Scheduling the Allocation of Data Fragments in a Distributed Database Environment: A Machine Learning Approach

Alok R. Chaturvedi, Ashok K. Choubey, *Member, IEEE,* and Jinsheng Roan

***Abstract*— Different database fragmentation and allocation strategies have been proposed to partially replicate data in a partitioned, distributed database (DDB) environment. The replication strategies include database snapshots, materialized views, and quasi-copies. These strategies are 'static' and do not adapt to the changes in the data usage patterns. Furthermore, they often require expensive update synchronizations to maintain data consistency and do not exploit the knowledge embedded in the query history.**

This paper describes a machine learning based time invariant fragmentation method (MLTIF) that acquires knowledge about the data usage patterns for each node. Based on this knowledge, MLTIF designs time invariant fragments (TIF) and schedules its allocation and selective update for a specified time period. Simulation is used to compare the effectiveness of the MLTIF approach with that of full replication, materialized views, and non replication strategies. Initial results indicate that for most normal operating conditions, the MLTIF approach can be effective.

***Index Terms*— Machine learning, distributed database, time invariant fragmentation, scheduling.**

I. Introduction

In a DISTRIBUTED DATABASE SYSTEM (DDBS), geographically dispersed databases are interconnected by a computer network, and their administration is done by a distributed database management system (DDBMS) in such a way that the distribution of logical and physical components of the databases are transparent to the user. The interest in (DDBS) research is motivated primarily by reliability, performance, and economic concerns. The reliability concern pertains to making the DDBS fault-tolerant; performance concerns include reducing query response time and increasing throughput; and economic concerns include reducing data communication and update synchronization costs.

To realize the above objectives, the database may be partitioned into a number of non-overlapping fragments and allocated over the network. Different strategies have been adopted to allocate data fragments in a DDBS. One data allocation strategy permits a single copy of the database to be stored in the network (no replication). Here, the database or its fragments are allocated to the nodes that minimize the overall system communication cost, query response time, and/or other criteria depending upon the objectives of the database designer [6], [18].

Manuscript received November 25, 1992. Review of this manuscript was processed by Editor S. S. Erenguc.

A. R. Chaturvedi is with the Krannert Graduate School of Management, Purdue University, West Lafayette, IN 47907.

A. K. Choubey is with the Information Systems—Data Management Applications, Bell Atlantic, Freehold, NJ 07728 USA.

J. Roan is with thr Institute of Business Administration, National Chung Cheng University, Min-Hsiung Hsiang, Chia-Yi, Taiwan, R.O.C.

IEEE Log Number 9400867.

Another strategy involves storing multiple copies of all (complete replication) or a part (partial replication) of the database across the network [3], [5], [7], [10], [11], [17]. Although this reduces transmission costs and response time, it increases data redundancy, storage costs, and update costs. Several partial replication techniques have been proposed. Database snapshots are the read-only replica of a selected portion of the database [1]; materialized views are stored copies of the result of retrieving views from the database [4]; and quasi files are the portions of database stored in the cache memory at the user nodes [2].

Although these techniques have proven merits, they have the following shortcomings:

1) They are 'static' techniques, and do not adapt to the changes in the usage pattern in determining the content of the data. They assume the data usage is constant, while in reality, requirements tend to change frequently. Moreover, these techniques do not utilize the knowledge hidden in the query history of each node.
2) At a given time, data in a base table may be inconsistent with that of its copies. To overcome this problem, the above techniques require expensive update synchronizations [2], [4], [12]–[14], [20].
3) In practice, most DDBS have attributes of an entity whose values do not change for lengthy periods of time. For instance, a typical customer relation of a commercial database has attributes, such as *customer number, name, address,* and *telephone*, whose values are relatively 'static.' This is an important property of data, and is called *time invariance.* This property is not exploited by any of the above methods.

The goal of this paper is to develop an adaptive method, based on the time invariance concept, that can autonomously detect data usage patterns from the query history of the given database, identify time-invariant fragments and their respective time windows, and allocate these fragments to the nodes such that data communication and update synchronizations costs are minimized. Re-stating the problem concisely,

Given: (1) *A non-replicated distributed database, and*
(2) *Query history for each node in the network*
Find: (1) *Acceptable time intervals*
(2) *Time invariant fragments (TIF's)*
(3) *Schedule for TIF allocation and update.*

II. Design and Allocation of Time Invariant Fragments

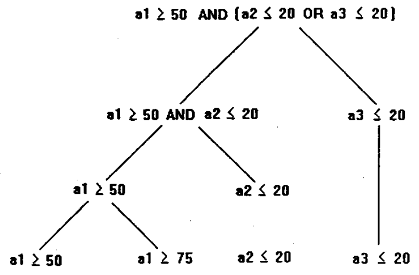
A time invariant fragment (TIF) is a partition of a base relation whose contents are 'static' during a specified time interval. In other words, the values of each component attribute in a TIF are constant throughout this time interval. These attributes are called *Time-Invariant Attributes* (TIA's), and the remaining attributes, *Time-Sensitive Attributes* (TSA's). A formal definition of TIA and TSA is given below.

Let e_{ij} be the jth attribute in the ith tuple of a database relation R. Then the value sets of R and its jth attribute, a_j, can be defined, in terms of e_{ij}, as follows:

$R = \{e_{ij} \mid i = 1, 2, \ldots, m; j = 1, 2, \ldots, n\}$,
$a_j = \{e_{ij} | i = 1, 2, \ldots, m\}$.
a_j is a TSA for time interval T
if $\exists\ (v(e_{ijt}) \neq v(e_{ijt'}))$
$i \in \{1, 2, \ldots, m\}$
$t, t' \in T$
$t \neq t'$
a_j is a TIA for time interval T
if $\forall(v(e_{ijt}) = v(e_{ijt'}))$
$i \in \{1, 2, \ldots, m\}$
$t, t' \in T$
$t \neq t'$
where: $v(e_{ijt})$ is the value of e_{ij} at time t,
m is the cardinality of R, and
n is the degree of R.

TIF's are constructed from the query history of the entire system. Some queries are periodic while others are *ad hoc*. The periodic queries, typically, follow patterns; for example, monthly sales statistics are inquired at the end of every month. Some periodic queries tend to appear together such as the group of queries required for the generation of a monthly sales report by geographic area, customer type, product type, and sales personnel. Some of these queries may access a large amount of common data. If the update of the database is not intensive in the time interval within which data access takes place, it could be beneficial to retrieve the common/shared time-invariant, remote data in one attempt.

The need to acquire knowledge about the retrieval patterns from query history suggests the use of a machine learning technique. One of the roles of machine learning is to seek to acquire knowledge from available data and use it to create new theories about the domain in question, in an entirely automated manner. Machine learning techniques employ a small number of extremely general induction strategies coupled with some basic domain knowledge. The domain knowledge may involve structural descriptions, procedural explanations, or even discoveries of new domain concepts. Due to the inductive nature of reasoning involved there is always some possibility of error. Consequently, most techniques allow for self-improvement through disconfirmatory feedback. Many learning strategies have been devised to solve problems in different domains [8]. Learning strategies include learning by being told, learning from example, learning from observations, case-based learning, analogical learning, and explanation-based learning.



Q1: SELECT a1, a3
FROM R1
WHERE a1> 50
AND a2< 20

Q2: SELECT a1, a2
FROM R1
WHERE a1> 75
AND a3< 30

Fig. 1. Conceptual aggregation.

One of the induction based strategies, learning from observation, is adopted for the design of time invariant data fragments in a distributed database environment. The technique, called machine learning based time invariant fragmentation (MLTIF), has its roots in goal directed conceptual aggregation (GDCA), developed by Chaturvedi et al [8]. The creation of a TIF is a problem solving exercise for MLTIF. From the timestamps on the queries an initial time slice is determined. Each query in the history is then decomposed into sub-expressions. Next, patterns of data retrieval and modification is generated from the sub-expressions. Similar patterns are aggregated to form the most general concepts. An example to demonstrate this process is given below.

