Journal of the Korean
OR/ MS Society
Vol. 21, No. 1, April 1996

Refreshing Distributed Multiple Views and Replicas

Wookey Lee* · Jooseok Park** · Sukho Kang***

Abstract

In this paper we prescribe a replication server scheme with an algorithm DRF (Differential Refresh File) to refresh multiple materialized views and replicas in distributed environments. Before sending relevant tuples in server sites to client sites, an effective tuple reduction scheme is developed as a preprocessor to reduce the transmission cost. Because it utilizes differential files without touching base relations, the DRF scheme can help to minimize the number of locks, which enhances the system's performance.
Keywords: Differential files, Materialized views, Master files, Screen tests, Semi-join.

I. Introduction

One of the famous dilemmas in distributed data base systems is to guarantee data availability as well as their consistency. For availability's sake, data can be replicated in every local site where needed. Data replication is necessary but burdens the system, since mutual consistencies of those replicated data must be maintained. The schemes that uphold mutual consistencies are generally addressed as follows: pessimistic locking, optimistic schemes, time stamping procedures, and so on [2, 6]. The two-phase locking is the most widely employed protocol in the distributed environments, and there are so many mechanisms suggested in terms of 2PL; including site locking [1], cycle detection [17], site graph and global 2PL

* Dept. of Computer Science, SungKyul Univ. AnYang, Korea. wookey@ara.snu.ac.kr
** Business & Administration, KyungHee Univ. Seoul, Korea 130-701
*** Industrial Engineering, Seoul National Univ. Seoul, Korea 151-742

[4], and quasi-serialization [7], etc. The locking-based approach is certain but is liable to be slow, since it results heavy transaction activities and can nearly bring the network down by the sheer volume of messages sent among many dispersed replicas[19].

Materialized views are known to be a cost efficient alternative of data replications. Virtual views do not exist physically, but materialized views are stored as a separate table. Itis useful when users' application may approve non-current or 'near real time' data, or need frequent accesses with which the replication server can manage materialized views and various replicas in distributed sites.

There are three kinds of strategies to make the materialized views up-to-date; immediate updates [3, 5, 16], deferred updates [10], and periodic updates [10, 13, 15, 18]. The trade-off between the currency of materialization and their costs are associated in choosing a strategy. [10] and [14] addressed the timings of update quantitatively with a centralized DBMS and distributed one respectively. Periodic updates can include immediate updates by setting the intervals with no time lag to accommodate the refresh processes [8].

The simplest way to update the view is to re-execute the view definition, but it causes unnecessary locks of those tables and inadmissible communication costs. Here, we adopt the differential update method that does not reflect the whole base tables but the changed portion only by using the log as a differential file, which relieves the difficulty of the concurrency control [9, 26]. Most studies of the differential update have not considered their distributed environments. If ever, they are restricted to selection view (S-View) or selection-projection view (SP-View). They do not support the materialized views or replica with differential updates made by join or union operation (J-View) [8, 12, 13, 15]. Join operation is one of the most time-consuming and data-intensive operations in relational query processing. It is important that joins be performed efficiently because they are executed frequently and they include Selection-Projection operations. This study, therefore, deals with the structure of a replication server to refresh differential and join materialized views and to support various kinds of replicas (it also can embrace various fragments like vertical, horizontal, and mixed fragments as well as peer copies, and possible to extracts and versions) in their distributed environments.

Ⅱ. An Architecture of Replication Server

1. The basic concept

A relational schema $\mathbb{R}$ is a set of database relations and a relation R is an instance over $\mathbb{R}$. Let R(TID^b, VTID, A_u, TS^b) be a base relation[1] located at site S_b where A_u are data attributes, and TID^b is a physical identifier of the tuple and the VTID is employed in many cases as the primary key of the base relation. The TID^b and VTID are basically labeled by DBMS. TS^b is the time-stamp that the base relation was lastly changed by the committed transactions.

Let R_j^c for j $\in \Omega$ be fully synchronized copies (replica) of R $\in \mathbb{R}$ and Ω is a set of sites denoted as an integer. (For the sake of convenience, the site identifiers are denoted two-folds such that both j$\in\Omega$ and S_j mean site j.) Views V1, ……, Vn are materialized at remote sites, and some of them are selection views (S view), some are selection/projection views (SP view) and others are join views (SPJ views). The schema of view Vi :=(VID, S_i, EXP_i, A_u, LR_i, NR_i) where VID is view identifier; S_i is the site where the view is stored; EXP_i is the predicate of view definition; A_u is the set of attributes needed; LR_i is the last refresh time; and NR_i denotes the next refresh time.

Example : Two tables are suggested as follows: SUPPLIER(VTID^s, S#, SNAME, QTY, P#) is at site 1, PRODUCT(VTID^p, PNAME, COLOR, WT) at site 2. (Here the subscripts s and p denote each tables, and mean that they are not equal though denoted of the same notation.) A materialized view (V1) at site 3 is defined as followings:

```
CREATE MATERIALIZED VIEW V1 AS
SELECT SUPPLIER.SNAME, PRODUCT. PNAME, "Quantity=", QTY
FROM SUPPLIER, PRODUCT
WHERE SUPPLIER.P# = PRODUCT.P# AND QTY < 80 AND COLOR <>Y;
```

A differential file is used to refresh materialized views and replicas, and has the following schema: DF(VTID^d, A_u OP, TS^d, PTS). Where the VTID^d is the VTID of differential file DF and the superscript *d* means that it can be different from that of the base bable R. But here we assume that they are the same, ie, VTID^b=VTID^d=VTID. The OP indicates types of the operation to be done for each tuple: it will be one of three codes: 'ins', 'del', or 'del$_m$' and

1) The term 'base relation' and 'base table' will be used without discrimination.

