경영정보학연구
제5권 2호
1995년 12월

저장뷰를 통한 분산 데이타베이스의 구현

이 우 기[1] 강 석 호[2] 박 주 석[3] 이 종 호[4]

Supporting Materialized Views in Distributed Database Systems

In distributed database systems materialized views are useful to solve many problems caused by data replication. This paper deals with the problem of updating materialized views incluing join operations. We use a differential update which uses logs that record the change of base tables in certain periods. This method avoids locking of the base tables so that it makes the system more efficient. We update materialized views periodically to save the updating cost envolved. A screen test is applied to differential files to eliminate tuples that are irrelevant to any of the views being updated. Using these methods, a detailed procedure is addressed to refresh materialized views. Then we show that our update procedure performs better than a semijoin approach.

Keywords: Distributed Databases, Materialized Views, Differential Updates, Semi-Join, Screen Test.

1) 서울대학교 산업공학과
2) 서울대학교 산업공학과
3) 경희대학교 경영학과
4) 기아자동차

I. 서 론

분산 데이타베이스 시스템의 구현을 위해서 데이타를 중복 배치하여 시스템의 가용성(availability)과 성능(performance)의 향상을 도모하는 경우가 많다. 즉, 자주 사용되는 데이타를 필요로 하는 곳에 저장함으로써 접근 비용을 절감할 수 있고 또한 시스템이 고장났을 경우 적어도 하나의 데이타를 이용하게 될 확률을 높이려는 것이다[4].

그러나 실제로 이러한 이득을 얻기 위해서는 데이타의 mutual consistency가 보장되어야 한다. 이를 위해 가장 많이 사용되는 것은 잠금(locking)으로 이 방법은 타사용자의 테이블 접근을 제한한다. 분산 트랜잭션의 경우 신뢰성을 유지하기 위한 protocol로 two-phase commit, three-phase commit 등이 있다 [14]. 이 중 가장 많이 사용되는 two-phase commit은 확실하기는 하나 느리며 통신량이 많고 deadlock이 일어날 수 있다. 분산 데이타베이스의 모든 지역에서 일어나는 모든 갱신에 대해 이러한 과정이 일어나므로 심한 경우 트랜잭션을 거의 수행될 수 없게 한다[19].

또한 같은 데이타의 copy를 가지는 지역으로의 통신이 고장났을 때 이러한 copy들간에 mutual consistency를 유지하는 것이 어려워진다. 이와 같은 통신상의 고장을 partition failure라 한다. 이는 네트워크를 partition이라 불리는 subnetwork으로 분할시킨다. Partition을 처리하기 위한 protocol에는 primary site, voting, grid protocol 등이 있다 [4], [5], [12], [10], [3]. 그러나 위의 프로토콜은 network partition 동안 데이터의 가용성을 심하게 제약한다[7].

이와 같이 데이타를 중복할 경우 생기는 많은 문제점을 해결하면서 동시에 데이타 중복의 효과를 얻기 위한 방안들 중 저장뷰(materialized view)가 가장 뛰어나다 [17], [11], [13], [15], [18], [20]. 데이타베이스의 기본 테이블에서 유도되어 논리적으로는 존재하나 물리적으로는 존재하지 않는 가상뷰(Virtual View)와 달리 저장뷰는 물리적으로 존재하는 것이다. 데이타를 중복하여 배치할 경우 이들이 항상 최신의 값을 유지하도록 해야 하나 저장뷰는 시간적 차이를 두고 데이타가 갱신됨으로써 분산된 데이타의 concurrency control, reliability protocol이 필요하지 않게 되어 위에서 언급된 문제를 해결할 수 있다.

지금까지 이 분야에 대한 연구는 분산 상황을 고려하지 못한 것이 대부분이었고 또한 분산 상황을 고려한 경우에도 기본 테이블 하나만을 이용하는 Selection 뷰나 Selection-Projection 뷰에 한정되어 있었다. 또한 여러 개의 기본 테이블을 조인하는 뷰까지 지원해 줄 수 있어야만 분산 데이타베이스의 구현을 앞당길 수 있는 것이다. 따라서 본 연구에서는 분산 상황하에서 조인 저장뷰(join materialized view)의 갱신을 처리하는 구조가 주요한 논점이 된다.