Example 1: Here we show how MLTIF creates aggregate concepts from queries. Suppose there are two queries, $Q1$ and $Q2$ in the query history. These queries are first decomposed into sub-expressions such as $a1 \geq 50, a2 \leq 20$. Using aggregation operators (AND, OR, etc.,), these expressions are converted into higher level concepts till a single concept covers all data requirements for a relation (Fig. 1). Finally, the highest level concept is used to create a query to the base table for creating the fragment.

The concepts, generated by MLTIF are evaluated using a cost-based evaluation function. These steps are repeated till the least cost time slice and most general concepts are determined. Finally, TIF's for the time slice, and for each node, will be created from the base tables using the most general concepts. The contents of TIF's are illustrated in Fig. 2.

The creation of proprietary TIF's for each node in the network could lead to the sub-optimality and computational intractability problems. When a number of nodes have high degrees of commonality between their respective TIF's, the likelihood of a large amount of common data being transmitted to individual nodes increases, as do the update synchronization costs. In addition, as the size of the network increases, the TIF approach may become computationally intractable. In order to reduce the costs and computational complexity, network nodes

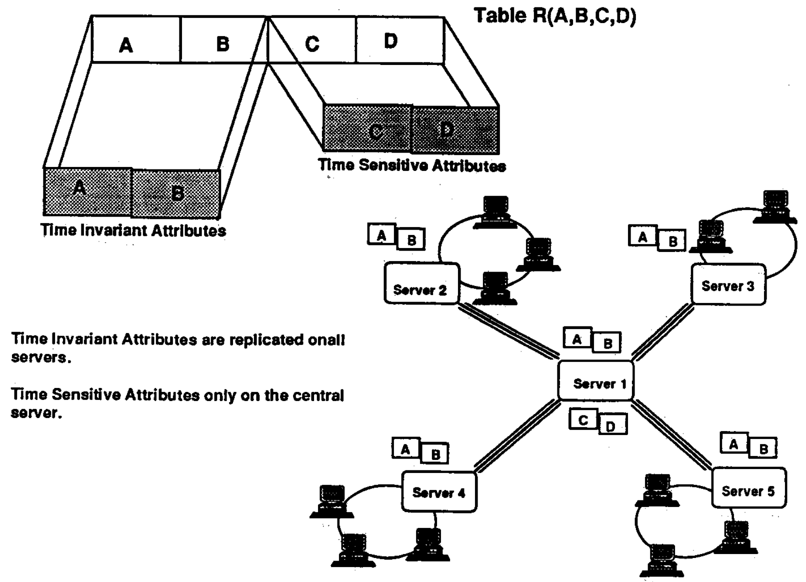


Table R(A,B,C,D)
A
B
C
D
Time Sensitive Attributes
Time Invariant Attributes
Time Invariant Attributes are replicated onall servers.
Time Sensitive Attributes only on the central server.
Server 1
Server 2
Server 3
Server 4
Server 5

Fig. 2. The contents of TIF's.

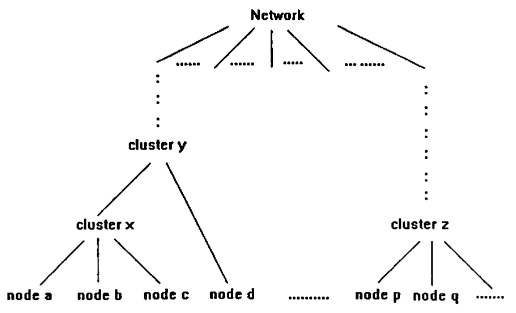


Fig. 3. TIF hierarchy.

are clustered according to the closeness of their locations and to the similarity in their query patterns (Fig. 3). Nodes with similar query patterns in a geographic area are assigned to the same cluster. For each cluster, one of the nodes may be selected or a new data server node may be created to store the common TIF. The common TIF may be retrieved by the member nodes in the cluster and may be refreshed when update synchronizations take place. Multiple levels of node clustering may be required. In such an event, two or more clusters may be combined into a larger cluster, and the TIF's are created for the member clusters in the same fashion as that for the nodes. The TIF hierarchy grows as long as it is computationally necessary. The increases in data retrieval costs and response times are offset by the decreases in TIF creation costs, data storage costs, and update synchronization costs.

III. Algorithm for Creation and Allocation of TIF's

The premise of the MLTIF approach is its capability of detecting the patterns that are likely to vary over time. As discussed above, a static methodology does not work in the ever-changing environment. A new approach that allows dynamic, continuous, and automatic detection of query patterns of a user node is required to provide the benefits discussed previously. Conceptual aggregation allows the computer program to aggregate observations (query statements) and provide meaningful explanation of aggregate concepts (query patterns) formed. Situations that are not currently handled are:

a) *Queries With Statistical Operations.* Attributes with statistical function such as SUM, MAX, and MIN, etc., are excluded from query statements. These operations usually involve simple values to be sent over from the remote base tables as a result of such functions. Our approach creates TIF's that contain fragments of remote base tables which would unnecessarily retrieve 'raw data' and increase communication cost. Therefore it would be more appropriate to set aside attributes with these statistical functions in the creation of TIF's.
b) *Updates With Insertion and Deletion.* Deletion and insertion of tuples are usually done periodically in batch mode. These update queries are excluded from this research without loss of generality.
c) *Data Movement.* Update with modification that causes data to move from one base table to another that is stored at different site. This is usually found in the change in the value of an attribute whose values are the basis of a horizontal partitioning of the database table. When the new value exceeds the value range of a base table, the entire tuple has to be removed from this table and added to an appropriate one. This type of update is an equivalence of an deletion followed immediately by an insertion.

The following algorithm is used by MLTIF to determine the TIF's for each site based on the query histories of the entire system.

Step 1. Classify the queries of each site into retrieval and update queries. Further classify the update queries into insertion, deletion, and modification.

Step 2. Classify the retrieval queries of each site into local-access and remote-access based on the location of data these queries address. Classify modification queries in a similar fashion.

Step 3. Transmit all the remote-access modification queries for each base table to their respective sites. For each base table, use these remote-access modification queries and the local-access modification queries to determine the time sensitive and invariant attributes.

Step 4. Remove time sensitive attributes from each remote-access retrieval query if these attributes are time sensitive in their remote base tables. All remaining attributes in this remote-access retrieval query are time invariant.

Step 5. Apply aggregation technique to construct the query for the creation of a TIF from a remote base table for each site. This is based on the entire remote-access retrieval queries for the base table at this site.

We now explain each step in detail.

Step 1. The type of database access of each query is identified. In SQL query language, a query start with a 'SELECT' is a retrieval, an 'UPDATE' a modification, an 'INSERT' an insertion, or a 'DELETE' a deletion query. Only retrieval and modification queries are used in the following steps for the determination of TIF's. Insertion and deletion queries are discarded.

Step 2. A retrieval or modification query in a partitioned, non-replicated database environment may refer to a data attribute of one of the following three types:

A. The attribute and all the data values are locally stored in a base table.
B. The attribute is not stored in any local base table.
C. The attribute is stored in a local base table but the referred data values are not stored or partially stored in this table. The data values that are not locally stored may spread over more than one remote base tables.

A type A attribute is either an attribute that is included in a vertically partitioned local base table or one that is used as the only attribute for horizontal partitioning of the original non-partitioned table and the values of this attribute in the local base table are the super-set of the required values of this attribute in the query statement. A type B attribute is an attribute that is not stored in the local base table which is a fragment of a vertically partitioned table. A type C attribute is an attribute whose values in the local base table are not the super-set of the required values in the query statement. This can be a result of a horizontal or a mixed partitioning of the original table.

All attributes in the SELECT part (selecting attributes) of a retrieval query or in the SET part (modifying attributes) of a modification query, or in WHERE part (restricting attributes) of both types of queries will be assigned one of these three types accordingly.A query is 'local-access' if all its attributes are of type A, is 'remote-access' if all its attributes are of type B, and otherwise is 'mixed-access'. Dealing with a mixed-access query is not as straightforward, as it is successively decomposed into multiple sub-queries till each subquery can be classified as a local-access or a remote-access query.