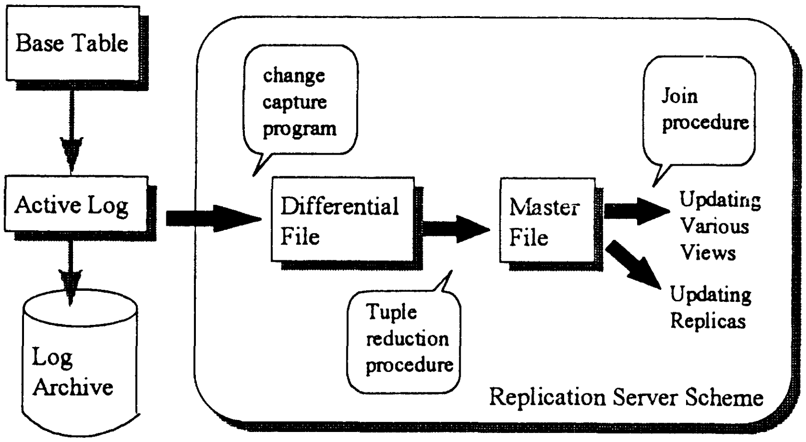

Figure 2-1 Replication Server Scheme

'ins_m' in series, where 'ins' means insertion, 'del' deletion, and a modification is depicted as 'del_m' and 'ins_m' in series with the same time stamp. TS^d=the time the differential tuple was appended(we assume TS^d is equal to TS^b). PTS is the previous value of TS^d, and it will be Null, if it is newly inserted.

[TABLE2-1] Example tables: SUPPLIER, PRODUCT, and a materialized view V1

SUPPLIER

$VTID^s$	S#	SNAME	QTY	P#
1	S1	JAMES	60	P1
2	S2	MARGOT	70	P3
3	S3	JUN	20	P1
4	S4	SIMON	40	P2
5	S5	MICHAEL	40	P6

PRODUCT

$VTID^p$	P#	PNAME	COLOR	WT
101	P1	PIN	G	200
102	P2	WASHER	V	600
103	P3	BOLT	R	300
104	P4	NUT	G	700
105	P5	PIN	Y	300
106	P6	WASHER	B	200

View V1 (Current Time 1:00)

SNAME	PNAME	GQY
JAMES	PIN	Quantity=60
MARGOT	BOLT	Quantity=70
SIMON	WASHER	Quantity=50
MICHAEL	WASHER	Quantity=40

An example of the differential file is suggested in [TABLE 2-2]. The changed (by committed transactions) tuple and its operation codes(OP's) are recorded with time stamp in series. (For explanations sake, a record number is appended virtually, they are depicted at the right hand side of the differential files.) For example, record number 1 and 2 mean that the QTY of tuple S2 is modified from 50 to 70 at time 2:15 and record 5 means that a tuple {S4, JIM, 60, P3} is newly inserted at time 3:00, etc.

The differential update scheme basically reduces communication costs greatly by sending differential files to the relevant sites instead of sending huge base tables. Here we want to reduce the contents of the differential files much more through the tuple reduction procedure described below. The reduction procedure and multiple query optimization technique are addressed in [3, 11, 13, 19, 22].

The tuples that have passed the reduction process are pipelined to a procedure that appends a file, called Master File, having the following schema: MF(VTID, A_u, OP^v, {Site, VID}), where OP^v indicates types of update to be done for each view or replica in the list {Site, VID}. The superscript v means that the operation codes of a differential file are integrated so that they may be different from those of the differential file itself. A_u is the relevant attributes: it will be Null when OP^v is 'del' (since the remote view needs only a VITD for a deletion): the inserted data item of A_u will be denoted as OP^v is 'ins', and the modified data item will be 'mod'. In case of modification we will assume that $A_u = \{A_i = Value_i\}$ where A_i is the name of the modified attribute and $Value_i$ is its new value.

The Replication Server Scheme covers all the procedure that captures the changed data from the active log and creates a differential file, and compresses the tuples through the reduction procedure, and finally serves the update needs of the client sites. See [Figure 2-1].

[TABLE 2-2] Example Differential Files

(a)Differential File of SUPPLIER

VTID^s	S#	SNAME	QTY	P#	OP	TS	PTS	(record number)
2	S2	MARGOT	50	P3	delm	2:15	1:30	1
2	S2	MARGOT	70	P3	insm	2:15	1:30	2
4	S4	SIMON	40	P2	delm	2:20	2:00	3
4	S4	SIMON	50	P2	insm	2:20	2:00	4
6	S6	JIM	60	P3	ins	3:00	Null	5
2	S2	MARGOT	70	P3	del	4:45	2:15	6
4	S4	SIMON	50	P2	delm	5:08	2:20	7
4	S4	SIMON	60	P2	insm	5:08	2:20	8
6	S6	JIM	60	P3	delm	6:43	3:00	9
6	S6	EUGENE	60	P3	insm	6:43	3:00	10
7	S7	LEE	40	P3	ins	7:00	Null	11
8	S2	MARGOT	80	P3	ins	7:12	Null	12
6	S6	EUGENE	60	P3	del	8:40	6:43	13

(b)Differential File of PRODUCT

VTID^p	P#	PNAME	COLOR	WT	OP	TS	PTS
105	P5	PIN	Y	300	del	3:15	1:00
103	P3	BOLT	R	300	delm	6:10	0:30
103	p3	BOLT	R	500	insm	6:10	0:30

Ⅲ. The Reduction Procedure

1. The Duplicate Elimination Procedure and the screen test

Several cases of standard screen tests were suggested in [3, 12, 20]. Blakeley et. al [3] considered that every tuple that is changed by the committed transactions should be tested to be irrelevant or not through the view definitions so that it may take time very much.

