial database state:

$$\begin{array}{ll} w = 0 & w' = 0 \\ x = 0 & x' = 0 \\ y = 0 & y' = 0 \\ z = 0 & z' = 0 \end{array}$$

Suppose a series of update transactions increase w by 1, increase x by 2, increase y by 3, and increase z by 4 sequentially. Because $Max(w) = 5$, $Max(x) = 5$, and $Max(y) = 5$, the system does not need to propagate the effect to the secondary site for the first three transactions. However, because $Max(z) = 3$, the last transaction will cause the propagation of (at least) the value of object z to the secondary site. Moreover, due to the presence of the integrity constraints $z - y < 5$, $y - x < 5$, and $x - w < 5$, the object value of y, x, and w have to be propagated to the secondary site together with the object value of z. After propagation, the database state is as follows:

$$\begin{array}{ll} w = 1 & w' = 1 \\ x = 2 & x' = 2 \\ y = 3 & y' = 3 \\ z = 4 & z' = 4 \end{array}$$ □

In this example, the database state is correct. However, in this case, the object value of y, x, and w doesn't really need to be propagated, because the integrity constraints at the secondary site still hold even without the propagations. Only the object value of z needs to be propagated. Thus, method B may introduce extra propagations, as compared to method A.

It is the advantage of this method that there is no need to maintain integrity constraints at the secondary site. As a whole, there exist trade-offs between constraint checking overhead at the secondary site and the amount of propagation data for method A and B. We compare the two methods in the following section.

3.1.3 Comparisons of method A and B

In the following, we deal with the trade-offs between method A and method B. There are two factors that affect the performance. First, integrity and divergence constraint checking may incur local processor checking overhead. Second, information transfer will incur communication overhead. In a loosely coupled distributed database system as described in section 2, we assume that the communication cost is the dominant factor, compared to the processing cost.

We define the communication cost ($ComCost$) by the following equation,

$ComCost = T_{startup} + T_{unit} \times NumberofUnit$,

where $T_{startup}$ is the communication start up time before transmission. T_{unit} is the transmission time per communication unit.

Next, we describe the communication cost of method A and B. Let M and N be the volumes of propagated data using method A and B, respectively. N must be greater than or equal to M. Let K be the rounds of propagation caused by an update for method A. Let the communication cost of method A and B be $ComCost_A$ and $ComCost_B$, respectively.

$ComCost_A = (T_{startup} + T_{unit} \times \frac{M}{K}) \times K = K \times T_{startup} + T_{unit} \times M$

$ComCost_B = T_{startup} + T_{unit} \times N$

If K, the rounds of propagation caused by an update for method A, is significant and M and N, the volumes of propagated data using method A and B, are close, method B is better; otherwise, choose method A.

3.2 Maintaining consistency constraints implicitly

In this section, we consider a transaction processing environment where consistency is enforced by transactions. That is, integrity constraints are not given explicitly, but the execution of transactions ensures consistency. A transaction is a sequence of reads and writes against a database. The execution of transactions in conventional applications satisfies two properties: atomicity and serializability. The atomicity property means the sequence of reads and writes in a transaction is regarded as a single atomic action against the database. That is, a transaction either brings the database state into a new consistency state or does nothing. The serializability property means that the effect of concurrent execution of more than one transaction is the same as that of executing the same set of transactions one at a time. At the primary site, the execution of transactions satisfies these two properties. However, at the secondary site, merely satisfying these two properties may not be desirable. Since the secondary site stores replicas of some data at the primary site, there must exist some desirable relationship between the data stored at both sites. Later we will discuss the relationship between the data stored at both sites.

3.2.1 Notation

Let T_i, $1 \leq i \leq m+1$, be an update transaction that updates to one or more of objects $x_1, \ldots, x_n$. We assume the database server at the primary site ensures serializability in its execution of transactions. Let $T_1, T_2, \ldots, T_m$ be the transactions executed at the primary site but yet to be propagated to the secondary site. Without loss of generality, let the serialization order of these transactions be $T_1 \rightarrow T_2 \rightarrow \ldots \rightarrow T_m$. Let T_{m+1} be the next update transaction at the primary site that violates divergence constraints. Then some transactions in $\{T_1, T_2, \ldots, T_{m+1}\}$ need to be propagated to the secondary site to keep the database state consistent at the secondary site consistent.

Obviously, we need to propagate to the secondary site some execution effect that is already applied to the primary site. Exactly what to be propagated is an issue that need to be considered. The following example illustrates the effects of propagating different types of information.