All attributes of a mixed-access query are first assigned into one of the two groups: group A in which all attributes are of type Aand group B in which all attributes are of type B. An attribute of type C is assigned to both groups if the location of the base table containing its value cannot be identified from the database fragmentation conditions. An attribute that is not used to horizontally partition a database is usually of this type unless other restricting attributes in the same query statement can be used to assist the identification of the data source locations. We will exclude the selecting or modifying attributes with no restricting attribute stored in the same group. For a vertically partitioned database table, this situation implies that join operations are required for this query. For a horizontally partitioned database table, this means that there is not data to be accessed from this group of attributes. The union of all the local attributes in group A is still local and a query statement can be constructed to retrieve these data values from the local base table. This query is 'local-access'. A 'remote-access' query statement can be constructed from attributes in group B in the same fashion for the remote data of mixed-access query attributes.

Step 3. Remote-access modification queries are transmitted to the sites of the base tables they modify. The union of modifying attributes of both remote-and local-access queries for each base table will include some values that would vary in this time slice. These attributes are therefore 'time sensitive'. The supplement set of the attributes of this base table is 'time invariant'. The information of time sensitive attributes of a base table is sent back to all the sites with queries that modify this base table.

Step 4. Since a time sensitive restricting attribute will cause invalid selection of data fragment, this attribute should be eliminated from the remote-access retrieval queries. All time sensitive selecting attributes are dropped from these query statements. A remote access retrieval queries becomes invalid if there is no restricting or selecting attributes left in the query after the removal of time-sensitive attributes. Therefore, all the attributes in the queries are time-invariant, and will be used as the basis for constructing query statements for the creation of TIF's. The reason we did not further consider insertion and deletion queries after Step 1 is now clear. These two types of queries always change the content of base tables.

Step 5. At each site, all queries now consist of time-invariant attributes only. The remote access retrieval queries are grouped by base tables and for each base table all the queries are decomposed into sub-expressions. MLTIF takes all the sub-expressions corresponding to all the restricting attributes of the queries for a base table and aggregate them to form a higher level concept of the restricting attributes iteratively until no further aggregation is required. This final highest concept is used to construct the query to create a TIF of that remote base table. The restricting attributes that are not part of selecting attributes are added into the selecting attribute list because the restricting condition of a restricting attribute is an aggregation of the individual conditions. TIF's of the same domain and structure are integrated into a single TIF. All the TIF's for a site can be constructed in this manner. An example illustrating MLTIF was presented in Section I.

A. Illustrative Examples

We now illustrate this procedure by two examples. The first example shows the creation of a TIF in a horizontally partitioned database environment and the second example shows the creation of a TIF in a vertically partitioned database environment.

Example 2: Suppose two sites in a network each stores a horizontally partitioned database table as follows:

```
CUST(CNUM, CNAME, CITY, CYTD_ORDER,
  CREDIT_LINE, CREDIT_USED)
```

Site A: stores the fragment with CITY = '*A*', and
Site B: stores the fragment with CITY = '*B*'.

The query histories are as follows:

Site A:

```
1. SELECT CNUM, CNAME, CYTD_ORDER
   FROM CUST
   WHERE CYTD_ORDER > 1000000
   AND CITY = 'A'
2. SELECT CNUM, CNAME, CYTD_ORDER
   FROM CUST
   WHERE CYTD_ORDER > 1500000
   AND CREDIT_LINE < 100000
3. SELECT CNUM, CNAME, CREDIT_LINE,
   CREDIT_USED FROM CUST
   WHERE CREDIT_LINE > 150000
4. UPDATE CUST
   SET CYTD_ORDER = CYTD_ORDER - 200
   WHERE CNAME = 'JBM'
   AND CITY = 'A'
```

Site B :

```
5. SELECT CNUM, CNAME, CREDIT_LINE
   FROM CUST
   WHERE CREDIT_LINE > 200000
6. UPDATE CUST
   SET CREDIT_USED = CREDIT_USED-200 WHERE
   CNUM = 'N321'
7. INSERT
   INTO CUST (CNUM, CNAME, CITY, CYTD_ORDER,
      CREDIT_LINE, CREDIT_USED)
   VALUES (N938, 'ABC INC.', 'B', 10000, 1000000, 0)
```

The creation procedure of the TIF's of CUST relation for both sites is as follows:

Step 1:

Classify query type.
insert queries: 7
delete queries: none
modification queries: 4, 6
retrieval queries: 1, 2, 3, 5
query 7 will be discarded since it is an INSERT query.

Step 2: Classify local-/remote-access queries.

Retrieval queries:

Site A:

```
Local-access: 1, 2, 3 /* because it is transparent
                        whether
Remote-access: 2, 3     CYTD_ORDER > 1500000
                        and
                        CREDIT_LINE < 100 000 in
                        query 2 are local or remote
```

Site B:

Local-access: 5
Remote-access: 5

Modification queries:

Site A:

```
Local-access: 4
Remote-access: none /* because CITY = 'A' cannot
                       be divided into two fragments */
```

Site B:

Local-access: 6
Remote-access: 6

Step 3: Identify time sensitivity.

Site A: Time-sensitive: CYTD_ORDER, CREDIT_USED
Time-invariant: all other attributes

Site B: Time-sensitive: CREDIT_USED
Time-invariant: all other attributes

Step 4: Remove time-sensitive attributes from remote-access retrieval queries.

Site A: remote-access query

```
Query 2: SELECT CNUM, CNAME, CYTD_ORDER
         FROM CUST
         WHERE CYTD_ORDER > 1500000
         AND CREDIT_LINE < 100000
Query 3: SELECT CNUM, CNAME, CREDIT_LINE
         FROM CUST
         WHERE CREDIT_LINE > 150000
```

Site B: Remote-access retrieval query

```
Query 5: SELECT CNUM, CNAME, CREDIT_LINE
         FROM CUST
         WHERE CREDIT_LINE > 200000
```

Step 5: Construct the queries to create TIF's

MLTIF generates the following remote queries to create TIF's for both sites.

```
Site A: SELECT CNUM, CNAME, CREDIT_LINE
        FROM CUST
        WHERE CYTD_ORDER > 1500000
        AND (CREDITLINE < 100000 OR
                CREDIT_LINE > 150000)
```

Site B: There is only one remote-access query. MLTIF will not generate a new query.

Example 3: Suppose two sites in a network each stores a vertically partitioned database table as follows:

CUST(CNUM, CNAME, CITY, CYTD_ORDER, CREDIT_LINE, CREDIT_USED)

Site A: stores the fragment **CUST(CNUM, CNAME, CYTD_ORDER)**

Site B: stores the fragment **CUST(CNUM, CITY, CREDIT_LINE, CREDIT_USED)**

The query histories are as follows:

Site A:

```
1. SELECT CNUM, CNAME, CYTD_ORDER
   FROM CUST
   WHERE CYTD_ORDER ≥ 1000000
```

```
2. SELECT CNUM, CYTD_ORDER, CREDIT_LINE
   FROM CUST
   WHERE CYTD_ORDER ≥ 1000000
   AND CREDIT_LINE < 100000
   AND CITY = 'A'
3. SELECT CNUM, CNAME, CYTD_ORDER,
   CREDIT_LINE
   FROM CUST
   WHERE CYTD_ORDER ≥ 10000000
   AND CREDIT_LINE < 500000
   AND CITY = 'B'
   Site B:
4. SELECT CNUM, CNAME, CREDIT_LINE,
   CREDIT_USED, (CREDIT_USED / CREDIT_LINE)
   FROM CUST
   WHERE CREDIT_USED ≥ CREDIT_LINE * 0.9
5. UPDATE CUST
   SET CREDIT_USED = CREDIT_LINE - 2000
   WHERE CNUM = 10555
6. SELECT CNUM, CNAME, CYTD_ORDER,
   CREDIT_LINE, CREDIT_USED
   FROM CUST
   WHERE CNUM = 10555
7. INSERT
   INTO CUST(CNUM, CNAME, CITY,
   CREDIT_LINE, CREDIT_USED, CREDIT_ORDER)
VALUE (10056, 'ABC INC.', 'NEW YORK', 10000, 0, 0)
```

Step 1: Classify query type.

insert queries: 7
delete queries: none
modification queries: 6
retrieval queries: 1, 2, 3, 4, 5
query 7 will be discarded since it is an INSERT query.