본 구조에서는 기본 테이블에 일어난 변화를 저장뷰(materialized view)에 반영시켜 주기

위해 테이블 전체를 읽는 방식을 피하고 일정 기간동안 테이블에 일어난 변화를 기록하는 log를 이용하는 디퍼런셜 갱신(differential update) 방법을 사용한다. 이 방법은 테이블의 잠금(locking)을 피함으로 시스템의 성능을 향상시킬 수 있다. 또한 갱신에 드는 비용을 줄이기 위해 이를 주기적으로 갱신하는 방법을 택하였다. 마지막으로 갱신 시 필요한 통신량을 최소화하기 위해 기본 테이블에서 일어난 변화 중에서 저장뷰에 반영되어야 할 부분만을 screen test를 거친 후 저장뷰가 있는 지역으로 보낸다. 위의 방법을 이용하여 분산 상황에서 저장뷰를 효과적으로 갱신하기 위한 효과적인 구조와 알고리듬을 찾고자 한다.

II. 분산 데이타 처리 구조

1. 디퍼런셜 갱신 방식의 개요

저장뷰를 갱신함에 있어서 본 연구에서는 Segev와 Park이 제안한 디퍼런셜 갱신(differential update) 방식[17]을 채택하여 튜플 축소 및 screening을 처리하고[5], 여기서는 그러한 튜플들의 조인 과정을 설명하기로 한다. [2]와 [17]에서는 불필요한 디퍼런셜 튜플의 제거를 위한 여러 기법을 제안하여 기존의 방식 보다 나은 성능을 보여 주었다.

디퍼런셜 갱신 방식은 가장 최근에 저장뷰에 행해진 갱신 직후에 발생한 테이블의 변화만을 기록한 디퍼런셜 파일(differential file)을 이용한다. 그 이유는 저장뷰의 갱신 시 통신량을 최소화하고 기본 테이블 전체에 접근하는 횟수를 줄여 시스템의 성능을 향상시키기 위함이다.

디퍼런셜 파일을 사용할 때 각 튜플을 분별하기 위한 key로는 튜플 생성 시 DBMS가 부여하는 tuple identifier인 VTID를 이용한다. 그리고 튜플에 발생한 변화를 나타내 주기 위해 사용되는 속성으로 Op-Code가 있다. 이는 새로이 입력되는 튜플에는 ins, 삭제가 되면 del를 적어 주며, 변경이 발생할 경우 del_m과 ins_m을 연이어 사용하여 이를 표시해 줄 수 있다.

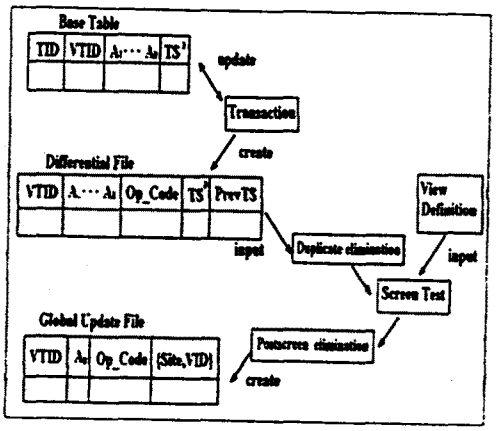
한 튜플에 여러 번의 변화가 발생하였다면 처음과 마지막을 제외한 부분을 반영해 줄 필요가 없는데 이 과정이 duplicate elimination이다. 이를 거친 튜플들은 screen test를 거치게 된다. screen test는 기본 테이블에서 유도된 뷰의 정의를 이용하여 튜플에 발생한 변화를 저장뷰에 반영시켜 줄지 여부를 결정하는 것이다.

