

A Hierarchical Export/Import Scheme for Data Sharing in a Federated Distributed Database System

Hyunchul Kang
Department of Computer Science
Chung-Ang University
Seoul, 156-756
Korea

Sang Hyuk Son
Department of Computer Science
University of Virginia
Charlottesville, VA 22903
U.S.A.

Abstract

When a data object is shared in a federated distributed database system (FDDBS), the exporter is responsible to refresh data of all importers. This causes the bottleneck phenomenon at the exporter's site due to all importers competing for its refresh service. In this paper, we propose an efficient export/import scheme for data sharing in a FDDBS that can solve this problem. We devise the procedures for the exporter to update its copy, and for the importers to access their copies, and the negotiation protocol for data exporting/importing between two databases in the federation. We introduce the concept of the cost of data exporting, and derive equations to quantify it. Knowing this cost helps each database decide in negotiation whether it can export data further or not. Experimental results show that using our scheme, the bottleneck phenomenon at the exporter's site can be eliminated at the penalty of incurring more data transmission in the federation than using a simple export/import scheme that suffers from the bottleneck problem, but they also show that this penalty is not severe.

1. Introduction

The conventional approach to integrating multiple databases each of which was independently designed and managed is to integrate their database schemata towards a global schema and to define mappings between the global schema and each of the local schemata. The user queries and/or transactions are specified against the global schema and translated to be executed against the relevant local databases. This approach provides users with physically distributed but logically centralized view of the integrated database system. Users lose the control over their local databases. They should conform to the global view of the database.

An alternative approach to database integration employs a federated architecture [HM85]. A federated distributed database system (FDDBS) provides users with not only physically but

also logically distributed view of the integrated database system. Users do not lose the control over their local databases. That is, the autonomy of each local database system is preserved while data sharing with other database systems is achieved. In a FDDBS, there is no global schema. Instead, the export/import mechanism is used for data sharing. Each database participating in the federation defines data that it is willing to share with other databases in its export schema, and defines data that it can access but is owned by other database in its import schema. Before a data object is shared between two databases, they talk to each other to decide which data they are going to export/import and in what capacity. This procedure is called data access negotiation.

The federated approach to database integration for data sharing seems very practical one. It can avoid the difficulty of schema integration which is often not completely doable, eliminate or reduce high complexity of distributed data management of the conventional distributed database system (CDDBS) possibly involving heterogeneity among local database systems, and allow users to keep the control over their local databases. Recently, techniques to solve various problems encountered in the FDDBS attract many researchers [AB89][W89]. However, the FDDBS technology is in its infancy, problems have not been deeply investigated, and not even yet been addressed or identified.

In this paper, we propose an efficient export/import scheme for data sharing in a FDDBS. Our work is motivated to deal with the problem described below. Suppose a database in the FDDBS (say, DB1) stores sharable data D, and another database (say, DB2) has imported it after negotiation with DB1. We call DB1 the owner or exporter of D and DB2 the borrower or importer of D. When a data object is shared among multiple databases in the federation, we assume the following: Only the owner can update the data and all borrowers can access the data in read only mode. Thus, the owner is responsible to propagate its updates to all its borrowers. (The owner takes this responsibility during the data access negotiation with importers.) However, the propagation is deferred up to the point where a borrower wants to refresh its copy and requests the owner to send the current copy. These assumptions are reasonable in the FDDBS because the autonomy of every local database system should be preserved.

Now we introduce the concept of the cost of data exporting. The exporter is supposed to pay the cost of refreshing data copies of the importers on their requests. This cost involves the processing time and data transmission over the communication network. Since borrowers compete for the refresh service of the owner, if an owner have exported data to a number of borrowers, the owner's database site becomes the bottleneck. Thus, for a

sharable data object D , the number of borrowers that can import D and get the refresh service from the owner is limited to a certain number determined by the system characteristics, the processing capability of the owner, the frequency of updating D by the owner, the frequencies of accessing D by borrowers and so on. This number might be too small to satisfy the sharing requirements and needs of the users in the FDDBS.

The export/import scheme that we propose relieves the bottleneck phenomenon at the owner's database system and thus enhance the degree of data sharing. To share a data object D in the federation by our export/import scheme, we need procedures for the owner to update D , for the importers to access D , and a protocol for two databases to negotiate the access to D . We devised them. In doing so, the cost of exporting is quantified to be used in the data access negotiation. We evaluated our scheme and showed its superiority.

The rest of this paper is organized as follows. Section 2 briefly describes our scheme. Related works are surveyed and discussed in section 3. Section 4 presents the update and the access procedures. Section 5 derives the equations to quantify the cost of exporting and presents the negotiation protocol. Section 6 reports the evaluation of our scheme. Concluding remarks and further research directions are in section 7.

2. Our Export/Import Scheme

The main idea of our scheme is briefly described in this section. More through treatment is given in section 4 and 5. When a data object is shared among multiple databases in the federation, the export/import relationship among them can be represented by an n -ary tree structure. The root node of the tree represents the owner of the data, and all non-root nodes represent its borrowers. For example, Figure 1(a) represents the export/import relationship where the owner (the root node numbered 1) has exported the data to 6 borrowers (children nodes of the root numbered 2 through 7). In this flat n -ary tree of height 2, the exporting cost of the root (i.e. the owner) increases as the number of the children (i.e. the borrowers) increases leading to the bottleneck phenomenon at the root.

In our scheme, called a hierarchical export/import scheme, to lower the exporting cost of the owner, a borrower is allowed to indirectly import the data from the owner as depicted in an export/import relationship tree of Figure 1(b). In it, the root (node 1 which is the owner) has directly imported data (say, D) only to node 2 and 3, and let them again export D to further nodes (2 to 4; and 3 to 5,6,7). Thus, node 2 is not only an importer but also an exporter. So is node 3. In general, the non-leaf nodes are exporters, the non-root nodes are importers, and thus non-root/non-leaf nodes are both importers and exporters at the same time. As far as the update propagation is concerned, node 1 is supposed to do that only to node 2 and 3. Node 4 is supposed to get the refresh service from node 2, its parent node, and node 5, 6, and 7 are supposed to get it from node 3, their parent node. Suppose, for example, node 4 wants to refresh its data but its parent, node 2, cannot do that because its copy is also outdated. Then, node 2 has to refresh its copy first by requesting its parent, node 1, to do that, causing a series of data refreshing. In summary, a node except the root gets data refresh service directly from its parent node, and a node except the leaves is responsible to refresh its direct childrens' data only. The height of