Step 2: Classify local-/remote-access queries.

Retrieval queries:
Site A: Local-access: 1, 2, 3
Remote-access: 2, 3
Site B: Local-access: 6
Remote-access: 6

(Note: Query 4 is discarded because the remote set contains no restricting attribute.)

Modification queries:
Site A: None.
Site B: Local-access: 5
Remote-access: none

Step 3: Identify time sensitivity.

Site A: Time-sensitive: none
Time-invariant: all attributes
Site B: Time-sensitive: CREDIT_USED
Time-invariant: all other attributes

Step 4: Remove time-sensitive attributes from remote-access retrieval queries.

Site A: remote-access queries

```
Query 2:  SELECT CNUM, CREDIT_LINE
          FROM CUST
          WHERE CREDIT_LINE < 100000 /*because
                                CYTD_ORDER >=
          AND CITY = 'A'   1000000 is not in
                                remote set*/
Query 3:  SELECT CNUM, CREDIT_LINE
          FROM CUST
          WHERE CREDIT_LINE < 500000
          AND CITY = 'B'
```

Site B: remote-access query

```
Query 6:  SELECT CNUM, CNAME, CYTD_ORDER
          FROM CUST
          WHERE CNUM = 10555
```

Step 5: Construct the queries to create TIF's.

MLTIF generates the following remote queries to create TIF's for both sites.

```
Site A: SELECT CNUM, CREDIT_LINE, CITY
        FROM CUST
        WHERE CREDIT_LINE < 500000
        AND (CITY = 'A' OR CITY = 'B')
```

(Note: CITY is added into the selecting attribute list.)

Site B: There is only one remote-access query. MLTIF will not generate a new query.

B. TIF's for Retrieval Queries With Two-way Join

MLTIF processes two-way join queries in the following manner (all the other steps remain the same):

A. Determine TIF content by evaluating two-way join retrieval queries one be one.

Step A.1 Split the two-way join query into two single-table sub-queries (by dropping the join condition clause and splitting the selecting attributes) for the tables involved. Send each sub-query, along with the join condition clause to the nodes containing a copy of the base table involved. Some nodes might receive both sub-queries.

Step A.2 Evaluate sub-queries. Determine from the local base table the join attribute values. Put these values in a set, called *unique join-value set.* Transmit the set to the nodes receiving the other sub-query. Every node receiving a sub-query should have remote join values for the other sub-query.

Step A.3 Take the union of the remote unique join-value sets and the local unique join-value set, if any, at each node. Determine the data items satisfying the original two-way join query from the resultant set and the local base table.

Example 4: Suppose the two nodes in a network store data as follows:

$T1(\underline{a}, b, c, d, e, f)$ where a is a primary key,
$T2(\underline{f}, g, h)$ where f is a primary key,
Node A: stores the following base table fragment of $T1$ with $e =$'A',

	$\underline{a}$	b	c	d	e	f
tuple 1	$C001$	Acme	1000	200	A	S01
tuple 2	$C003$	Erso	500	100	A	S01
tuple 3	$C004$	Giant	100	100	A	S02

and base table $T2$:

	f	g	h
tuple 1	$S01$	Cocki	F
tuple 2	$S02$	Sam	M
tuple 3	$S03$	George	M
tuple 4	$S04$	Diane	F

Node B: stores the following base table fragment of $T1$ with e =‘B’.

	a	b	c	d	e	f
tuple 1	$C002$	Dons	2000	800	B	$S03$
tuple 2	$C005$	Gray	800	600	B	$S04$

The query history at each node is as follows:
Node A:

```
1. SELECT a, b, c
   FROM T1
   WHERE c > 1000

2. SELECT a, b, c, d, e
   FROM T1
   WHERE b = 'Gray'

3. SELECT a, b, c
   FROM T1
   WHERE e ='A'

4. UPDATE T1
   SET c = c - 100
   WHERE a = '001'
```

Node B:

```
1. SELECT a, b, c, d
   FROM T1
   WHERE c <= 500

2. SELECT T1.a, T1.b, T1.c, T2.f, T2.g, T2.h
   FROM T1, T2
   WHERE T1.c >= 1000
         AND T1.f = T2.f
         AND T2.h = 'F'

3. UPDATE T1
   SET d = 500
   WHERE d < 500
```

MLTIF determines time-invariant fragments in the following manner:

There are no two-way join queries at node A while query 2 of node B is a two-way join retrieval query. The query is first decomposed into two single-table queries for base tables $T1$ and $T2$ as follows:

```
Sub-query B.2.1: SELECT a, b, c, f
                 FROM T1
                 WHERE T1.c >= 1000
and
Sub-query B.2.2: SELECT f, g, h
                 FROM T2
                 WHERE T2.h = 'F'
```

The join condition is $T1.f = T2.f$.

Note that the join attribute f is included in the selecting attribute list of both Sub-queries so that the original query can be recreated.

Since both nodes have a fragment of base table $T1$, Sub-query B.2.1 and the join condition are submitted to node A. Sub-query B.2.2 will also be submitted to node A with the join condition because base table $T2$ is stored only at node A.

At Node A: The only condition clause in Sub-query B.2.1 contains a TSA, thus the condition is dropped. As no restricting condition remains in the sub-query, all the three tuples of join attribute, f, for $T1$ will be selected. The unique join-value set of attribute f is $\{S01, S02\}$, and it is transmitted to the relevant nodes participating in the join operation. In this example, the set is not sent to node B because no $T2$ data is stored at node B.

The condition clause of Sub-query B.2.2 received from node B does not contain a TSA and the evaluation of the query shows that tuples 1 and 4 of base table $T2$ at node A satisfy the query. Thus, the unique join-value set of attribute, f, of $T2$ becomes {S01, S04}. It is transmitted to node B where a fragment of base table $T1$ is stored.

Note that there is actually no need to send any unique join-value set to query node because no data will be replicated from itself.

At Node B: Similarly, sub-query B.2.1 is evaluated and a unique join-value set, {S03}, is generated and transmitted to A. Note that the restricting attribute, c, is a TIA at node B and would not be dropped from the sub-query as the case at node A.

At each node, the unique join-value sets are combined, join operation is evaluated, and the retrieval matrices are marked for the data to be replicated.

Thus the TIF created at node A for the base table $T1$ at node B is as follows:

	a	b	c	d	e
tuple 1	002	Dons	2000	filler	filler
tuple 2	005	Gray	800	600	B

Note that attributes d and e of tuple 1 are occupied by ‘fillers’ because the values of these two data items are not replicated in the local TIF. A filler is a predefined value for non-replicated cells in the TIF’s.

Node B: A TIF created at node B for the base table $T1$ at node A is as follows:

	a	b	c	d	e	f
tuple 1	001	Acme	filler	filler	filler	$S01$
tuple 2	003	Erso	filler	filler	filler	$S01$
tuple 3	004	Giant	filler	filler	filler	filler

TIF created at node B for the base table $T2$ at node A is as follows:

	f	g	h
tuple 1	$S01$	Cocki	F

C. Query Processing in TIF Environment

Query processing with TIF’s is different from that in a non-replicated environment. Local base tables and TIF’s may contain *some* of but not *all* of the data needed to answer a query. To satisfy the query request, it is important to accurately determine the portion of data that satisfies the query but resides at a remote node. Hence, each query is processed at the relevant base table nodes to determine the data items (not available in local base table or in TIF) to be transmitted to the query node. Processing of modification and retrieval queries is presented below.