Example 5: We assume that the initial database state S_0 is $\{x = 0, y = 0, z = 0, w = 0\}$. There are a series of update transactions: T_1:update(x,y), T_2:update(w), T_3:update(z,w), and T_4:update(y,z). Suppose each update operation increments every mentioned data object by one. For example, update(x,y) will increase both x and y by one. The execution order is T_1, T_2, T_3, and T_4 at the primary site. We assume that both T_2 and T_3 satisfy divergence constraints, but T_1 and T_4 do not. The divergence constraints are as follows: $Max(x) = 0, Max(y) = 3, Max(z) = 1$, and $Max(w) = 3$. After executing T_1, T_2, T_3, and T_4, the database state S_4 at the primary site is $\{x = 1, y = 2, z = 2, w = 2\}$. There are two alternatives to update the database state at the secondary site:

1. Pass the operations of updated transactions:
   a.) Executing the update transactions T_1 and T_4 at the secondary site in order, then the system brings the database state to $\{x = 1, y = 2, z = 1, w = 0\}$.
   b.) Executing the update transactions T_1, T_2, T_3, and T_4 at the secondary site in order, then the system brings the database state to $\{x = 1, y = 2, z = 2, w = 2\}$.
2. Pass the data that is updated by the transactions violating some divergence constraints.
   The database state at the secondary site becomes $\{x = 1, y = 2, z = 2, w = 0\}$.

We don't consider method 1.a to be correct. After executing T_1, T_2, T_3, and T_4 the correct change of database states at the primary site are $\{x=1, y=1, z=0, w=0\}$, $\{x=1, y=1, z=0, w=1\}$, $\{x=1, y=1, z=1, w=2\}$, and $\{x=1, y=2, z=2, w=2\}$. By applying method 1.a, the resultant database state at the secondary site is $\{x=1, y=2, z=1, w=0\}$, which is not any of the above database states. Although T_2 and T_3 may be propagated to the secondary site later, the final state will be different. Since the execution of $\{T_1, T_2, T_3, T_4\}$ and $\{T_1, T_4, T_2, T_3\}$ may yield different results. Similarly, method 2 results in a database state that is not any of the previous states in which the primary site has ever been. So, only method 1.b brings the database state to a consistent state of both the primary and the secondary site. □

3.2.2 Definition of correctness

If the database is in a consistent state before executing the transaction, it will be in a consistent state after the complete execution of the transaction, assuming that no interference with other transactions occurs.

$S_0 \xrightarrow{T_1} S_1 \xrightarrow{T_2} S_2 \xrightarrow{T_3} \ldots \xrightarrow{T_m} S_m$

Let S_0 denote the initial database state before executing any transactions. We assume that transactions are executed in the order of $T_1, T_2, \ldots, T_m$. The execution of $T_i, 1 \leq i \leq m$, brings the database state from S_{i-1} to S_i. At any time, the database state S_k at the primary site and the database state S_l at the secondary site maintain the relationship $k \leq l$. That is, the database state at the secondary site is the previous or current database state at the primary site.

Definition [correctness of transaction execution]. Let P stands for the primary site and S stands for the secondary site. Let T stands for any time point and δ stands for any period of time. The database state at the secondary site can be previous database state at the primary site at some time. However, the final database state at the secondary must be the same as the database state at the primary site at certain time. So, an execution of transactions at the primary and secondary site is correct, if

$\exists\, \delta \geq 0$ $s.t. DatabaseState(P, T-\delta) = DatabaseState(S, T)$

We use an example to explain the correctness of transaction execution.

Example 6: Consider the same initial database state, transactions, and divergence constraints as in Example 5. After executing T_1, T_2, T_3, and T_4, the database state changed at the primary site are:
$S_1 = \{x=1, y=1, z=0, w=0\}$,
$S_2 = \{x=1, y=1, z=0, w=1\}$,
$S_3 = \{x=1, y=1, z=1, w=2\}$, and
$S_4 = \{x=1, y=2, z=2, w=2\}$.

After executing the update transaction T_1 at the primary site, the database state is S_1 at the primary site and the secondary site. After executing the update transaction T_2 at the primary site, the database state is S_2 at the primary site and S_1 at the secondary site. After executing the update transaction T_3 at the primary site, the database state is S_3 at the primary site and S_1 at the secondary site. After executing the update transaction T_4 at the primary site, the database state is S_4 at both the primary site and the secondary site. □

3.2.3 Issues and problems

As described in Section 2, we use asynchronous propagation method to improve the system performance on maintaining the secondary copy. In this section, we will discuss what to propagate and when to propagate. These two issues must be carefully considered to ensure the correctness and efficiency of the system.