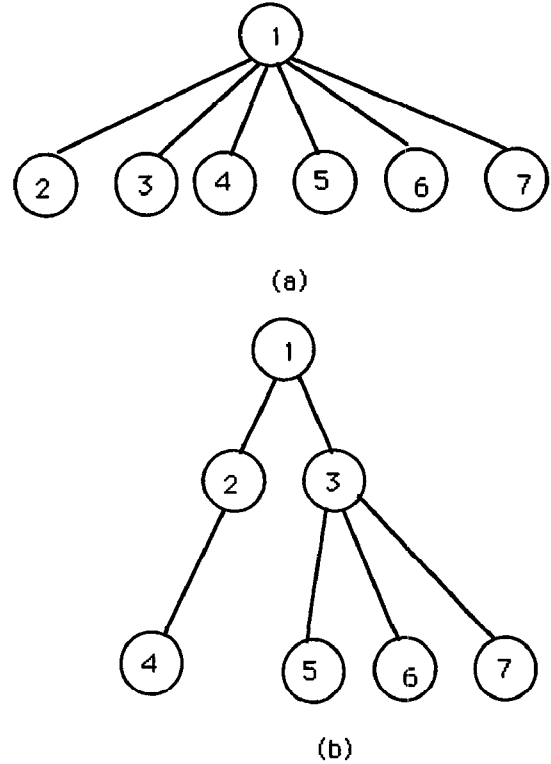


Figure 1 :
Export/Import Relationship Trees

the export/import relationship tree of our scheme can be greater than 2. The tree structure employed in our scheme is just a logical one, thus, our scheme does not require the network topology to be a tree.

In our scheme, when a data object is shared, the owner's update of the data need not be synchronized with the borrowers' accesses. Each shared data copy is assigned with a version number. When a borrower wants to access the current version of the data, it has first to check with the owner to see if its copy is current. If not, it requests its parent to refresh its copy. This inquiry to the owner is the only case where the borrowers which are not the direct children of the owner have direct communication with the owner.

3. Related Works

Few works have been conducted in the area of the FDDBS. No directly related work dealing with the same problem of this paper has been reported in the literature. In this section, we survey and discuss some relevant works conducted in the past in the areas of the FDDBS and the CDDBS.

A scheme for update propagation in the FDDBS using the quasy-copy was proposed in [AB89]. A quasy-copy is a borrower's copy whose value is allowed to deviate from that of the owner's copy in a controlled fashion. In this scheme, all the borrowers' copies are implemented as quasy-copies. Assuming that data consistency requirement can be relaxed, the updates done to the owner's copy are propagated only to those borrowers' copies whose values get deviated from that of the owner's copy beyond the predefined tolerable amount. The aim of this scheme

is to reduce the owner's workload incurred to maintain the consistency of the borrowers' copies.

In the CDDBS when data is replicated, a scheme to synchronize its updates and accesses to preserve data consistency is to direct all updates to a single copy which is designated as the primary copy, and to bring all other copies, called secondary copies, up-to-date by propagating the updates of the primary copy. The schemes for such update propagation in the CDDBS were investigated. The propagation is either immediate or deferred. A deferred approach using the database snapshot [AL80] is discussed in [D83]. A database snapshot stores the as-of data which may not be up-to-date. When the inconsistency of the snapshots is not tolerable, they are refreshed to the current state. The secondary copies can be managed as snapshots in applications where the absolute data consistency is not required. Recently, Lindsay et al. [L86] proposed a differential snapshot refresh scheme where only the updated portion of the primary copy, not the entire, is transmitted to the site of the snapshot.

More complicated deferred propagation schemes, the anti-entropy scheme and the rumor mongering scheme, were investigated in [DG87]. In the anti-entropy scheme, every site regularly chooses another site at random and resolves any differences between their copies. In the rumor mongering scheme, after a site receives a new update, it periodically chooses another site at random and ensures it has seen the update. When a site has tried to propagate the update to too many sites that have already seen the update, it stops propagating the update. These two schemes are called epidemic due to their behavior.

Exporting/importing data in a FDDBS results in data replication. Data replication requires access synchronization to preserve data consistency. Various schemes were proposed for replica control in the CDDBS. They include the read-one/write-all scheme of SDD-1 [BSR80], the primary copy scheme of distributed INGRES [S79], the majority consensus approach [T79], and the quorum-based approach [G79]. However, the conditions and requirements of the replica control in the CDDBS are different from those in the FDDBS. In the CDDBS, the primary purpose of data replication is to enhance reliability and availability of the system in spite of failures and to store data near the point of use for rapid access. On the other hand in the FDDBS, the reason of data replication is data sharing. (Note that in the CDDBS data sharing among multiple databases is not explicitly recognized. Users can access any data stored at any local database unless the security subsystem prohibits it since they are accessing a logically centralized database.) As far as data update is concerned, in the CDDBS any authorized user is able to update the replicated data but in the FDDBS it is reasonable to assume that only the owner of the data can update its copy and all the importers can access their copies in read only mode. There are two reasons for this. First, the importers are regarded as borrowers of the data. Second, it is not desirable to allow the owner to access borrowers' databases to update their copies because the site autonomy is extremely important in the FDDBS. Thus, if an importer wants to access the current version of the imported data, it has first to request the owner to send the current copy. The owner is then responsible to send its up-to-date copy to the requesting importer. The owner takes this responsibility during data access negotiation with importers.

The update propagation from the owner to the importers in

the FDDBS resembles that performed in the primary copy scheme for replica control in the CDDBS discussed above. However, the semantics of update propagation in the CDDBS and those in the FDDBS are different from each other. In the CDDBS, to fully take advantage of data replication, the primary copy site is usually obligated to propagate the updates as early and completely as possible to all secondary copies. On the other hand, in the FDDBS since the site autonomy is highly desired, the owner of the replicated (shared more precisely speaking) data should be freely able to update its copy and the update propagation is initiated only on the importer's request.

The above arguments imply that although the replica control techniques in the CDDBS are applicable to data sharing mechanism in the FDDBS to preserve data consistency, they are not feasible due to different requirements.