Screen test를 통과한 튜플들은 VTID에 따라 sorting하여 같은 VTID를 가지는 튜플의 Op_Code를 합쳐 하나로 만드는데 이것이 postscreening elimination 과정이다. 이 과정을 거친 튜플들은 최종적으로 Global Update File(GUF)을 생성하여 변경 사항을 저장뷰가 있는 여러 지역으로 보내 뷰의 내용을 갱신하게 된다.(그림 1 참조)

2. 조인 저장뷰의 갱신

조인 저장뷰를 이용할 때 가장 큰 문제가 되

5) Blakeley의 방식[2]을 참조.



TID : 튜플의 물리적 identifier
VTID : 튜플의 고유 identifier
A : 속성값(attribute)
TS^B : 테이블의 최종적인 변경 시간
TS^D : 디퍼런셜 파일의 최종적인 변경 시간
opcode : 작업 내역(ins, del, ins_m, del_m)
VID : 뷰 기호(view I.D.)

[그림 1] 저장뷰 갱신 절차

는 것은 조인하여 저장뷰를 생성한 후 어떻게 기본 테이블의 변화를 반영하느냐는 것이다. 또한 새로운 튜플이 입력되거나 기존의 튜플이 삭제 또는 변경되었을 경우 이 튜플들이 새로이 조인되어 저장뷰에 추가되어야 하는지 여부를 결정해 주어야 한다.

이 문제를 해결하기 위해서 위에서 설명한 GUF를 이용한다. 조인에 참여하는 각 테이블은 디퍼런셜 파일을 유지하고 또한 여기서 유도되는 뷰 관리를 위해 GUF[6]를 생성한다. GUF 중 Op_Code가 ins인 튜플의 경우 조인되는 상대 테이블이 위치한 지역으로 보내어 조인한 후 기존의 저장뷰에 추가되어야 한다. 또한 조인 시 사용되는 속성이 변경되어 Op_Code가 mod인 튜플도 또한 새로이 조인되어 추가되어야 한다. 이외의 튜플들은 새로이 조인할 필요 없이 기존의 저장뷰가 위치한 장소에 직접 GUF를 생성하여 보내 이를 갱신한다. 즉, GUF의 VTID와 같은 VTID를 가지는 튜플에 대해 삭제(delete)와 변경(modification)을 행할 수 있다. 이 때 GUF의 schema는 다음과 같다.

GUF(VTID, Au, Op_Code, {Site,VID})
Au : Op_Code가 ins일 경우 모든 해당 속성값의 변화 기록
Op_Code가 del일 경우 공집합
Op_Code가 mod일 경우 새로 입력되는 속성에 대한 값
{Site, VID} : {해당 저장뷰의 위치, 뷰 ID}

한편 조인되는 상대 테이블이 위치한 지역에는 새로이 조인되어야 하는 튜플들이 모이게 된다. 만일 여러 지역의 테이블과 조인이 이루어지는 경우라면 조인이 이루어지는 지역에서는 다른 지역에서 온 튜플들의 크기가 모두 틀리므로 이를 하나의 테이블로 만들어 관리하기가 어렵다. 따라서 새로이 JADF(Join Attribute Differential File)를 생성하여 하나의 테이블로 관리한다. JADF의 schema는 다음과 같다.

6) GUF나 JADF, S 및 R등에 사용되는 아래 첨자는 장소를 나타낸다. 예컨대 GUFi는 'i'장소의 GUF를 또한 Rj는 j 장소의 기본 릴레이션을 각각 의미한다.

JADF= :(Site_ID, VTID, Join_Attribute, Pointer)

Site_ID : 디퍼런셜 파일이 있는 지역의 고유 ID

VTID : 한 튜플의 고유 ID

Join_Attribute:조인 시 사용되는 속성값

Pointer : 저장뷰에서 필요로 하는 속성값으로 이루어진 디퍼런셜 파일의 record와 연결시켜 주는 포인터

위의 방법을 사용하여 조인할 경우 조인은 다음 두 가지로 나누어 처리해야 한다.