What to propagate

We consider two kinds of environments. One allows ad-hoc transactions and the other allows only predefined transactions. In an ad-hoc transaction processing environment, we can group the effect of all unpropagated transactions into one big transaction and send it to the secondary site for execution. Alternatively, we can group the effect into a sequence of smaller transactions, and send them to the secondary site for execution.

Gluing a set of transactions to one big transaction will introduce some problems. Firstly, a long-duration transaction will hold more resources during execution of this transaction. Secondly, the possibility for deadlocks is dramatically increased, which will bring down the system performance and increase the possibility for abortion and restart. However, there are advantages for this approach. It may eliminate duplicate operations appeared so far. For example, suppose an object is updated several times before propagation, all the updates can be reduced into a single update operation. This will reduce execution time at the secondary site after propagation. Moreover, the big transaction will reduce the total communication time.

In a predefined transaction processing environment, only a finite types of transactions are allowed. An example is the banking system, where only *Withdraw*, *Deposit*, *Transfer* and *Balance* are provided. Under such an environment, we can use commutativity to switch the order of unpropagated transactions and group a set of adjacent transactions in to a single transaction. Two transactions T_1 and T_2 are said to be commutative if and only if, for any system state S, the following two conditions hold: 1) the states that result from the execution sequences $T_1;T_2$ and $T_2;T_1$ applied to S are not distinguishable, and 2) both T_1 and T_2 have the same return values in both execution sequences. For example, two *Deposit* transactions on a bank account are commutative and can therefore be admitted concurrently. We can use this activity to reduce the amount of propagation data.

Example 7: A bank provides the predefined transactional operations as described above. Initial execution at the primary site are

1. Deposit(X, 100): one deposit \$100 to an account X.
2. Withdraw(Y, 200): one withdraw \$200 from an account Y.
3. Deposit(X, 300): one deposit \$300 to an account X.
4. Withdraw(Y,100): one withdraw \$100 from an account Y.

We can apply the commutative law to switch the execution order of operations and then merge some operations to create a shorter list of transactions to be submitted to the secondary site.
Commutativity phase:

$Deposit(X, 100)$
$Deposit(X, 300)$
$Withdraw(Y, 200)$
$Withdraw(Y, 100)$

Merging phase:

$Deposit(X, 400)$
$Withdraw(Y, 300)$

As a result, only two transactions are propagated to the secondary site. □

When to propagate

For some types of divergence constraints, e.g. version and arithmetics, the propagation of updated effect occurs only when an update arrives and this update violates some divergence constraints. For other types of divergence constraints, e.g. delay and period, the propagation of update effect is triggered by a timer. Moreover, the updated effect must be applied to the secondary site before the deadline. A formula can be derived to determine when to propagate a set of transactions. We assume that communication lines are of no failure. That is, the transmission time is countable. Let the current unpropagated transactions be T_1, T_2, ..., T_i, and X be a set of data items that are updated by these transactions and have some timing constraints. Let x denote the data item in X that has the earliest deadline. Suppose t_{D_x} is the deadline for x, and the current time is $t_{current}$. Transaction T_1, T_2, ..., T_i are propagated if the following formula holds,

$t_{D_x} - t_{current} - (t_{group} + (t_{startup} + t_{unit} \times NumberofUnit) + t_{execution}) \leq \varepsilon$,

where t_{group} denotes the time to group a series of transactions to be a sequence of transactions and eliminate some duplicate operations. As described in previous subsection, the subformula $(t_{startup} + t_{unit} \times NumberofUnit)$ accounts for the total communication time for transmitting these transactions. $t_{execution}$ stands for the transaction execution time at the secondary site. ε is a threshold. ε should be set big enough to account for the variation of the calculation but not too big to cause extra propagation.

In summary, the processing steps of transaction propagation are the following:

1. Determine the propagation time point, according to the above formula.
2. Group a series of transactions into a sequence of transactions, and eliminate some duplicate operations.
3. Assemble these transactions into some packages for transmission.
4. Transmit these packages through network.
5. Disassemble these transmitted packages into a sequence of transactions.
6. Execute these transactions.

Step 1 to step 3 are processed at the primary site. Step 4 is a communication step. Step 5 and step 6 are processed at the secondary site.