In this paper, we use a tree structure for efficient data sharing in a FDDBS. This tree is just a logical structure and our scheme does not require the network topology to be a tree. Such logical tree structures were used to solve other problems in distributed environments. In [AA89], a logical tree is used to solve mutual exclusion problem in distributed operating systems. In [K90][AA90], it is used for the replica control in the CDDBS context.

4. Update and Access Procedures

When data D is shared, its export/import relationship in the federation is represented as an n -ary tree $T(D)$. The root node of $T(D)$ is the owner of D . All other nodes are borrowers of D . The owner and the borrowers are sharers of D . Each non-root node imports D from its parent node and each non-leaf node exports D to its children nodes. All non-leaf nodes are exporters of D , and all non-root nodes are importers of D . Each non-root node requests the current version of D only to its parent node, and each non-leaf node propagates its copy only to its direct children nodes only on their requests.

We assume that only the owner of D can update it, and all the borrowers access D in read only mode. For the owner to freely update D not concerned with the consistency of the borrowers' copies of D , version number is used. Every sharer's copy of D is assigned a version number. Whenever owner's copy is updated, its version number is increased, and when a borrower's copy is refreshed, its version number is accordingly updated. When a borrower is to access the current version of D , it first has to check with the owner to see if its copy is current.

Let $D(i)$ be the copy of D stored at site i of the network and $V(i)$ be D 's version number at site i . Now we present the procedures for the owner to update and for the borrower to access D in Figure 2(a) and 2(b), respectively. Since a node of the export/import tree is a site of the network, we use the terms node and site interchangeably. To simplify the presentation we assume that a borrower always wants to access the current version of D and that there is no system failure. The elimination of the first assumption is discussed shortly. The elimination of the second assumption, failure-freeness, is dealt with in subsection 6.3

In the access procedure when the borrower at site i requests its parent site, $\text{parent}(i)$, to send the current version of D , that is, D of $V(r)$, it invokes the refresh procedure presented in Figure 2(c) to accomplish this. It is a recursive one, and we see that the

```

procedure update (r, D) :
/* r is the owner's site number
*/
update D(r) and increase V(r)

```

(a) update procedure

```

procedure access (i, D) :
/* i is the site number of the borrower that
   accesses D
*/
check V(r) where r is the owner's site number
if V(i) < V(r) then /* outdated */
{ refresh( parent(i), V(r), D, i )
  /* request parent(i) to send D of V(r) where
    parent(i) is the parent site of site i
  */
  receive from parent(i) D of V(r) and
  update D(i) and V(i) accordingly
}
access D(i)

```

(b) access procedure

```

procedure refresh (i,v,D,j) :
/* site j requests its parent, site i,
   to send D of version v
*/
if V(i) < v then /* outdated */
{ refresh(parent(i),v,D,i)
  receive from parent(i) D of v and
  update D(i) and V(i) accordingly
}
send D(i) to site j

```

(c) refresh procedure

Figure 2 :
Update/Access/Refresh Procedures

recursion is always terminated since the root has the most current version.

An important thing to note on the access and the refresh procedures is that when a borrower site i is to access D , all the copies of D at the ancestor nodes of site i which are on the path from the root to site i are refreshed to the current version. This benefits later accesses to D by these sites because they need not refresh their copies unless further update was done to D . That is, for an ancestor node of site i , the cost of refreshing its copy and its children's copies is amortized later.

In presenting our procedures we made the assumption that a borrower always wants to access the current version of the data. Now we discuss its elimination. Users may be satisfied with the outdated version of data stored at their local databases (this decision can be made before checking the owner's version number or after having done that.) In that case, no refresh effort involving possibly large amount of data transmission is necessary, and the rapid local access is achieved. We can attach to the version number a timestamp marking the real time when the owner's copy is updated. This time need not be stamped with

the synchronized clock in the distributed environment. It can help users decide whether the data they have imported to their sites is reliable to their needs or not. The access procedure presented above can be easily extended to accomodate the outdated data access on user's will.

5. The Cost of Exporting and The Negotiation Protocol

In a FDDBS, data access negotiation between the exporter and the importer is conducted before a data object is shared. When a data object is shared, the exporter does not want to lose its site autonomy due to managing data copies exported to other sites. In this section, first we define the concept of the cost of exporting a data object to other sites. Then we present a negotiation protocol.

5.1. Cost of Data Exporting

When a site (say, site 1) wants to import data object D from another site (say, site 2), during the negotiation process site 2 computes the cost of exporting D to site 1 and if that is bearable to site 2, it allows site 1 to import D and thereafter takes responsibility to refresh D at site 1 on its request. If it turns out that site 2 has no further capability to export D to site 1, it denies exporting D . In defining the cost of exporting D , we need the parameters described in Table 1.

Consider the tree in Figure 3 representing export/import relationship. As shown, site i has exported D to site j , that is, $i = \text{parent}(j)$. Now suppose a third site (say, site k) would like to import D from site j . Suppose site j has decided to export D to site k . Then $C(j,k)$, the cost of this exporting that is to be paid by site j during a unit period of time, is

$$C(j,k) = f(k) * p(k) * \{ p(j) * C_{\text{refresh}}(i,j,R) + C_{\text{refresh}}(j,k,T) \}$$

In words, when site k wants to access D , if $D(k)$ were outdated, site k requests site j to send the current version of D incurring the cost of $C_{\text{refresh}}(j,k,T)$, and in doing so if $D(j)$ were

$C_{\text{refresh}}(i,j,X)$	The cost of refreshing $D(j)$ by sending $D(i)$ which is up-to-date on the request of site j where $i = \text{parent}(j)$. When $X = R$, it is the cost of site j receiving $D(i)$. When $X = T$, it is the cost of site i transmitting $D(i)$.
$f(i)$	The frequency of accessing D at the non-root site i during a unit period of time.
$p(i)$	The probability that when the non-root site i is to access D , $D(i)$ is in outdated state.
$C_{\text{check}}(i)$	The cost of the non-root site i checking with the root site r for the current version number of D , that is, $V(r)$.