Before sending all the tuples to the relevant remote sites, we reduce them in a (tuple) reduction procedure by the following 3 steps: a duplicate elimination process, a screen test, and a post-screening elimination. (Replicas or the peer copies of the base table that should be updated immediately, of course, need not this reduction procedure.)

At first, the duplicate elimination procedure exterminates all tuples with the same VTID value in DF except only the first and/or the last. We need next definitions: A subsets of views, $SV=\{SV_1, \cdots\cdots, SV_k\}$, for $k \leq n$ are refreshed at time t_r. We divide the set SV into mutually exclusive disjoint subsets $SV_1, \cdots\cdots, SV_m$, such that $\cup_j SV_j=SV$ and all views of SV_j have the same last refresh time denoted by TR_j, that is $LR_i = TR_j$, for all $V_i \in SV_j$. The set $SV_1, \cdots\cdots, SV_m$ is ordered such that $TR_1 < TR_2 < \cdots\cdots < TR_m$ and they are grouped by the refresh time TR_j. The set SV_j is expected that at least of one cycle of refresh time units, TU_j, are passed with no changes in the base table. If $TR_j < T_i^d[TS]$ where $\min(T_i^d[TS]) < TR_k < \max(T_i^d[TS])$ for $k \in \{1, 2, \cdots\cdots, n\}$, then set $LT_j=t_r$, $NT_j=t_r+TU_j$ and $SV:=SV-\{SV_j, \cdots\cdots, SV_m\}$ for $V_i \in SV$. This procedure eliminates all the tuples with the same VTID bacause each VTID is uniquely endowed by DBMS and is not re-assigned again. The types of OP are limited one of the following $7(=2^3-1)$ sequences: (1)ins: (2)del: (3)ins and del: (4){delm, insm}: (5){delm, insm} and del: (6)ins and {insm, delm}: (7)ins and {insm, delm} and del.

The procedure scans the Differential File backwards from the last tuple group-wisely delineated by TR_j. Tuples between TR_{k+1} and TR_{k+2} are apparently irrelevant to views $SV_{k+1}, \cdots\cdots, SV_m$ for these views had been already updated and thus the tuples in $TR_k \leq T_i^d[TS] < TR_{k+1}$ are selected as following 3 cases: whether the types of OP are (1)ins or insm, or (2)del, or (3)delm. For ins, it is the first instance of DF with the same VTID value, then the previous tuples are not considered any more.

OP:=insm means that the tuple will be possibly the last one, thus it will be selected without further scan of DF. In case of deletion, it is always associated with the last instance of the VTID value, then it is divided by the following 2 cases: no treatment, if $PTS \leq T_{min}$; select the tuple, if $PTS > TR_1$. If the type of OP is delm, then the next 3 cases are possible: If $PTS \leq TR_1$, no treatment: if $PTS > TR_k$, it implies that this tuple does not have the first VTID value according to the views $SV_1, \cdots\cdots, SV_k$ and simply proceed with the scan: otherwise, if $TR_j < PTS \leq TR_{j+1} \leq TR_k$, this implies that the tuple has the first instance of the VTID value in DF according to views $SV_{j+1}, \cdots\cdots, SV_k$.

Example : DF tuples can be considered between time 2:15 and 8:40 and searched backwards. At first, {S6, Eugene, 60, P3, del, 18, 9} has the OP:=del and PTS:=6:43, then record number=5 is selected but the number 9 and 10 are excluded, for they are duplicated with the same VTID: number 12, 11 and 8 are included since the OP's are 'ins', then the tuples numbered 3, 4, and 7 are excluded: the tuple numbered 6 has PTS:=2:15, thus record 1 and 2 are

[TABLE 3−1] Differential File of SUPPLIER after the Duplicate Elimination Procedure

VTID'	S#	SNAME	QTY	P#	OP	TS	PTS	(record number)
6	S6	JIM	60	P3	ins	3:00	Null	5
2	S2	MARGOT	70	P3	del	4:45	2:15	6
4	S4	SIMON	60	P2	insm	5:08	2:20	8
7	S7	LEE	40	P3	ins	7:00	Null	11
8	S2	MARGOT	80	P3	ins	7:12	Null	12
6	S6	EUGENE	60	P3	del	8:40	6:43	13

irrelevant. The example DF and its results are in [TABLE 3-1].

2. The screen Test

The tuples that passed the duplicate elimination process are hanged by the screen test that is the second process of the reduction procedure to exclude tuples to the view definition(here we construct a screen tree). Screen Tests are addressed in [3, 10, 15]. The predicates are evaluated to True or False.

Example: In the example differential file record number 12 is current but turned out to be false and deleted, for it is out of the range in the view definition (QTY 〈 80).