1) 조인이 foreign key(composite key일 수 있음)에 의하여 이루어지는 경우
2) 조인이 일반 속성값에 의해 이루어지는 경우

위의 두 경우 모두 디퍼런셜 파일에서 조인되어야 할 튜플을 조인에 참여하는 테이블이 위치한 지역으로 보내 JADF를 생성한다. 1)의 경우 S_i에서만 $GUFi_i$를 이용하여 $JADF_i$를 만들어 줄 수 있다. R_i에 새로운 튜플이 입력되었을 경우 조인 시 이에 대응하는 튜플이 참조집약성(referential integrity rule)에 의해 R_i에 반드시 존재하므로 $JADF_i$를 만들 수 있는 것이다. 또한 foreign key가 변경된 튜플(이 경우 Op_Code는 mod)에 대해서도 $JADF_i$를 만들어야 한다. 이와 달리 테이블 R_i에서는 새로운 튜플이 입력되었더라도 $JADF_i$를 만들 필요가 없다. 왜냐하면 이 경우 새로이 입력된 튜플에 상응하는 테이블 R_i의 튜플이 아직 생성되지 않았기 때문이다. 만약 이 둘이 동시에 생성되더라도 테이블 R_i에 의해 생기는 $JADF_i$로 조인이 이루어지므로 문제가 되지 않는다.

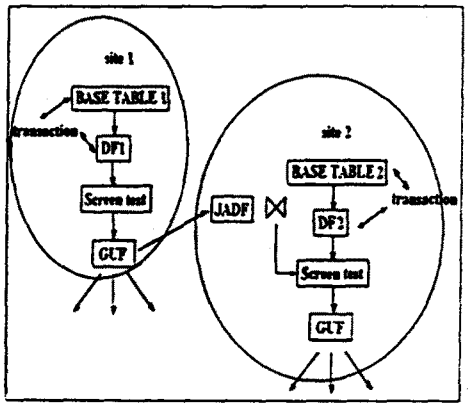
다음 Relation R_i에 의해 site2에 생긴 $JADF_i$의 Join_Attribute 즉 foreign key와 DFR_i의 primary_key와 먼저 비교한다. 만약 두 부분이 일치한다면 조인을 위해 R_i 즉, 전체 기본 테이블을 검색할 필요가 없으므로 처리 시간이 줄어들게 된다. 일치하는 부분이 없다면 R_i 전체를 검색할 수밖에 없다.

2)의 경우는 1)과 달리 조인에 참여하는 모든 테이블의 디퍼런셜 파일 중 새로 입력된 튜플이나 조인 속성값이 변경된 튜플에 의해 JADF를 생성한다. 왜냐하면 일반 속성을 이용하여 조인을 하기 때문에 앞에서 사용된 참조집약성을 이용할 수 없기 때문이다. 또한 JADF와 디퍼런셜 파일만을 이용하여 조인할 수 없으므로 테이블 전체를 검색하여 조인해야 한다.

3. JADF조인 알고리듬(그림 2 참조)

1) DFRi에서 Ui개의 튜플을 읽는다.
2) Duplicate elimination과 screen test를 한다.
3) 위의 튜플들을 sorting한 후 postscreening elimination을 하여 GUFi를 만든다.
4) GUFi의 모든 튜플에서 다음 Sub-Join 알고리듬을 수행한다.

[Sub_Join 알고리듬]



(GUF : Global Update File, JADF:Join Attribute Differential File, DF : Differential File)

[그림 2] JADF를 이용한 저장뷰의 갱신

i) Op_Code가 ins이고 foreign key를 가지는 테이블이면 Sj에 보내 $JADF_i$를 만든다.

i-1) $JADF_i$와 같은 조인 Attribute를 가지는 튜플을 DF_{Ri} 상에서 택해 조인한다. (단 같은 VTID가 있을 경우 Time Stamp가 가장 늦은 튜플과 조인한다. Op_Code가 del인 것은 제외한다.) 만일 DF_{Ri}상에 없으면 R_i전부를 검색하여 조인한다.

i-2) 조인된 튜플을 screen test한 다음 GUF를 만들어서 S_m에 보낸다.

ii) Op_Code가 mod이면 다음 단계로 나뉜다.

ii-1) 저장뷰(materialized view)에서 사용되는 일반 속성이 변경되었을 경우 S_m에 보낸다.

ii-2) modification이 foreign key(=join attribute)의 변경이라면 Sj에 보내 $JADF_i$를 만든다. Sub_Join 알고리듬 i-1)로 간다.

iii) Op_Code가 del이면 S_m에 보낸다.