Table 1 : Parameters

also outdated, $D(j)$ should be refreshed first. This incurs the cost of $C_{\text{refresh}}(i,j,R)$. Let

$$C_{\text{ref}}(i,j,k) = p(j) * C_{\text{refresh}}(i,j,R) + C_{\text{refresh}}(j,k,T)$$

Then,

$$C(j,k) = f(k) * p(k) * C_{\text{ref}}(i,j,k)$$

Suppose site j has exported D to site k . In the tree of Figure 3, since site i has exported D to site j , D has been imported indirectly from site i to site k by way of site j . This implies that not only site j has to pay the cost of refreshing $D(k)$ but also site i has to pay the cost of refreshing $D(j)$ on the indirect request of site k . Let this cost be $C(i,k)$. Then,

$$C(i,k) = f(k) * p(k) * p(j) * C_{\text{ref}}(h,i,j)$$

In words, site i has to refresh $D(j)$ when site k wants to access D and both of $D(j)$ and $D(k)$ are outdated. In general, for non-root/non-leaf site i and its decendent site j , that is, $j = \text{child}(\text{child}(\dots \text{child}(i))\dots)$ (see Figure 4), we obtain

$$C(i,j) = f(j) * \prod_{k=i+1}^j p(k) * C_{\text{ref}}(i-1,i,i+1)$$

where $2 \leq i \leq n-1$ and $i+1 \leq j \leq n$

For the root node 1 of Figure 4, since $D(1)$ is always current,

$$C(1,j) = f(j) * \prod_{k=2}^j p(k) * C_{\text{refresh}}(1,2,T) \text{ where } 2 \leq j \leq n$$

Now we can compute $\text{COST}(i)$, the cost of non-root/non-leaf site i exporting D to all its descendent sites.

$$\text{COST}(i) = \sum_j C(i,j)$$

where j is a descendent site of i that has imported D from site i directly or indirectly. For a leaf node i ,

$$\text{COST}(i) = 0$$

For the root node r ,

$$\text{COST}(r) = \sum_j C(r,j) + \sum_j f(j) * C_{\text{check}}(j)$$

where j is a descendent site of the root. The second term represents the cost of site j checking with the root for the current version number of $D(r)$.

5.2. Negotiation Protocol

Now we present the negotiation protocol first in words informally, and then formally in Figure 5. Consider the tree of Figure 3. Suppose site k wants to import D from site j . Then the negotiation protocol is executed as follows:

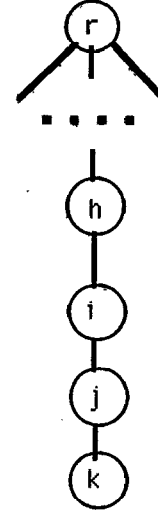


Figure 3 :
Export/Import Relationship Tree
/* site k requests site j to export data */

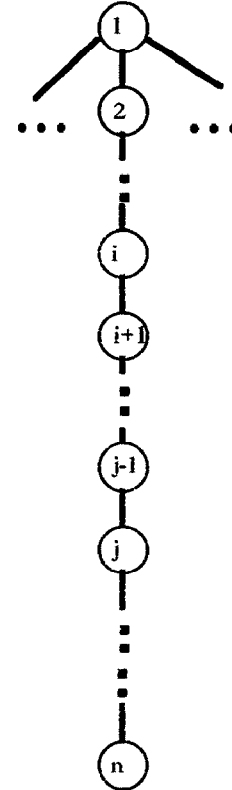


Figure 4 :
Export/Import Relationship Tree

site k : tells its intention to import D to site j.
site j : computes $C(j,k)$.
 if $COST(j) + C(j,k)$ is not acceptable,
 then notify to site k of denial of sharing.
 else { asks site i, its parent, whether exporting D to
 site k is OK with site i or not.
 if the answer is OK, then export D to site k.
 else notify to site k of denial of sharing.
 }

When site k tells its intention to import D to site j, at least $f(k)$ and $p(k)$ should be sent to site j for it to compute $C(j,k)$. As far as $p(k)$ is concerned, we assume that it can be computed by some method considering the frequency of the root updating D and $f(k)$. When site j asks site i, its parent, whether exporting D to site k is OK with site i or not, site i is supposed to do the same thing that site j have just done. That is, it computes $C(i,k)$, and checks if $COST(i) + C(i,k)$ is acceptable or not. If acceptable, it asks its parent, site h, if site j exporting D to site k is OK with site h or not, and so on. Thus, the initial negotiation between site j and site k incurs a series of further negotiations following the path from site k to the root. Eventually the root is asked the same question. Then, it has to compute $C(r,k)$ and check if $COST(r) + C(r,k) + f(k) * Ccheck(k)$ is acceptable, and so on. In Figure 5, the negotiation protocol is presented formally.

6. Evaluation of Our Scheme

In this section, we evaluate our proposal. We compare 4 export/import relationship trees. These are depicted in Figure 6. Suppose 14 sites have imported data object D from a site. These sites are numbered from 1 to 15. Site 1 is the root site, the owner, and site 2 through 15 are the borrowers. Figure 6(a) depicts the export/import relationship where all 14 sites have directly imported D from site 1. In this flat tree of height 2, all borrowers are independent of each other, competing for the refresh service of the owner. The work on data access negotiation in a FDDBS using quasy-copy [AB89] assumed this type of export/import hierarchy. The primary copy schemes for replica control in the CDDBS also assumed this type of flat propagation hierarchy except the epidemic approaches. On the other hand, trees of Figure 6(b)-(d) have height greater than 2. The tree in Figure 6(b) is a skewed tree or a unary tree, that in Figure 6(c) is a complete binary tree, and that in Figure 6(d) is a complete ternary tree. We conducted two types of experiments to compare these 4 trees. Their results are reported in subsection 6.1 and 6.2. Finally, in subsection 6.3, we discuss how our scheme deals with system failures and the effects of system failures on the flat export/import hierarchy and on the more general ones.

6.1. Experiment I

We compared the 4 trees in terms of (1) the exporting cost of the root site which is the owner of D during a unit period of time, and (2) the sum of exporting costs of all sites during a unit period of time. The unit period of time can be a minute, an hour, or a day, and so on. We made the following simplification in our comparison. First, the cost of each site checking with the root for the current version number is ignored since this cost is common to all of 4 trees. Second, we assumed that $p(2), p(3), \dots, p(15)$ are equal. Third, we assumed that the computer network on top

```

negotiation protocol (i,j,D,k) :
/* negotiation between site i and site j
  where i=parent(j) for exporting D to site
  k. site i returns to site j OK meaning that
  site i is OK for exporting D to site k or
  NOTOK.
*/
site j : starts negotiation by sending site i
  f(k), p(k), ..., p(j).
site i : computes C(i,k).
  case: i=root : if COST(i)+C(i,k)+
    f(k)*Ccheck(k) is not acceptable,
    then return NOTOK to site j.
    else return OK to site j.
  case: i≠root : if COST(i) + C(i,k) is not
    acceptable,
    then return NOTOK to site j.
    else if negotiate(parent(i),i,D,k)=OK
    then return OK to site j.
    else return NOTOK to site j.