3. The Post Screening Elimination

After the screen test, the remaining tuples are sorted by the primary keys not by the VTID (say, S#, for example), and they are requested to implement the third procedure, named the Post Screening Elimination (See [Table 3-2]). In the procedure, some tuples are ignored (if the OP:=ins and del in series), and some are unified (if the OP's are delm and insm respectiverly, then they are unified as 'mod' that menas modification). In case of del and ins, they have different VTID values each other, but in reality they are the same tuple to be modified.

Example: record number 6 and 11 are unified as one tuple described 'mod', and record unumber 5 and 13 are ignored by the Post-Screening Elimination rules see [TABLE 3-2]; the results of the Post-Screening Elimination are suggested with the new record numbers in [TABLE 3-3].

[TABLE 3-2] Post Screening Elimination Rules

Output of Screen Test	Op by Post Screening
ins, del	ignore
insm, del	ignore
ins, delm	ignore
insm, delm	ignore
ins	ins
insm	ins
ins, insm*	ignore
del	del
delm	del
delm, del	del
del, ins	mod
delm, insm	mod
del, insm	mod
delm, ins	mod

[TABLE 3−3] The final Differential File of SUPPLIER after the Post-Screening Elimination

$VTID^s$	S#	SNAME	QTY	P#	OP	TS	PTS	(record number)
2	S2	MARGOT	70	P3	del	4:45	2:15	1′
4	S4	SIMON	60	P2	mod	5:08	2:20	2′
7	S7	LEE	40	P3	ins	7:00	NULL	3′

Ⅳ. Updating Join Materialized Views

1. Immediate Updates

Immediate updates to the peer copy and some fragments (vertical or horizontal) can simply be supported in the scheme of section Ⅱ. There may by several methods to immediate updates. When a transaction is committed, then it can invoke remote update to the replicas and fragments by triggering or any stored procedures [5]. Another method is depicted in [3]. In our scheme an alternative can be suggested by setting NR_i := the commit time of transaction for replicas that want immediate update. In such an immediate update, it is fundamentally a matter of trade-off between the currency of the data and the system performance: the more local sites are concerned to the replicas (including join operation), the worse update efficiency of a replica, we can not, of course, use of the benefits of the reduction procedure described in section Ⅱ.

2. Updating join materialized views and replicas

After reducing the differential file, one of the important problems in the replication server is how to reflect the changes of base tables to the views. Before sending the reduced tuples, we can determine whether these tuples need to be referred to remote sites or not. The Selection views (S-View) and Selection-Projection views (SP-View) need not to be referred to remote sites, and possibly sent to view sites directly. In Join (J-View), however, tuples are to be sent to the pertinent sites and should be joined with the local data and then re-sent to the view or replica sites.

In join operation, we consider two kinds of tuple changes: 'ins' and 'mod'. Because the deleted tuples of DF are not to be joined, thus they are sent directly to the view sites where the pertinent views are stored. If the type of the OP is 'ins', then they are sent to the join site anew; they are to be transmitted to the related sites that the tables participated in join are located. After being joined with the table, these tuples are appended to the relevant materialized views. When the attribute used in join predicate is changed (in this case the OP is 'mod'), such tuples that contain these must be manipulated in the similar way, and at last those tuples are updated to the views.

The relevant tuples to be joined will be collected at the site where the join is to be performed. If the join operation is carried out by the tuples from several sites, then it is difficult to manage these tuples as one table, since the sizes of these tuples may be different with each other. Thus we prescribe a new architecture called DRF (Differential Refresh File) method. The schema of DRF is as follows: DRF(Site-ID, VTID, Au) where Site-ID is a unique identifier of the site where the differential file comes from and Au is the attributes that are used in join predicate, and it has internal pointers to connect the attributes of differential tuple in the DRF.

When we make a join with DRF where the relevant relation is located. Without loss of generality, we here set R_i be in site S_i and R_i in S_j, and the materialized view V3 in S_k. We also assume here that A_r^i of R_i be a foreign key is related to the primary key, A_1^j, of R_j(A_r^i be called the r-th attribute if table i, but here we set merely A_r). DRF_j cannot be created in the site other than S_j. If tuples are changed (deleted, modified, and inserted) in R_j, then the JDF_j

need not be effected any how. Because it will not trigger any new relationship with the tuples of R_i. Even though there is a new insertion in both two tables simultaneously, join can be performed merely by using DRF_j only.

Once DRF_j are sent to the site S_j, there are two strategies: if all the tuples sent from site S_i is matched with those of DF_{Rj}, then there is no need to search all base table of R_j, reducing the processing time needed to join. If there exist at least one tuple of DFR_i that does not match with DF_{Rj}, then all the table of R_j cannot help being searched.

[TABLE 4-1] The Final View(V1) at time 9:00

View V1 (Current Time 9:00)

SNAME	PNAME	QTY
JAMES	PIN	Quantity=60
MICHAEL	WASHER	Quantity=40
SIMON	WASHER	Quantity=60
LEE	BOLT	Quantity=40

Example: The final treatments by the DRF algorithm to the reduced tuples are as follows; record 1' (in [TABLE 3-3]) be sent directly to the view site and deleted; record 2' and 3' are sent to site 2 to be referenced; then at first searching the Differential File of PRODUCT and Joined(=new record number 3'); but there still remains no matched tuple(=new record number 2'), then the base table of PRODUCT can not but be searched and to join the relevant tuples. The final materialized view at current time(TR_j):= 9:00 is in [TABLE 4-1].