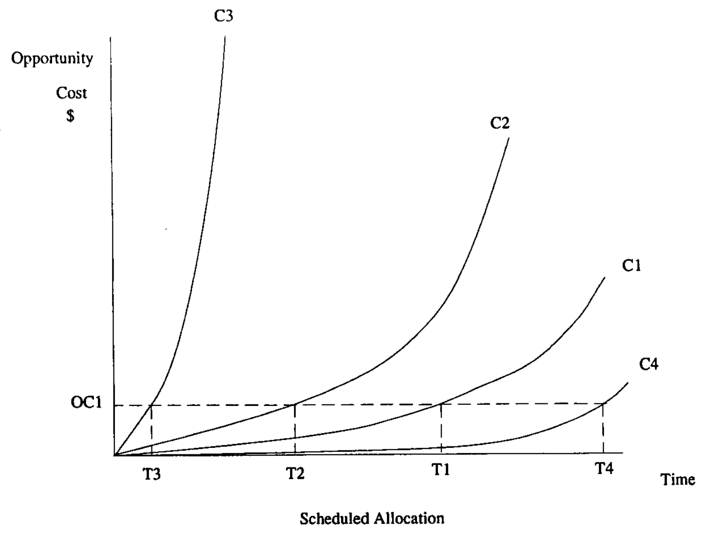


Fig. 4. Scheduling the allocation of TIF's.

Modification Queries: A modification query in a TIF environment is processed in the following manner:

Step 1. Consult data directory to determine the nodes where the relevant base tables are stored. Submit the query to those nodes.

Step 2. Update the relevant base tables. Examine TSA and Retrieval matrices for the base tables to explore whether the updated data items have been replicated at other nodes. Propagate the update to those TIF's that contain a copy of the updated data items.

Step 3. Replace the old TIF data items at the destination nodes.

Retrieval Queries: Single-table retrieval queries and two-way join retrieval queries require different treatments.

Single-Table Retrievals: The processing of a retrieval query is illustrated below:

Step 1. Consult data directory to determine the nodes where the relevant base tables are stored. Submit the retrieval query to those nodes.

Step 2. Process the query at the destination nodes. Examine the Retrieval matrices to determine if the data items exist in the TIF's at the query node. Transmit those data items have not been replicated to the query node.

Step 3. Merge data retrieved from local base table, local TIF, and remote base tables.

Two-Way Join Retrievals: Tne processing of a two-way retriaval query is illustrated below:

Step 1. Split the two-way join query into two single-table sub-queries. Consult data directory to determine the location of the relevant base tables. Send sub-queries to those nodes.

Step 2. Evaluate queries at destination nodes for the join attribute and transmit the unique join-value set to the nodes which receive the other sub-query.

Step 3. Process the sub-queries and check the respective Retrieval matrix (or matrices, if the node contains both base tables to be joined) as each node where the join values are available. Determine whether the data items that satisfy the join query have been replicated in the query node TIF's. Transmit the data items to the query node if they have not been replicated.

Step 4. Merge the data retreived from local base table, local TIF, and remote base tables.

D. Scheduling the Allocation and Update of TIF's

Allocation of TIF's to the user nodes may require intense data communication. Therefore, by sending TIF's selectively and/or during the non-peak hours can substantially reduce the communication costs. In situations where the update of critical data items is intense, the tolerance to the delay in update can be very low and cost of inaccuracy of information can be high. By contrast, in time-irrelevant applications where changes in data values are not critical to decision making, the tolerance to non-current data can be high, and the cost of inaccuracy of data can be low. The MLTIF algorithm is based on the trade-off between the opportunity cost due to inaccurate data and cost of maintaining the accuracy.

The opportunity cost of inaccurate data can be expected to increase over time as shown by the curve $C1$ in Fig. 4. The slope of the curve increases as time increases. Applications having low tolerance to non-current data will experience a cost curve similar to $C2$, where the slope of the curve increases very fast. In an extreme case, where the currency of data is extremely critical, the slope of the cost curve will be infinite as shown by the curve, $C3$. The applications that can tolerate non-current data will have their cost curves similar to $C4$, where the slope of the curve has a long flat lead time before deviating from the time axis. Therefore, for a given level of opportunity cost, OC1, MLTIF will schedule the creation and allocation of TIF's every $T1$, $T2$, $T3$, and $T4$ time periods for the above-mentioned classes of applications. In addition,

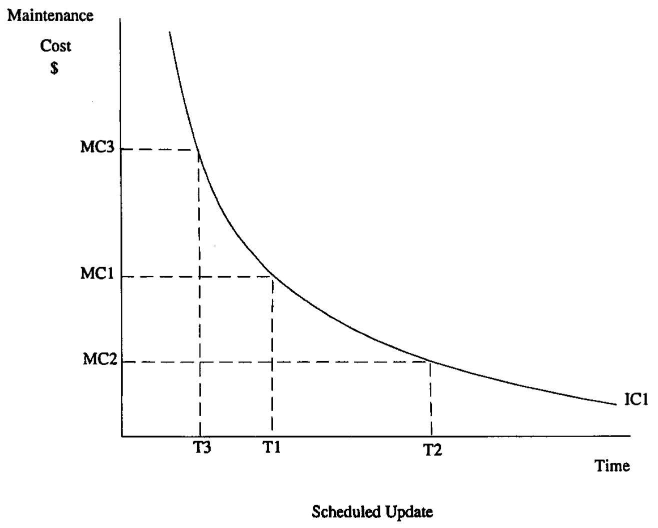


Fig. 5. Scheduling selective updates of TIF's.

MLTIF continuously monitors and adjusts these time periods to reflect the most current usage pattern.

For scheduling selective updates, it is important for MLTIF to know the age of TIF's. The age of a TIF is determined by the elapsed time between the creation of the TIF and the first update of any of its data item in the source base table. As shown in Fig. 5, to keep the cost of non-current data under a specified level, IC1, the age of a TIF data item has to be maintained within $T1$ units of time. The corresponding cost of currency maintenance is MC1. The currency maintenance cost of time invariant fragments climbs as the age limitations are stricter to reduce the cost of inaccuracy of the information. Therefore, a trade-off between the currency of data and the cost of maintaining it has to be made by MLTIF in the selection of the data items to be stored in the local TIF's.

MLTIF determines the most appropriate schedule for update of TIF's, also by aquiring knowledge from the query history. In making its update decision, MLTIF considers three important issues—'when' to update, 'where' the update source is, and 'how' the update should be done. Refreshing of TIF's can proceed immediately after the update of the base relation or be deferred until a query is made to the TIF. The deferred strategy may include periodic update, update on-demand regardless of queries, random, or a combination of the above methods [4]. The source (s) of the data to be used in the refresh of a TIF may be from its base relation or other view (s) recently created from this base relation.

E. Benefits of the TIF Approach

a) *Reducing database restructuring costs* User's requirements, needs, and the use of data is not constant in the current volatile business environment. A change in the data usage pattern may result in restructuring of a distributed database at a significant cost. The proposed strategy helps reduce the restructuring cost, because TIF is created from query patterns and automatically adjusts to changes in the environment.

b) *Reducing transmission costs* Overall transmission costs for query processing is reduced by storing TIF's locally. Also, the creation of TIF's is based upon query patterns and the corresponding data can be transmitted to its respective sites during economy or non-peak hours.

c) *Reducing update costs* Multiple copies of TIF do not create the problem of update synchronization because, by definition, TIF's do not change for a given time interval.

d) *Improving response time* Unlike other replication techniques, since TIF's do not change for a given time interval, a complete replication of TIF's is possible to provide more local data, and without unnecessarily creating update problems.

e) *Continuously improving performance* Since TIF is tied to the query history, as the size of query history increases, TIF tends towards optimality.

IV. Evaluation of the MLTIF Approach

To demonstrate the usefulness of the MLTIF approach and the conditions under which it may work best, we compare its performance with that of non-replication, full-replication, and materialized view approaches using simulation. The comparison is based on a given query history for a given time interval. Assuming data storage cost being negligible and unit data transmission costs between any two nodes being equal, the costs for creation of replicas, data retrievals, and modification for each approach are formulated. Detailed assumptions and the values assigned to the parameters can be found in the Appendix. The costs are averaged for 50 simulation runs for each setting involving different percentages of modification queries, sizes of network, and number of queries. Detailed findings are presented below.