```

Figure 5 : Negotiation Protocol

of which the FDDBS is implemented is fully connected and employs the point-to-point communication, and we let $Crefresh(i,j,X) = 1$ where $1 \leq i, j \leq 15$ and X is in $\{R,T\}$. This means that the refresh cost for any pair of site i and site j amounts to the number of transmission of D over their communication link. Finally, we let $f(i) = 1, 2 \leq i \leq 15$.

The results are depicted in Figure 7. In Figure 7(a), the exporting cost of the root site (i.e. the number of transmissions from the root) is depicted for various values of $p(i), 2 \leq i \leq 15$, all of which are set to the same value q, which varies from 0.0 to 1.0. As arity of the tree gets lower, the cost of the root decreases. The improvement of the unary, the binary, and the ternary tree over the flat tree is up to about 90% for the unary, 80% for the binary, and 70% for the ternary tree (see Figure 7(b)) where the improvement is measured as follows:

$$\frac{\text{rootcost(flat tree)} - \text{rootcost(low arity tree)}}{\text{rootcost(flat tree)}} * 100\%$$

These improvements come from the fact that the root site has to send its current copy of D only to its direct children sites which are not the entire set of borrower sites in the unary, the binary, or the ternary tree, whereas in the flat tree, every site is a direct child of the root.

In Figure 7(c), the total sum of the exporting costs of all sites in the tree (i.e. the total number of transmissions in the tree) is depicted for various values of $p(i), 2 \leq i \leq 15$, all of which are set to the same value q, which varies from 0.0 to 1.0. As the arity of the tree gets lower, the sum of the costs of all nodes increases. The degradation of the unary, the binary, and the ternary tree over the flat tree is measured as follows:

$$\frac{\text{costsum(low arity tree)} - \text{costsum(flat tree)}}{\text{costsum(flat tree)}} * 100\%$$

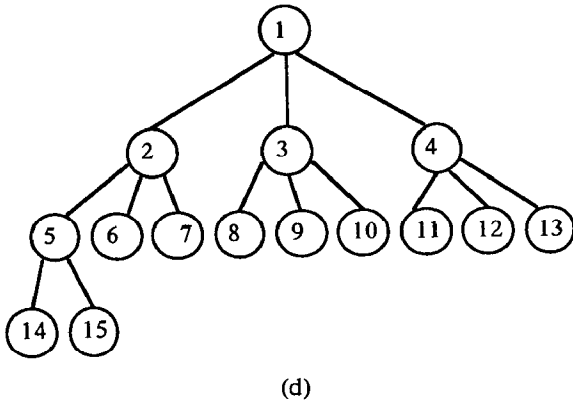
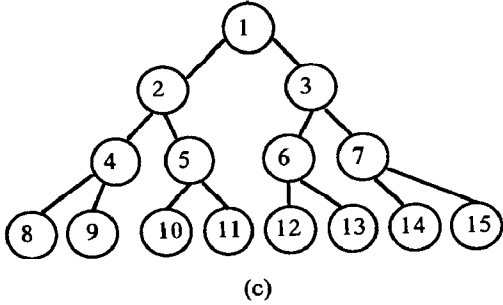
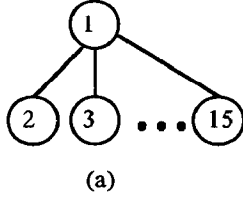


Figure 6 :
Export/Import Relationship Trees
in Experiments

It is about between 100 and 200% when q is near 0.5, and as q grows beyond that, degradation becomes drastic for the unary (up to 1300%), up to about 300% for the binary, and up to about 200% for the ternary tree (see Figure 7(d)). The maximal degradation takes place when $q = 1$. The degradation results from the fact that when a site wants to refresh its copy, in the flat tree the owner directly refreshes it incurring only 1 transmission, whereas in other trees the owner may indirectly refresh it, and thus incurring generally more than 1 transmission.