4 Constraint maintenance under a partial replication environment

The main difference between full and partial replication environment is that at the full replication environment transactions only access local data, but at the partial replication environment transactions may access data that are not local. Thus, the constraint maintenance under a partial replication environment is more difficult than that under a full replication environment. That is, we must maintain not only local consistency constraints but also the constraints that involve remote site's data. For simplicity, we only deal with the explicit integrity constraints rather than the implicit integrity constraints at a partial replication environment. Besides, we will describe an approach on how to maintain constraints under such an environment.

There are some observations under a partial replication environment. When we load more secondary copies to a site, not only more divergence constraints but also additional integrity constraints are introduced. We illustrate the pitfalls for maintaining consistency constraints in the following example. At the same time, we use this example to explain how the constraint enforcement algorithm work step by step. The algorithm of constraint maintenance for Partial_Replication is listed in Figure 4.

Example 8: At the primary site, namely site 1, we have three object x, y, and z. Assume that the initial database state is $\{x = 5, y = 5, z = 5\}$. Without loss of generality, we assume that site 2 contains x', site 3 contains y', and site 4 contains z'. Initially, $x' = x$, $y' = y$, and $z' = z$. Suppose there is an integrity constraint among x,y, and z such as $\{x + y + z < 16\}$ which is expressed as $IC(x, y, z)$. The transaction is T_1: $\{Write(x, 5, 4), Write(y, 5, 4), Write(z, 5, 7)\}$ at site 1. Note that, $Write(x, old_x, new_x)$ means write object x and replace old value of x with new value of x.

User must maintain an integrity constraint using "if $IC(x, y, z)$ then commit else abort". We assume that after executing write operation the system does not need to propagate the effect to the secondary site. That is, the divergence constraints are satisfied. So, after $Write(x, 5, 4)$ we must ensure that $IC(x, y, z)$ and $IC(x', y, z)$ are maintained. That is, the database server at site 1 maintains the additional integrity constraint set $IC' = \{x' + y + z < 16\}$. Note that B involved with O_i is object x and E is objects $\{y, z\}$. $IC(B', E) = IC(x', y, z)$ is not in $IC' = \emptyset$. This processing step is marked as line number 1 in Figure 4.

Similarly, after $Write(y, 5, 4)$ we must ensure that $IC(x, y, z)$ and $IC' = \{x' + y + z < 16, x + y' + z < 16\}$ are maintained. Finally, after $Write(z, 5, 7)$, IC' is $\{x' + y + z < 16, x + y + z' < 16\}$. This processing step is marked as line number 2 in Figure 4. Additionally, if the update operation violates the divergence constraints, then the database server will propagate the current value of the updated object from site 1 to the secondary site. This processing step is marked as line number 3 in Figure 4. □

In general, after executing retrieval operation at the secondary sites the system does not need to maintain integrity constraint explicitly. So, if we do not maintain the above constraints at site 1, then some errors will occur at site 2, site 3, or site 4 when user execute the retrieval operation at the corresponding site. For example, at site 4 suppose a user places $Read(x, y, z)$ operation. While z can be found in the local site, x and y must be transmitted from site 1. Thus, the values read by this operations will be (x, y, z'). This is why $IC(x, y, z')$ needs to be maintained at site 1.

The goal of this enforcement algorithm is to automaticly detect what set of integrity constraints we need to maintain. In general, there are two propagation approaches. One is only to propagate the violated object's value and another is to propagate the values of all updated objects which belong to the identical integrity constraint. Both approaches will bring about the changement of the integrity constraints set. We choose the first approach in this algorithm for simplicity.

5 Conclusions

In this paper, we first discussed a system architecture for the data replication environment. Then we discussed the issues of constraint maintenance under the full and partial replication environment.