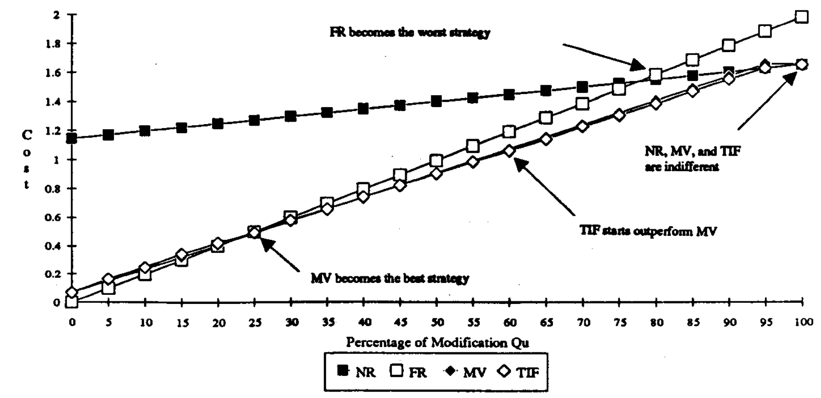


Fig. 6. Costs of the four strategies.(200 base queries, five nodes).

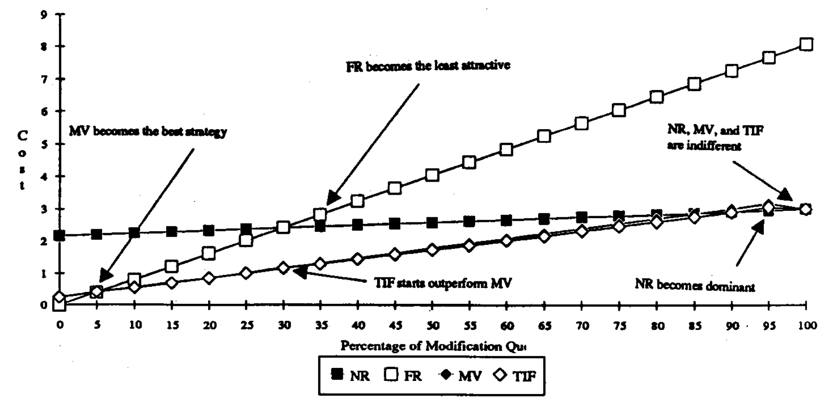


Fig. 7. Costs of the four strategies.(200 base queries, 10 nodes).

A. The Effect of Percentage of Modification Queries

Intuitively, the full-replication approach (FR) should perform best for low modification rates, and the non-replication approach (NR) for high modification rates. Between these two extremes, a partial data replication approach should be more desirable as long as the decrease in data retrieval costs offset the increase in the update synchronization costs. To evaluate TIF's performance relative to the other strategies, we simulate different scenarios varying modification query percentages from 0 to 100 at 5% level. Results of the simulation is summarized in Fig. 6. It is clear that FR has the lowest cost when modification queries in the network is below 25%. As the percentage of modification queries increases, FR becomes less attractive than the two partial data replication approaches as the benefit of full data locality is offset by the high update propagation costs. The materialized view approach (MV) becomes the most attractive alternative when the modification queries varies between 25% and 60%. TIF becomes the dominant approach when modification queries exceeds 60%. The benefit of TIF is that it excludes the time sensitive data from replication and, in the best case, as assumed in this analysis, there is no update propagation. However, TIF requires data transmission to answer queries accessing remote time sensitive data. MV, on the other hand, does not require any data transmission because it materializes these data into local replicas. As such, MV benefits from lower update propagation costs at low modification query percentages. As modification queries increase, MV is overwhelmed with materialized view updates and is surpassed by TIF. Note that when all the queries in the network are modification queries, the costs for NR, MV, and TIF are the same because there is no data replication for MV and TIF. Thus, the three cost lines in Fig. 6 merge when the percentage of modification queries is 100%.

B. The Effect of Network Size

When the size of the network increases, TIF becomes even more attractive. Fig. 7 shows the cost lines of the four strategies when the number of nodes is increased from 5 to 10 with everything else being the same. Here, MV dominates FR at 5% as against 25% modification queries in five-node network. This is because there are more nodes (nine rather than four) requiring the propagation. Since update propagation is more expensive than data retrievals, the benefit of data locality of FR is offset more quickly in a larger network. TIF becomes the preferred strategy at 30% modification queries as against 60% in the five-node network. As more materialized views may be required in a larger network and, consequently, higher cost is incurred in maintaining data currency. NR becomes the

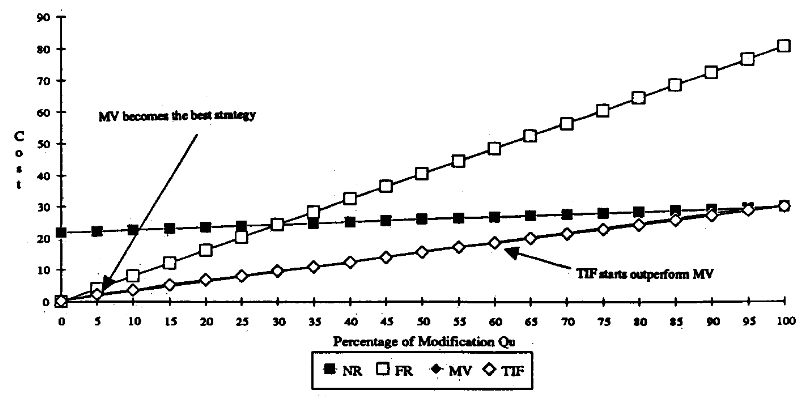


Fig. 8. Costs of the four strategies (2000 base queries, 10 nodes).

most effective strategy when more than 95% are modification queries. Saving in the retrievals of remote time invariant data does not offset the cost of TIF creation as it does in a smaller network.

C. The Effect of Number of Queries

In the above analysis the number of base queries in the network is set to 200. This parameter represents the total number of unique queries in the network. For a 10-node network with 2000 base queries the costs of the four strategies are plotted in Fig. 8. We observe that TIF would not become the preferred strategy until modification queries reaches 60%, which is higher than that in the 200 base queries network (30%). The increase in the number of retrieval queries requires higher retrieval costs of time sensitive data, and, consequently, delays the low cost lead of TIF. NR is never attractive unless all the queries are modification queries. The increases in relative magnitude of both retrieval and modification costs to TIF creation cost makes NR more difficult to surpass TIF. The above reasoning shows that:

1) FR is still the best strategy for small and/or non-modification-query networks,
2) MV has a larger dominance area,
3) TIF also has a larger dominance area and a flatter dominance line, and
4) NR has a much flatter dominance line and does not become attractive until at very high modification percentages and the network size exceeding 100 nodes.

V. Conclusion

This paper presents a learning based approach to the creation and allocation of time invariant fragments. For an application session, first, the time-invariant fragments at a node are defined through conceptual aggregation of expressions in the query history. Next, a set of queries to retrieve remote data for the creation of time invariant fragments are generated. Finally, these remote data retrieval queries are executed to build time invariant fragments and transmitted to the destination node.

To demonstrate the usefulness of the TIF approach and the conditions under which it may work best, we compare its performance with that of non-replication, full-replication, and materialized view approaches using simulation. The initial results are promising. It shows that TIF approach can be effective under the following conditions:

1) The percentage of TSA's is low, i.e., most of the data is time invariant;
2) The percentage of modification queries is high, and
3) The size of the network is large.

The current research can be extended in the following directions:

1) *Improving the precision of time-sensitivity definition.* The definition of time-sensitivity is done to attribute level. A more precise definition such as at the value range level could provide more data to time invariant fragments and improve the percentage of local processing.
2) *Refreshing TIF's.* TIF's are created independently for each application session. Time invariant fragments across sessions may overlap, i.e. part of data in the time invariant fragments created in one session is still valid and can be used in another session (s). The detection of these common time-invariant data will reduce data transmission cost.