3. JOIN Algorithm

We assume that table R_i is in site i and R_j in site j to be joined at site j for $i \neq j$. Here, for convenience's sake, we set A_f be a foreign key of table R_i at site i; it is relevant to a primary key of table R_j. If T is a tuple, $T[A_u]$ denotes attribute A_u of T and the superscripted tuples T^d and T^b mean the differential tuple and the base tuple respectively. For example, $T_i^d[A_u]$ indicates the attributes of differential tuple of table R_i and $T_i^d[TS]$ means its time-stamp.

DRF Algorithm

1) Get $T_i^d[A_u]$ where $TR_j < T_i^d[TS] \leq t_r$

/*Get the tuples that have the same refresh times*/

2) Do Duplicate elimination and Screen test and postscreening elimination.

3) Go to algorithm DRF−JOIN

⟨DRF−JOIN process⟩

Create DRF as for $\forall A_u$ Do:

DRF_i [VTID] ← T_i^d [VTID]

DRF_i [A_u] ← T_i^d [A_u]

DRF_i [TS] ← T_i^d [TS]

DRF_i [OP] ← T_i^d [OP]

(1)deletion

If DRF_i [OP]=: del

Send DRF_j directly to site S_{MF} /*No need to acess sitej*/

(2)insertion

If DRF_i [OP]=: ins and $\exists$ $T_i^d[A_f]$ = $T_j[A_1]$ /*select Join attributes to send*/

Else stop;

Send DRF to site j

If T_i^{DRF} [A_f] = : T_j^d[A1] /*join DRF with the differential file of R_j*/

then J1 ← $T_i^{DRF} \otimes T_j^d$ /*⊗ means join operator*/

Else J2 ← $T_i^{DRF} \otimes T_j^b$ /*join DRF with the base table of R_j*/

Send J1∪J2 to site S_{MF} /*the results are sent to Master File site*/

(3)modification

If T_i^{DRF} [OP]=:mod AND T_i^{DRF} [A_f]≠:T_j^d[A1]

then send DRF_j to site S_{MF} /*if A_f(foreign key)is not changed at i*/

Else do J3 ⟨--- $T_i^{DRF} \otimes T_j^b$

/*There always exists the tuple in R_j by the Referential Integrity Rule*/

Send J3 to site S_{MF}

/*the results are sent to Master File site*/

V. Performance Analysis

1. General Notations

Ω : the set of site index for $i \in \Omega=\{1, 2, \cdots\cdots, n\}$

B : Page size (bytes)

SF : Semi join factor

Si, SMi : The site where Ri is located and the materialized view MVi is located

$C_{I/O}$, C_{comm} : I/O cost (ms/block), Transmission rate (bits/s)

H_{B-Sj} : Height of B+tree at Sj site

n_{Ri} : Number of Ri tuples per page($=B/W_{Ri}$)

Pr_{DF-Rj} : Probability that all the tuples needed to join operation is in DF_{Rj}

f(N, P, K) : Expected number of pages fetched when accessing K out of N tuples in a file occupying P disk pages [22]

Ui : Number of tuples in DF_{Ri}

Ui^e : Number of tuples in the result of duplicate elimination procedure in DF_{Ri}.

Ui^s : Number of tuples that pass the screen test in DF_{Ri}

Ui^t : Number of tuples to be transmitted to the view site in DF_{Ri}

U^{Ri}, U^{DRFj}, U^{j-r} : Number of tuples in R_i, DRF_j, and that are not joined with DF_{Rj} in DRF_j respectively

α_e, α_s, α_p : duplicate elimination factor, screen factor for view predicate, and postscreening elimination factor respectively.

Wins, Wdel, Wmod : Width of MF tuples with OP = ins, del and mod respectively

W_{Ri}, W_{DRFj}, W_{mvi}, W_B : width of R_i tuple, DRF_j tuples, materialzed view V_i and B+tree record respectively.

2. Cost functions

If there is no algorithms to manipulate differential files such as DRF, we cannot help but utilize base table methods to refresh views and replicas. Here we compared algorithm DRF to the Semi-join algorithm (among various join schemes, semi-join was addressed to be preferable for the distributed environments in [4, 6]). We expect that it is sufficient that the single update of DRF file compared to the Semi-join, since multiple views naturally will show much better performances. In comparison, we consider communication costs, I/O costs. (File holding costs and computing costs are neglected, because it is so small that they can not be computed. DB relations are stored in the costless Disks, and the portion of computing times at the main memory is below 1% in the total cost.) But the I/O costs to the Disks are considerable. We assumed here that R_i and R_j have clustered indexes on the attributes to be joined.

(1)Algorithm DRF

In order to establish the cost functions, we first determine the number of tuples that pass through each stage of the reduction procedures.