In a full replication environment, we have studied three distinct approaches for managing constraints in this form of replication. First two explicitly maintain consistency constraints. One maintains consistency constraints at both primary and secondary sites, and the other maintains consistency constraints only at the primary site. The trade-offs between constraint checking overhead at the secondary site and the amount of propagation data at the primary site for these two methods

```
Partial_Replication()
{Let P stand for the primary site and S stand for the secondary site.}
{Let B stand for the current updated object.}
{Let B' stand for the corresponding data object at the secondary site.}
{Let E stand for the objects which belong to the identical integrity constraint associated with object B, but don't contain object B.}
{Let IC stand for the set of integrity constraints.}
{Let IC' stand for the additional integrity constraint set which is dynamicly managed by this algorithm.}
{Let DC stand for the set of divergence constraints.}
Begin
  IC' = ∅;
  While (1) do Begin
    Accept an update O_i involving an
     object B at P;
    Execute O_i at P;
    If O_i violates IC after the update
    Then Rollback and Exit;
    1: If IC(B', E) is not in IC'
    Then IC' = IC' + IC(B', E);
    2: If O_i violates some integrity constraints in IC'
    Then Begin
       Propagate the current value of the relative
        violated object(s) in the same violated
        integrity constraint(s) from P to S;
       IC' = IC' - the violated integrity constraint(s);
      End
    3: If O_i violates DC after the update
    Then Begin
       Propagate the current value of the
        updated object B from P to S;
       IC' = IC' - IC(B', E);
      End
   End
End
```

Fig. 4. Constraint Enforcement Algorithm under a Partial Replication Environment

have been discussed. The third one is applied to an environment where consistency is maintained by the (serializable) execution of transactions. We have defined the correctness of transaction execution at the secondary site. In addition, we have studied various issues and problems of transaction execution about what to propagate and when to propagate. We have considered two kinds of environments. One allows ad-hoc transactions and the other allows only predefined transactions.

In a partial replication environment, we have studied some consistency constraints maintenance pitfalls which are usually easy to be neglected. We have also proposed an algorithm for constraint enforcement in this environment.

References

[1] R. Alonso, D. Barbar, and H. Garcia-Molina. Data caching issues in an information retrieval system. *ACM Transaction on Database Systems*, 15(3):359–384, 1990.

[2] D. Barbara-Milla and H. Garcia-Molina. The demarcation protocol: A technique for maintaining constraints in distributed database systems. *The International Journal on Very Large Data Bases*, 3(3):325–355, 1994.

[3] A. Bhide, A. Goyal, H.-I Hsiao, and Anant Jhingran. An efficient scheme for providing high availability. In *Proc. of ACM SIGMOD Int'l. Conf. on Management of Data*, pages 236–245, 1992.

[4] G. Copeland, W. Alexander, E. Boughter, and T. Keller. Data placement in bubba. In *Proc. of ACM SIGMOD Int'l. Conf. on Management of Data*, 1988.

[5] D. Daniels, L. B. Doo, A. Downing, C. Elsbernd, G. Hallmark, S. Jain, B. Jenkins, P. Lim, G. Smith, B. Souder, and J. Stamos. Oracle's symmetric replication technology and implications for application design. In *Proc. of ACM SIGMOD Int'l. Conf. on Management of Data*, 1994.

[6] D. DeWitt, S. Ghandeharizadeh, D. Bricker, H. Hsiao, and R. Rasmussen. The gamma database machine project. In *Proc. of ACM SIGMOD Int'l. Conf. on Management of Data*, 1988.

[7] A. Gorelik, Y. Wang, and M. Deppe. Sybase replication server. In *Proc. of ACM SIGMOD Int'l. Conf. on Management of Data*, 1994.

[8] Tandem Database Group. Nonstop sql, a distributed, high-performance, high-reliability implementation of sql. In *Workshop on High Performance Transaction Systems*, 1987.

[9] S. Y. Hwang, Keith K. S. Lee, and Y. H. Chin. Data replication in a distributed system: A performance study. In *Proc. of the 7th Int'l Conf. and Workshop on Database and Expert-System Applications*, 1996.

[10] R. Lenz, T. Kirsche, and B. Reinwald. Aspect – specifying consistency requirements for replicaed data. In *Proc. of the 7th Int'l Conf. on Parallel and Distributed Computing Systems*, 1994.

[11] A. Y. Levy, A. Silberschatz, D. Srivastava, and M. Zemankova. Challenges for global information systems. In *Proc. of the 20th Int'l Conf. on Very Large Data Bases*, 1989.

[12] Christos A. Polyzois and Hector Garcia-Molina. Evaluation of remote backup algorithms for transaction processing systems. *ACM Transaction on Database Systems*, 19(3):423–449, 1994.

[13] M. Rusinkiewicz, A. Sheth, and G. Karabatis. Specifying interdatabase dependencies in a multidatabase environment. *IEEE Computer*, 24(12):46–51, Dec. 1991.

[14] Teradata. *DBC/1012 Database Computer System Manual Release 2.0*. Teradata Corp., 1985. *Document No. C10-0001-02.*