Appendix

Notation, Formulation, and Values of Parameters for the Costs of the Four Data Allocation/Replication Approaches

Notation. The following notations are used in formulating the costs of the four approaches:

N Number of nodes in the network.

Q_n Total number of base queries in the network. These are unique queries inquired in the network. Additional retrieval queries are added when the network expands and the volume for each node is calculated through a distribution factor, as will be explained later.

$P_{\rm mq}$ Percentage of modification queries in the network.

$Q_{\rm mi}$ Number of modification queries at node i. $\sum_{i=1}^{N} Q_{mi} = Q_n^* P_{mq}$.

Q_{ri} Number of retrieval queries at node i.

Q_i Number of queries at node i. $Q_i = Q_{ri} + Q_{mi}$.

R_r Replication factor of network retrieval queries per additional node. When a node is broken into multiple nodes (e.g., West Coast is broken into Washington, Oregon, Northern California, and Southern California), a query in the original topology can be requested by one or more nodes in the new network because the nodes sharing the query result in the old network can now inquire themselves. The number of such additional queries is R_r percent of the network base retrieval queries for each additional node. Hence, the number of retrieval queries $= Q_N^*(1 - P_{mq}) * [1 + (N - 1)^* R_r], N > 1$. Since the number of retrieval queries increases as the network expands, the number of queries in the network also increases.

D_{mi} Distribution factor of network modification queries to node i. That is, $Q_{\mathrm{mi}} = Q_n * P_{\mathrm{mq}}/N^* D_{mi}$. Thus, $\sum_{i=1}^{N} Q_{\mathrm{mi}} = \sum_{i=1}^{N} Q_n{}^* P_{mq}/N^* D_{\mathrm{mi}} = Q_n{}^* P_{\mathrm{mq}}$ because the number of modification queries in the network is assumed to be a constant. Therefore, $\sum_{i=1}^{N} D_{\mathrm{mi}} = N$.

D_{ri} Distribution factor of network retrieval queries to node i. i.e., $Q_{\mathrm{ri}} = Q_N{}^*(1-P_{\mathrm{mq}})^*[1+(N-1)^* R_r]/N^* D_{ri}$, where $N > 1$, $\sum_{i=1}^{N} D_{ri} = N$.

P_{mi} Percentage of modification queries at node i. $P_{\mathrm{mi}} = Q_{\mathrm{mi}}/Q_i$.

A_{rri} Average percentage of retrieval query data is remote for node \$i\$ queries.

A_{mri} Average percentage of modification query data is remote for node i queries.

T_r Threshold network size for retrieval queries. When the network is small, it is easier to achieve high data locality through database partition. As the network size increases, the maintenance of high data locality becomes more difficult. When the expansion of the network exceeds a threshold size, the locality of data is sharply reduced to a very low level because the size of the unpartitioned base table is fixed and too much partitioning will result in meaningless fragments. The larger T_r is the larger the network can expand without a sharp reduce of data locality. To capture this nature of data locality, we propose that

$$A_{\mathrm{rri}} = 1)\ [(N-1)/N]^* 1.1^{(N-T_r)/2}, \quad \text{if } N < T_r$$
$$2)\ [(N-1)/N]^*[1 - e^{(T_r - N)/2}], \quad \text{if } N \geq 3T_r.$$

When $N = 1$, $A_{\mathrm{rr1}} = 0$.

T_m Threshold network size for modification queries. It is similar to T_r except it is for modification queries. Thus,

$$A_{\mathrm{rmi}} = 1)\ [(N-1)/N]^* 1.1^{(N-T_m)/2}, \text{ if } N < T_m$$
$$2)\ [(N-1)/N]^*[1 - e^{(T_m - N)/2}], \text{ if } N \geq T_m.$$

When $N = 1$, $A_{\mathrm{rm1}} = 0$.

P_{rtsai} Percentage of node i retrieval queries retrieve TSA's.

A_{rtsai} Average coverage of remote TSA's by the retrieval queries of node i.

M_m Cost multiplier for modification queries. It is more costly to modify a data item than to retrieve it because of additional operations and data transmissions. This overhead includes verification of the appropriateness of the modification, transmission of modified data back to the remote nodes from modification originating node, locking of the multiple copies if necessary, verification message of the modification from remote nodes, etc., are required for modification queries. Thus, $M_m > 1$.

R_{MV} Efficacy factor of a materialized view refresh method such as differential files, etc. It represents the ratio of entire-view-refresh cost incurred when a refresh method is applied. The lower the value, the more cost effective the refresh method is.

R_{FR} Efficacy factor of a full-replication update propagation method. The lower the value, the more cost effective the propagation method is.

E_{ri} Expansion factor from average data retrieved per query to overall coverage of data retrieved by queries of node i. A small value of E_{ri} indicates that the retrieval queries at node i access large amount of common remote data.

E_{mi} Expansion factor from remote modification data to overall modified data, both local and remote, by queries of node i. A small value of E_{mi} indicates that a large portion of data modified by the queries at node i are stored at local base tables.

P_{nmvm} Percentage of network modification queries modify data materialized in the views in the network.

P_{mvmi} Percentage of network modification queries modify data materialized in the view at node i. The number of modification queries is $Q_n{}^* P_{\mathrm{mq}}{}^* P_{\mathrm{mvmi}}$.

M_i Percentage of P_{nmvm} allocated to node i. Thus, $P_{\mathrm{mvmi}} = P_{\mathrm{nmvm}}{}^* M_i$.

C_{NR} Total costs of non-replication strategy.

C_{FR} Total costs of full-replication strategy.

C_{MV} Total costs of materialized view strategy.

C_{TIF} Total costs of TIF strategy.

Major assumptions To simplify the complexity of the problem, we assume the following:

1) Data storage/maintenance cost is negligible.
2) Unit data transmission costs between any two nodes are equal.
3) Database directory is replicated at all the nodes, i.e., the *possible* location of data can be identified locally.
4) Messages for requesting remote data in retrieval queries are included in the cost of the transmissions of the requested data. The messages for modifying remote data are also included in the remote modification cost. Typical messages include requesting for locks at remote nodes, granting locks from remote nodes, and requesting for releasing locks at remote nodes. To capture the relative higher cost of modification to retrieval queries for the same data, we include a cost multiplier for modification queries. Its values are assumed to be 1.5 in this analysis. The value means a modification query is 50% more costly than a retrieval query to access the same amount of remote data. The effect of the communication cost of these messages in the design of distributed systems has

been studied.[20]

5) Additional costs for unsuccessful retrieval and modification queries are assumed to be zero.
6) The amount of data transmitted in refreshing a materialized view is lower than that of creating the entire view. Whenever a discrepancy between the data in the view and that in the base table occurs (because of a modification query), only a portion of the view is refreshed. Various materialized view refresh techniques have been proposed and the effectiveness of these techniques depends upon environments under which they are utilized. To capture the performance of different materialized view refresh techniques, we include an efficacy factor in our model. In the current analysis, we assume a value of 10%, which indicates that on average only 10% of the view is refreshed to maintain the desired level of currency of data, although this may depend upon the refresh strategy selected.
7) The amount of data transmission in maintaining database currency for full-replication strategy is not proportional to the number of nodes. As for materialized views, more effective update propagation techniques could be utilized to reduce currency maintenance cost. To capture this phenomenon, we include an efficacy factor in our model similar to that for materialized views. We assume a 10% value for the factor in the current analysis. The value indicates that on average only 10% of the modified data are transmitted to other nodes to maintain database currency, although this may vary from system to system.
8) The value of a parameter is the same for all the nodes. Heterogeneous network nodes are expected in practice. But to keep the analysis from being too complicated, we assume the same value for a parameter across the network.