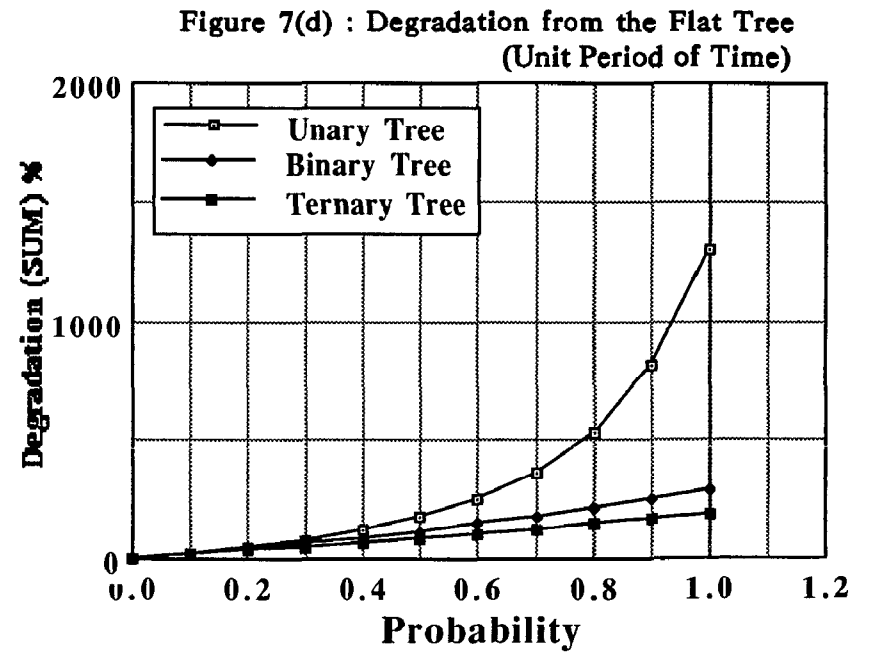
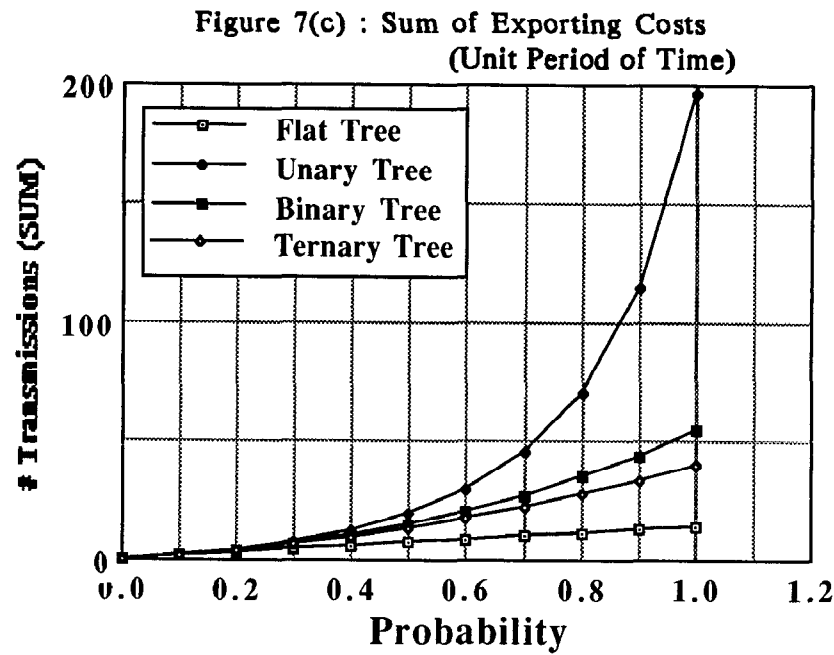
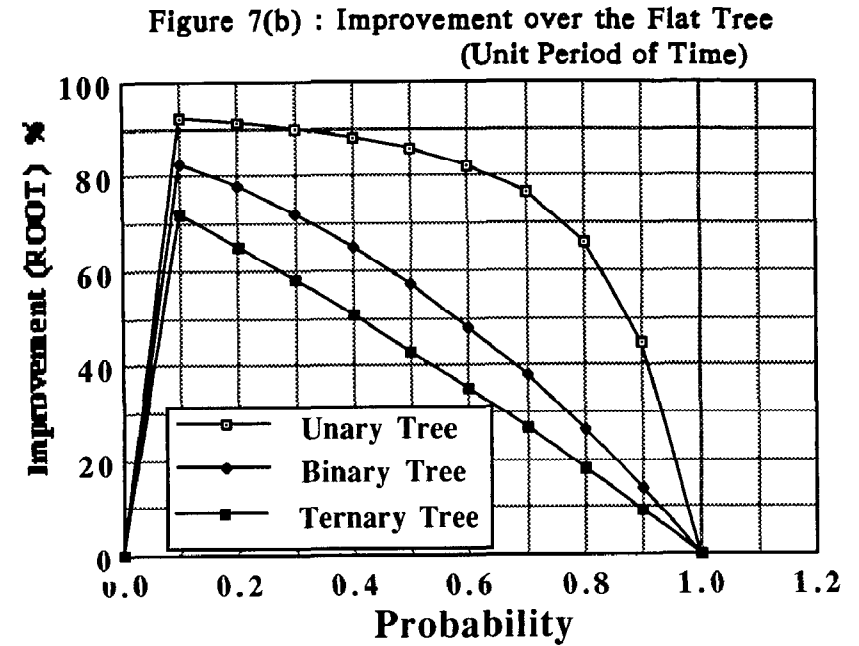
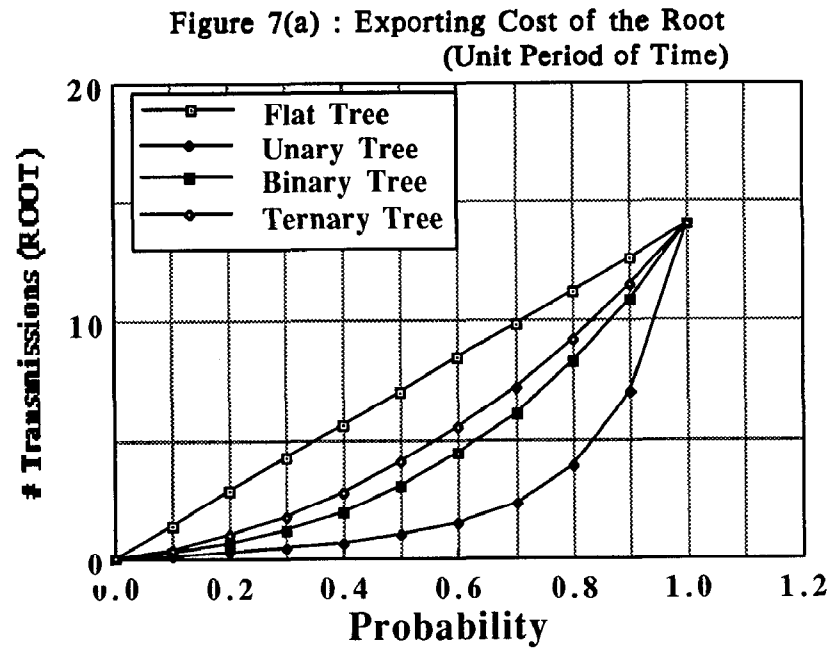
Although the degradation of the low arity trees over the flat tree is tolerable at low values of q , the degradation of the unary tree becomes severe at $q \geq 0.5$. This degradation nullifies the improvements in the cost of the root in our scheme. However, we think that the above experiment is biased against low arity trees and that the usual degradation is tolerable as argued below and indicated in the results of experiment II (see the next subsection).

In low arity trees if $D(i)$ for some site i is current, then all sites on the path from the root to site i have current version of D . If we reflected this to the values of $p(i)$'s in the experiment, some appropriate values should have been assigned to $p(i)$'s such that $p(i)$ gets less as site i gets closer to the root. Thus, experiment I is biased against low arity trees because we simply assumed that all $p(i)$'s are equal to some q . This assumption amounts to assuming very frequent updates of the owner's copy of D because $p(i)$ of site i is not at all less than $p(j)$ of site j where site j is a descendent of site i . In case the owner's copy of D is very frequently updated, the trees with low arity incurs much more number of data transmission than their high arity counterparts. For example, consider trees in Figure 6(a) and (b). Suppose site 10 wants to access current version of D . If $D(10)$ is outdated, 9 times of data transmission is required in the tree of Figure 6(b) in the worst case whereas only 1 transmission in that of Figure 6(a). In highly update intensive applications, later try to access D by other sites or site 10 usually cannot take advantage of earlier refreshing. Instead, it should initiate another series of refreshing. This waste of refreshing cost is avoided in the flat tree structure.