$Ui = U_{i.ins} + U_{i.del} + U_{i.delm} + U_{i.insm}$ (where $U_{i.delm} = U_{i.insm}$)

$Ui^e = U^e_{i.ins} + U^e_{i.del} + U^e_{i.delm} + U^e_{i.insm} = U_{i.ins} + U_{i.del} + \alpha_e(U_{i.delm} + U_{i.insm})$

$Ui^s = U^s_{i.ins} + U^s_{i.del} + U^s_{i.delm} + U^s_{i.insm} = \alpha_s(U^e_{i.ins} + U^e_{i.del}) + \alpha_s U^e_{i.delm} + \alpha_s U^e_{i.insm}$

$Ui^t = U^t_{i.ins} + U^t_{i.del} + U^t_{i.mod} = \alpha_p U^s_{i.ins} + \alpha_p U^s_{i.del} + \alpha_p(U^s_{i.insm} + U^s_{i.delm})$

The total cost in algorithm DRF can be divided by the site Si, Sj and Sm

Cost in Si=CIO0+CIO1+CIO2+CCOM1

CIO0=Cost of reading Ui tuples from the log$=C_{I/O}(U_{i.ins}+U_{i.del}+U_{i.mod})W_R/B$

CIO1=Cost of reading Ui tuples from $DF_{Ri}=C_{I/O}(U_{i.ins}+U_{i.del}+U_{i.mod})W_R/B$

CIO2=Cost of sorting U^s_i tuples $=C_{I/O}2*U^s_iW_R/B$

CCOM1=Cost of transmitting DRF tuples to S_j and $SM_i=8(U^t_{i.ins}W_{ins}+U^t_{i.del}W_{del}+U^t_{i.mod}W_{mod})/C_{comm}$

Cost in SMi=CIO3+CIO4

CIO3=Cost of accessing the B+tree at the view site$=C_{I/O}[(H_{B-SMi}-1)+f(\alpha_sN_r, \alpha_sN_rW_R/B, U^t)]$

CIO4=Cost of updating the data in the view table$=C_{I/O}2*f(\alpha_sN_r, \alpha_sN_rW_R/B, U^t)$

Cost in S_j=CIO5+CIO6+CIO7+CIO8+CIO9+CCOM2

CIO5=Cost of reading U_j tuples in $DF_{Rj}=C_{I/O}(U_{i.ins}+U_{i.del}+U_{i.mod})W_R/B$

CIO6=Cost of sorting U^s_j tuples $=C_{I/O}2*U^s_jW_R/B$

CIO7=Cost of reading JDF_j $=C_{I/O}*U_{DRFj}W_{JDFj}/B$

CIO8=Cost of sorting DRF_j by join attribute$=C_{I/O}2*U_{DRFj}W_{DRFj}/B$

CIO9=Cost of reading R_j tuples for join operation with U^{j-r}

$=Pr_{DF-Rj}*\{C_{I/O}[H_{B-sj}-1)+f(U^{Rj},U^{Rj}W_R/B, U^{j-r})\}$

CCOM2=Cost of sending joined tuple to SMi+Cost of sending the change of Relation Rj to SMi $=8*(U^{DRF_i}*W_{mvi}/C_{comm}+8(U^{t}_{idel}W_{del}+U^{t}_{imod}W_{mod})/C_{comm})$

Therefore, the total cost of DRF(TCD) is CIO0+CIO1+CIO2+CIO3+CIO4+CIO5+CIO6+CIO7+CIO8+CIO9+CCOM1+CCOM2

(2)Algorithm SEMI-JOIN

⟨Semi-join algorithm⟩

1) Send the attribute of R_i which is used in join predicate to site S_j where R_j is located. (It is assumed that the size of R_i is greater than that of R_j)

2) In S_j send the tuples of R_j, that are matched with the attributes of R_i sent from S_i, to S_j.

3) In S_i join R_i with the tuple sent from S_j and send them to the sites where materialized views are located.

(3) Cost functions

Cost in Si=BIO1+BCOM1+BIO2+BIO3+BCOM2

BIO1=Cost of reading join attribute index of Ri $=C_{I/O}[(H_{B-sj}-1)+U^{Ri}*W_R/B]$

BIO2=Cost of reading the tuples of Rj sent from Sj $=C_{I/O}*SF*U^{Rj}*W_R/B$

BIO3=Cost of reading Ri to join with the tuples of Rj $=C_{I/O}*[(H_{B-sj}-1)+U^{Ri}*W_R/B]$

BCOM1=Cost of sending the index to Sj $=8*Ccomm*U^{Ri}*W_R/B$

BCOM2=Cost of sending joined tuple to the sites where materialized views are located.

$=8*\alpha_s*U^{Rj}*$ Wmvi/Ccomm

Cost in Sj=BIO4+BIO5+BCOM3

BIO4=Cost of reading indexes of Ri sent from Si $=C_{I/O}*U^{Ri}W_R/B$

BIO5=Cost of reading Rj $=C_{I/O}[(H_{B-sj}-1)+f(U^{Rj}, U^{Rj}W_R/B, SF*U^{Rj})]$

BCOM3=Cost of sending the tuples that match the join attribute of Ri in Si $=8*SF*U^{Rj}*W_{Rj}/Ccomm$

Then the total cost of Semijoin (TCB) is BIO1+BIO2+BIO3+BIO4+BIO5+BCOM1+BCOM2+BCOM3.

3. Performance Analysis

The following values are assigned to the parameters for analysis. α_s is varied between 0.01 and 1.0. Let B=4000 bytes, $W_{Ri}=W_{Rj}=200$ bytes, WB=8 bytes, $C_{I/O}=25$ ms/block, $\alpha_e=0.6$, $\alpha_p=0.6$, Wins=200 bytes, Wdel=8 bytes, Wmod=100 bytes.

Assuming the above values and varying communication speed, we can get the total cost of each algorithm. The results are depicted in [figure 5-1] and [figure 5-2]. They show that the size of differential files is crucial because the total cost ratio is much more significant with small differential file (10k) in [Figure 5-1] rather than in [Figure 5-2] (50k). There was also a strong trend that the total costs became smaller and smaller as the communication speeds went up.