Cost formulation. The costs of the four strategies are formulated as follows:

Non-Replication

$$C_{\mathrm{NR}} = \sum_{i=1}^{N} [A_{\mathrm{rri}} \bullet Q_i \bullet (1 - P_{\mathrm{mi}})] \qquad \text{(Retrieval Cost)}$$
$$+ M_m \bullet \sum_{i=1}^{N} [A_{\mathrm{mri}} \bullet Q_i \bullet P_{\mathrm{mi}}] \qquad \text{(Modification Cost)}$$

Full-Replication

$$C_{\mathrm{FR}} = M_m \bullet (N-1) \bullet R_{\mathrm{FR}} \bullet \sum_{i=1}^{N} [E_{\mathrm{mi}} \bullet A_{\mathrm{mri}} \bullet Q_i \bullet P_{\mathrm{mi}}] \qquad \text{(Modification Cost)}$$

Materialized Views

$$C_{\mathrm{MV}} = \sum_{i=1}^{N} [E_{\mathrm{ri}} \bullet A_{\mathrm{rri}}] \qquad \text{(Creation Cost)}$$
$$+ M_m \bullet \sum_{i=1}^{N} [A_{\mathrm{mri}} \bullet Q_i \bullet P_{\mathrm{mi}}] \quad \text{(Modification Cost)}$$
$$+ R_{\mathrm{MV}} \bullet M_m \bullet [E_{\mathrm{ri}} \bullet A_{\mathrm{rri}} \bullet Q_n \bullet P_{\mathrm{mq}} \bullet P_{\mathrm{mvmi}}] \qquad \text{(Refresh Cost)}$$

Time-Invariant Fragmentation

$$C_{\mathrm{TIF}} = \sum_{i=1}^{N} [E_{\mathrm{ri}} \bullet A_{\mathrm{rri}} \bullet (1 - A_{\mathrm{rtsai}})] \qquad \text{(Creation Cost)}$$
$$+ \sum_{i=1}^{N} [A_{\mathrm{rri}} \bullet A_{\mathrm{rtsai}} \bullet Q_i \bullet (1 - P_{\mathrm{mi}}) \bullet P_{\mathrm{rtsai}}] \qquad \text{(TSA's Retrieval Cost)}$$
$$+ M_m \bullet \sum_{i=1}^{N} [A_{\mathrm{mri}} \bullet Q_i \bullet P_{\mathrm{mi}}] \text{(Modification Cost)}$$

Parameter Values. The parameter values used in the current study are based on realistic business scenarios. They are as follows:

A_{rtsai}	random number between 0% and 100%, with 80% probability of being below 20%, 15% probability of being between 20% and 50%, and 5% probability of being above 50%
D_{mi}	1
D_{ri}	1
E_{ri}	2.5
E_{mi}	3
M_i	10%
M_m	1.5%
P_{nmvm}	10%
P_{rtsai}	random number between 0% and 100%, with 80% probability of being below 20%, 15% probability of being between 20% and 50% and 5%$ probability of being above 50%
R_r	1%
R_{MV}	10%
R_{FR}	10%
T_r	50
T_m	50

References

[1] M. E. Adiba and B. G. Lindsay, "Database snapshots," in *Proc. Int. Conf. VLDB*, Montreal, PQ, Canada, 1980, pp. 86–91.

[2] R. Alonso, D. Barbara, H. Garcia-Molina, and S. Abad, "Quasi-copies: Efficient data sharing for information retrieval systems," in *Lecture Notes in Computer Science*, vol. 303, J. W. Schmidt, S. Ceri, and M. Missikoff, ed. Springer-Verlag, 1988, pp. 443–468.

[3] P. M. G. Apers, "Data allocation in distributed database systems," *ACM TODS*, vol. 13, no. 3, pp. 263–304, 1988.

[4] J. Blakeley, P. Larson, and F. Tompa, "Efficiently updating materialized views," *ACM SIGMOD 86*, pp. 61–71, 1986.

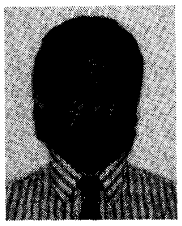
[5] R. G. Casey, "Allocation of copies of a file in an information network," in *Proc. Spring Joint Computer Conf., AFIPS*, 1972.

[6] S. Ceri, S. Navathe, and G. Wiederhold, "Distribution design of logical database schemas," *IEEE Trans. Software Eng.*, pp. 487–504, 1983.

[7] S. Ceri, B. Pernici, and G. Wiederhold, "Optimization problems and solution methods in the design of data distribution," *Information Systems*, vol. 14, no. 3, pp. 261–272, 1989.

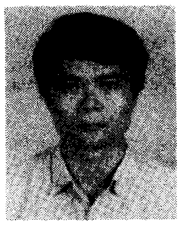
[8] A. Chaturvedi, G. Hutchinson, and D. Nazareth, "A synergistic approach to manufacturing systems control using machine learning," *J. Intelligent Manufacturing*, vol. 3, pp 43–57, 1992.
[9] W. W. Chu, "Optimal file allocation in a multiple computer system," *IEEE Trans. Computers*, vol. C-18, 1969.
[10] L. W. Dowdy and D. V. Foster, "Comparative models of the file assignment problem," *ACM Computing Surveys*, vol. 14, no. 2, pp. 287–313, 1982.
[11] A. Dutta, "Modeling of multiple copy update costs for file allocation in distributed databases," *Int. J. Computer and Info. Sci.*, vol. 14, no. 1, pp. 29–34, 1985.
[12] E. R. Hanson, "A performance analysis of view materialization strategies," in *Proc. ACM-SIGMOD Int. Conf. Manage. of Data*, 1987, pp. 440–453.
[13] B. Kahler and O. Risnes, "Extending logging for database snapshot refresh," in *Proc. Int. Conf. VLDB*, 1987, pp. 389–398.
[14] B. Lindsay, L. Haas, C. Mohan, H. Pirahesh, and P. Wilms, "A snapshot differential refresh algorithm," in *Proc. ACM-SIGMOD Int. Conf. Manage.of Data*, 1986, pp. 53–60.
[15] S. Navathe, S. Ceri, G. Wiederhold, and J. Dou, "Vertical partitioning algorithms for database design," *ACM TODS*, vol. 9, no. 4, pp. 680–710, 1984.
[16] S. Navathe and M. Ra, "Vertical partitioning for database design: A graphical algorithm," in *Proc. ACM-SIGMOD Int. Conf. Manage. of Data*, 1989, pp. 440–450.
[17] T. R. Rakes, L. S. Franz, and A. Sen, "A heuristic approximation for reducing problem size in network file allocation models," *Comput. &Ops. Res.*, vol. 11, no. 4, pp. 387–395, 1984.
[18] D. Sacca and G. Wiederhold, "Database partitioning in a cluster of processors," *Proc. Int. Conf. VLDB*, 1983, pp. 242–247.
[19] A. Segev and W. Fang, "Optimal update policies for distributed materialized views," *Manage. Sci.*, accepted for publication.
[20] A. Segev and J. Park, "Updating distributed materialized views," *IEEE Trans. Knowledge and Data Eng.*, vol. 1, pp. 173–84, 1989.
[21] D. G. Shin and K. B. Irani, "Partitioning a relational database horizontally using a knowledge-based approach," *ACM SIGMOD Record*, vol. 14, no. 4, pp. 95–105, 1985.

Alok R. Chaturvedi, for a photograph and a biography, please see page **193** of this issue of this TRANSACTIONS.



Ashok K. Choubey received the M. S. degree in computer engineering from Syracuse University in 1987.

He is a Senior Analyst with Information Systems—Data Management Applications Group, Bell Atlantic. He has extensive experience in distributed systems design, client/server applications, and database administration. His current research int erests include enterprise modeling and cooperative processing.



Jinsheng Roan received the M. S. degree from the University of Missouri-St. Louis, and the Ph. D. degree from Purdue University, West Lafayette, IN, both in management information systems. Currently, he is an Associate Professor of Business Administration at the national Chung Cheng University Taiwan. His prior experience includes information systems planning, development, and implementation. His research interests include distrib uted database, network design, machine learning, and information systems development.