However, with a moderate update frequency, such earlier refreshing is amortized by later accesses and as result the total number of data transmissions in the low arity tree eventually gets close to that of the flat tree. We can show this by proving the following. Suppose there are n non-root nodes in the export/import tree. Then between two consecutive updates, the total number of data transmissions is bounded by n for all types of trees. The proof is simple. For each non-root node i , when $D(i)$ is refreshed 1 transmission is incurred from its parent and once $D(i)$ is refreshed, it does not incur further transmission to refresh $D(i)$. Since there are n non-root nodes, n is the upper bound of the total number of data transmissions between any two consecutive updates.

Thus, the increase in the total number of data transmissions in the low arity trees compared to that in the flat tree is not so severe as indicated in the above experimentation results.

After all, from these results, we can conclude that by lowering the arity and thus lengthen the height of the export/import tree, we can alleviate the bottleneck phenomenon at the owner's site at the cost of incurring more transmission of the current version of D than in the flat export/import tree. This result implies that when no more copy of D can be directly



exported from the owner's site to other sites, D can be indirectly exported to more number of sites using export/import tree of height greater than 2. Thus, the degree of data sharing in a FDDBS is enhanced.

6.2. Experiment II

In this experimentation, we have conducted simulation to compare the 4 trees of Figure 6 in terms of (1) the exporting cost of the root between two consecutive updates of D, and (2) the sum of exporting costs of all sites between two consecutive updates of D. The results are depicted in Figure 8. In Figure 8(a), the exporting cost of the root (i.e. total number of transmissions from the root) is depicted for various values of the total number of accesses to D by all non-root sites in the tree between two consecutive updates of D. The number of accesses varies from 0 to 100. We obtained the cost by averaging the number of transmissions incurred in each of 100 update intervals. In each update interval, the sites accessing D were chosen randomly.

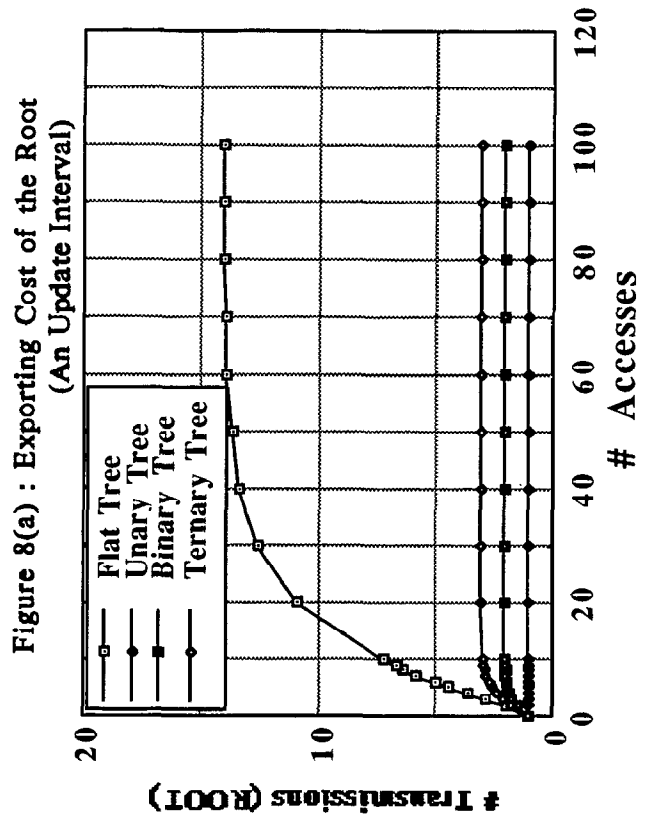
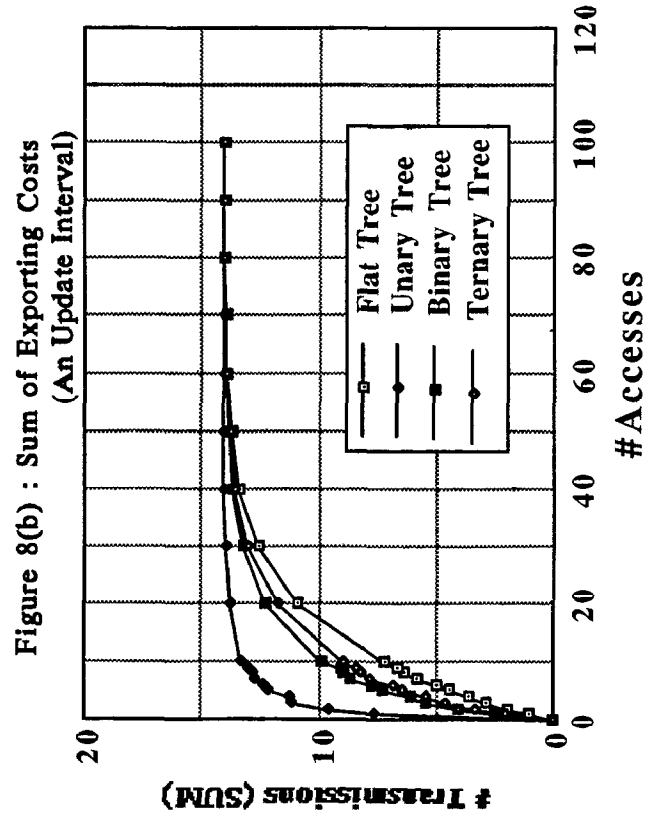
Since there are 14 importers, in the flat tree the cost of the root is bounded by 14. Similarly, it is bounded by 1, 2, and 3 in the unary, the binary, and the ternary tree, respectively. The improvement of the low arity trees over the flat tree becomes large as the number of accesses increases.

In Figure 8(b), the sum of the exporting costs of all sites (i.e. total number of transmissions in the tree) is depicted for various values of the total number of accesses to D by all non-root sites in the tree between two consecutive updates of D. The number of accesses varies from 0 to 100. The simulation was conducted in the same way as in Figure 8(a). As proven in subsection 6.1, for all trees the cost sum is bounded by 14, which is the number of all importers. For small number of accesses (i.e. for high update frequency as mentioned in subsection 6.1), the degradation of the low arity trees over the flat tree is large but it converges to 0 as the number of accesses increases.