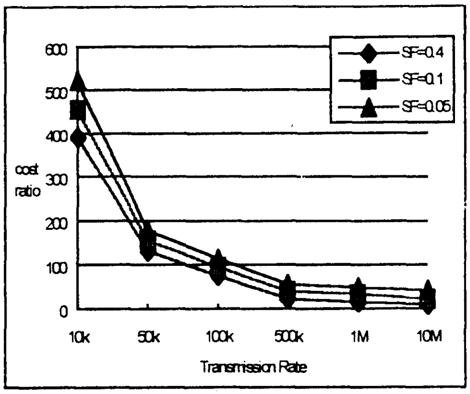

Figure 5－1 Total cost ratio Ⅰ(TCB/ TCD)

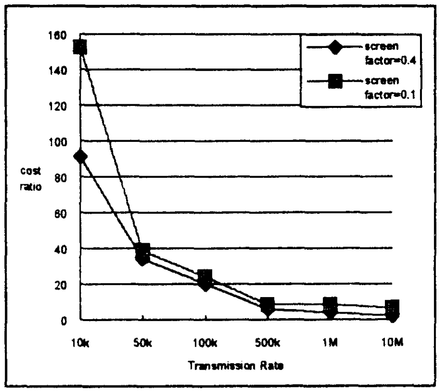

Figure 5－2 Total cost ratio Ⅱ(TCB/ TCD)

Then, it should be checked that what is the key components in cost changes as the communication speed goes up. In base table manipulation, transportation cost's portion is almost up to 70% of total cost, and naturally it goes down as the communication speed increases. Here, the cost to read each base table increased so much as the transportation decreased. In differential file manipulation, there is no major component. IO costs of differential files are big at each site, and still the transportation costs have some position but they are decreased as communication speed goes up. In both two cases, the other components are trivial, since the sums are under 10% at all the cases.

[figure 5-5] and [figure 5-6] show the communication cost ratio of the differential method and that of the semijoin one respectively. They show that algorithm DRF consumes a much smaller share of the communication cost than the semijoin, even if large differential files are

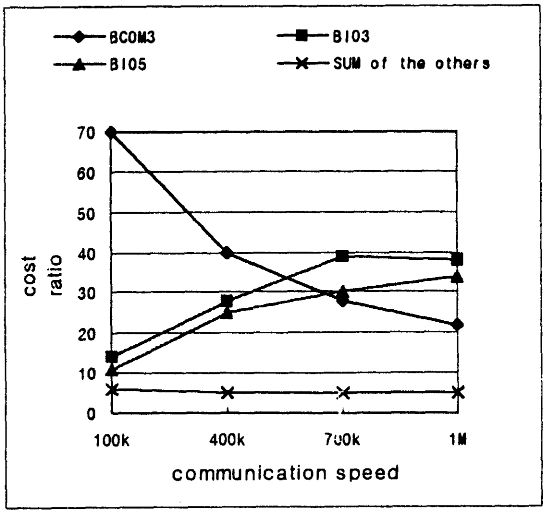

Figure 5-3 Cost Components in Semijoin

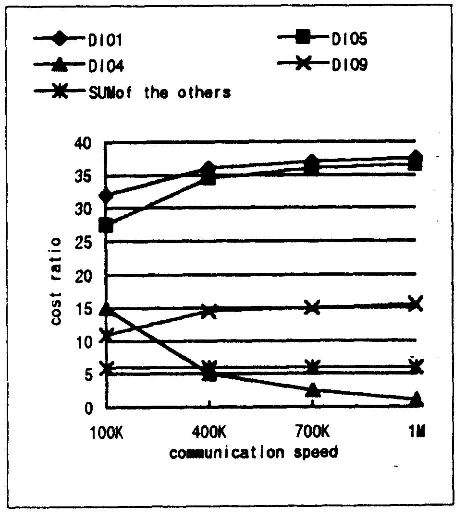

Figure 5-4 Cost Components in DRF

maintained (up to the half of base tables). Therefore the DRF scheme is practically meaningful especially in the distributed environments. The transportation cost is an important factor in both two cases. But even though the worst case such as ultra high communication and huge DF siges situations, the DRF scheme still has some advantages.