6.3. Data Sharing in System Failures

When system failure including site crash, communication link failure, and network partition occurs, the update and the access procedures and the negotiation protocols cannot be executed as described in section 4 and 5. We assume that when system failure presents, the export/import hierarchy is not changed. It implies that a borrower can access only what its parent can bring to it. Suppose data object D is shared as in the tree of Figure 4 and there is no failure except that site i has crashed. Then site j, a descendent of site i, can refresh D(j) only to the version V(i+1). If the users at site j are satisfied with that, then they still can access D of V(i+1) which is always more current than D(j). All non-root sites of the subtree whose root is site i have similar partial operability. This type of partial operability is achieved in presence of the communication failures too.

On the other hand, if the export/import hierarchy is the flat tree, then the crash of the root, the owner, forces all importers to be satisfied with what they store in their local databases. Besides, our scheme can be extended such that a site which cannot access the current version of the shared data due to the failures renegotiate data access with other databases. These advantages results from the flexibility of the export/import hierarchy of our scheme which is not flat.



7. Concluding Remarks

In this paper, we proposed an efficient export/import scheme for data sharing in the FDDBS. Our scheme employs a hierarchical export/import relationship where a borrower is allowed to export the data to other borrowers on behalf of the owner, and thus reduces the bottleneck phenomenon at the owner's site due to all borrowers competing for the owner's service to refresh their imported data. This bottleneck phenomenon can be serious when the export/import relationship is a flat tree structure, that is, when all borrowers directly import data from the owner. For our hierarchical export/import scheme, we devised the procedures for the owner to update the shared data, and for the borrower to access the imported data, and the negotiation protocol for data exporting/importing between two databases in the federation.

We introduced the concept of the cost of data exporting, and derived equations to quantify it. For each database in the federation, if it has exported some data, it can compute the cost of exporting that it has to pay. Knowing this cost helps each database decide in negotiation whether it can export data further or not.

We conducted two experiments to evaluate our scheme. The results of the first experiment showed that using hierarchical data export/import, we can alleviate the bottleneck phenomenon at the owner's site at the cost of incurring more transmission of the current version of the shared data among the databases in the federation than using the flat export/import relationship. The second experiment, however, showed that with a moderate frequency of the owner updating the shared data, the increase of total amount of data transmission in our scheme is not severe.

Future research issues include the following:

- (1) Our scheme deals with system failures passively, that is, even when failure is detected, the export/import relationship is not changed. More dynamic scheme can be devised by allowing the operable sites to stop the current export/import contracts and renegotiate with other databases to suit their needs.
- (2) An incremental update propagation scheme can be incorporated to our scheme. When an exporter sends the new version of the data to an importer, only the difference between the previous and the current version can be sent to reduce the transmission cost. Techniques proposed in recent works on the incremental refresh of snapshots [L86], materialized views [RK86a][RK86b][SP89] in distributed environments can be applied to our scheme.

8. References

- [AA89] D. Agrawal and A. Abbadi, "An Efficient Solution to the Distributed Mutual Exclusion Problem," Proc. ACM Symp. on Principles of Distributed Computing, 1989, pp. 193-200.
- [AA90] D. Agrawal and A. Abbadi, "The Tree Quorum Protocol: An Efficient Approach for Managing Replicated Data," Proc. Int'l Conf. on VLDB, 1990, pp. 243-254.
- [AB89] R. Alonso and D. Barbara, "Negotiating Data Access in Federated Database Systems," Proc. Int'l Conf. on Data Engg., 1989, pp. 56-65.
- [AL80] M. Adiba and B. Lindsay, "Database Snapshots," Proc. Int'l Conf. on VLDB, 1980, pp. 86-91.
- [BSR80] P. Bernstein, D. Shipman and J. Rothnie, "Concurrency Control in a System for Distributed Databases (SDD-1)," ACM Trans. on Database Syst., 5,1, Mar. 1980, pp. 18-51.
- [D83] C. Date, "An Introduction to Database Systems: Vol. II," Addison-Wesley Pub. Co., 1983, Section 7.6, pp. 306-309.
- [DG87] A. Demers et. al., "Epidemic Algorithms for Replicated Database Maintenance," Proc. ACM Symp. on Principles of Distributed Computing, 1987, pp. 1-12.
- [G79] D. Gifford, "Weighted Voting for Replicated Data," Proc. ACM Symp. on Operating Systems Principles, 1979, pp. 150-159.
- [HM85] D. Heimbigner and D. McLeod, "A Federated Architecture for Information Management," ACM Trans. on Office Inf. Syst., 3,3, Jul. 1985, pp. 253-278.
- [K90] A. Kumar, "Performance Analysis of a Hierarchical Quorum Consensus Algorithm for Replicated Objects," Proc. Int'l Conf. on Distributed Computing Systems, 1990, pp. 378-385.
- [L86] B. Lindsay et al., "A Snapshot Differential Refresh Algorithm," Proc. ACM SIGMOD Int'l Conf. on Management of Data, 1986, pp. 53-60.
- [RK86a] N. Roussopoulos and H. Kang, "Preliminary Design of ADMS+ : A Workstation-Mainframe Integrated Architecture for Database Management Systems," Proc. Int'l Conf. on VLDB, 1986, pp. 355-364.
- [RK86b] N. Roussopoulos and H. Kang, "Principles and Techniques in the Design of ADMS+," IEEE Computer, 19, 12, Dec. 1986, pp. 19-25.
- [S79] M. Stonebraker, "Concurrency Control and Consistency of Multiple Copies of Data in Distributed INGRES," IEEE Trans. on Softw. Engg., 5, 3, May 1979, pp. 188-194.
- [SP89] A. Segev and J. Park, "Updating Distributed Materialized Views," IEEE Trans. on Knowledge and Data Engg., 1, 2, Jun. 1989, pp. 173-184.
- [T79] R. Thomas, "A Majority Consensus Approach to Concurrency Control for Multiple Copy Databases," ACM Trans. on Database Syst., 4, 2, Jun. 1979, pp. 180-209.
- [W89] A. Wolski, "LINDA: A System for Loosely Integrated Databases," Proc. Int'l Conf. on Data Engg., 1989, pp. 66-73.