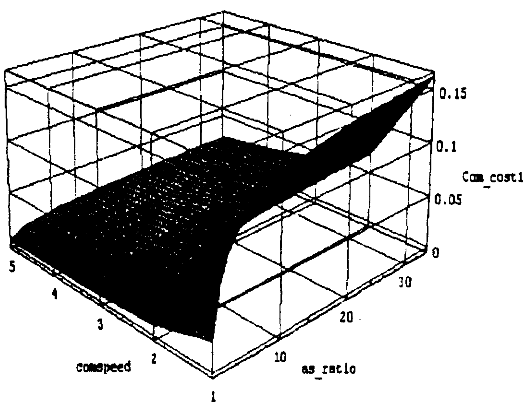

Figure 5-5 Communication cost ratio of DRF

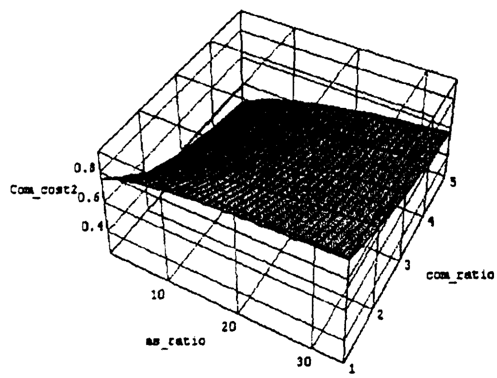

Figure 5-6 Communication cost ratio of Semi-Join

Ⅵ. Summary

A Replication Server scheme with an algorithm DRF is addressed to update multiple views and replicas effciently in a distributed environement. The peculiarity of this algorithm can be summarized as follows: (1) The DRF scheme can reduce the transmission cost significatntly by an effective tuple reduction procedure as a preprocessor before sending the relevant tuples. (2) Using differential files, the DRF scheme can minimize the base table locks, enhancing the systems's performance especially for distributed enviornment. (3) Because it utilizes differential files only that never touches base tables including in joining operations, the scheme can help to realize the distributed database systems.

The performace analyses show that the total cost of this algorithm is closely dependent on the number of differential tuples, the screen factor and the communication speed. As these factors decrease, so does the total cost immediately. The proportion of total cost to transmission cost in algorithm DRF is much smaller than that of semi-joins. Although in the worst case scenario with such factors as 1) ultra high communication rate (say, 100000000 BPS) 2)mammoth differential file magnitudes (up to the size of the base tables) 3) no screened cases the cost benefits are insignificant, the scheme still has some advantages. It is the most important point in solving the complexity of distributed database systems that the scheme can prevent the distributed transactions from 'locking all' the tables.

References

1. Alonso R., Garcia-Molina H., and L. Slem, "Concurrency Control and Recovery for Global Procedures in Federated Datbase Systems," IEEE Quart. Bull. Database Eng. (sept. 1987), 10(3): pp5-11.
2. Bernstein, P. A., Hadzilacos, V., and Goodman, N., Concurrency Control and Recovery in Database Systems, Addison Wesley, 1987.
3. Blakeley, Jose A., "Updating Materialized Database Views," Research Report CS-87-32, Univ. of Waterloo, May 1987.
4. Breibart Y. and Silvershatz A., "Multidatabase Update Issues." In Proc. ACM SIGMOD Int. Conf. on Management of Data, Chicago, June 1988, pp. 135-142.
5. CA Open INGRES, "Replication Server User's Guide", Document Number REP10-9(9)-16200,

1995.

6. Date, C. J., Introduction to Database Systems, Addison-Wesley, 5th ed. vol. 1, 1990.
7. Du, A. K., Elmargarmid, "Quasi-serializability: A Correctness Criterion for Global Concurrency Control in Inter Base". In Proc. 15th Int. Conf. on Very Large Data Bases, Amsterdam, August 1991, pp. 347-355.
8. Goldring, R. "A Discussion of Relational Database Replication Technology," InfoDB Spring 1994.
9. Gorelik, A., Wang, Y. and Deppe, M. "Sybase Replication Server," In Proc. ACM SIGMOD Int. Conf. Management of Data, May 1994.
10. Hanson, E. R., "A Performance Analysis of View Materialization Strategies," In Proc. ACM-SIGMOD Conf., Management of Data, May 1987.
11. Kahler, B. and Risnes, O., "Extending logging for database snapshot refresh," In Proc. Int. Conf. Very Large Data Bases, Brighton, England, Sept. 1987, pp. 389-398.
12. W. Lee, J. Park, S. Kang, "A Differential Join Scheme in Distributed Data Replication," Journal of Database Management, 1996(To appear).
13. Lindsay B. G., Hass L., Mohan C., Pirahesh H., and Wilms H., "A snapshot differential refresh algorithm," in Proc. ACM SIGMOD Conf. Management of Data, June 1986, pp. 53-60.
14. Segev, A. and Fang, W., "Optimal update policies for distributed materialized views," Dept. Computer Science, Lawrence Berkeley Lab., CA, Tech. Report LBL-26104, 1988.
15. Segev, A. and Park, J., "Updating Distributed Materialized Views," IEEE Transactions on Knowledge and Data Engineering, Vol. 1, No.2, Jun. 1989.
16. Sheth A. and Larson J., "TAILOR, A Tool for Updating Views," In working paper of Honeywell CSDD, 1988.
17. Shmueli, O., and Itai, A., "Maintenance of views," in Proc. ACM-SICMOD Conf. Management of Data, Boston, MA, 1984.
18. Sugihara, K., "Concurrency Control Based on Cycle Detection," In Proc. 3rd Int. Conf. on Data Engineering, Los Angeles, Calif. Feb. 1987, pp. 267-274.
19. Sybase, Replication Server Administration, Student Guide, Ver. 1, Sybase Inc. 1994.
20. The L., "Distribute Data Without Choking The Net," Datamation, Jan. 1994.
21. Tompa, F. W. and Blakeley, J. A., "Maintaining Materialized Views Without Accessing Base Data," Information Systems, vol. 13, 1988.
22. White, V. J. "Real User Soulution : Replicated Data," In DB/EXPO'94 Conf. Material, May 1994, pp.231-235.
23. Wookey Lee, Jooseok park, Sukho Kang, "Supporting Materialized Views in Distributed

Database Systems," Journal of the Korea Society of Management Information Systems, Vol. 5, No. 2, Dec. 1995.

24. Yao, S. B., "Approximating block accesses in database organizations," Communications of the ACM, Vol. 20, Apr. 1977.