5)저장뷰를 구성하는 S_m site의 B^+ tree에서 해당 튜플을 찾아 갱신한다.

III. 비용 함수의 계산

위에서 구한 JADF조인 알고리듬을 이용하여 저장뷰를 갱신하였을 때 기존의 방법인 semijoin을 이용한 방법에 비해 어느 정도의 성능 향상을 가져오는 지 알아보기 위해 비용 함수를 계산하였다. 또한 semijoin을 이용할 때의 비용 함수를 계산하여 이와 비교하였다. 비용을 계산할 때 CPU 비용은 지나치게 작으므로 무시하였고 I/O 비용과 통신비용을 그 비교 대상으로 하였다.

1. JADF 알고리듬의 비용 계산

1.1 General Notation

R_p : 장소 p의 기본 테이블(p ∈ {i,j,k})
DF_{Ri} : 기본 테이블 Ri의 디퍼런셜 파일
GUF_i : Ri의 GUF
$JADF_j$: Rj와 조인되는 JADF
S_i : Ri가 위치한 장소
SM_i : materialized 뷰 MVi가 위치한 장소
W_{Ri} : Ri의 튜플 크기
W_B : B^+ tree 레코드의 크기
B : 페이지 크기(bytes)
H_{BSi} : Sj 장소에서 B^+ tree 레코드의 높이
SF : semi join factor
f(N, P, K) : P페이지를 차지하는 파일에서 N개의 튜플 중 K개에 접근할 때 fetch되는 평균 페이지 수 [21]
U_i : DF_{Ri}의 튜플 수
U_i^e : DF_{Ri}에서 duplicate elimination의 결과 생긴 튜플의 수
U_i^s : screen test를 통과한 튜플의 수
U^{it} : DF_{Ri}에서 뷰에 전송되는 튜플의 수
U^{Ri} : R_i의 튜플 수
U^{JADFj} : $JADF_j$의 튜플 수
U^{j_r} : $JADF_j$중 DF_{Ri}와 조인되지 않은 튜플의 수
α_e : duplicate elimination factor
α_s : 뷰 predicate에서의 screen factor
α_P : postscreening elimination factor
W_{ins}, W_{del}, W_{mod} : Op_Code가 각각 ins, del, mod인 GUF 튜플의 평균 크기
W_{JADFi} : JADFj의 평균 튜플 크기
Wmvi : materialized 뷰의 평균 튜플 크기
n_{Ri} : R_i의 페이지 당 튜플 수(=⌈B/W_{Ri}⌋)
Pr_{DFRj} : 해당 튜플이 모두 DF_{Rj}에 있을 확률
$C_{I/O}$: I/O 비용(ms/block)
C_{comm} : 전송율(bits/s)

1.2 JADF 알고리듬의 비용 함수

총 비용은 Si에서 발생하는 비용, SMi에서 발생하는 비용, Sj에서 발생하는 비용으로 나누어 살펴볼 수 있다.

Si에서의 비용=CIO1+CIO2+CCOM1

$$CIO1 = C_{I/O}(U_{i.ins} + U_{i.del} + U_{i.delm} + U_{i.in-m}) W_R/B$$

$$CIO2 = C_{I/O}/O2U_i^s W_{R/}B$$

$$CCOM1 = 8(U^t_{i.ins} W_{ins} + U^t_{i.del} W_{del} + U^t_{i.mod} W_{mod})/C_{comm}$$

SMi에서의 비용=CIO3+CIO4

$$CIO3 = C_{I/O}[(H_{B-SMi}-1) + f(\alpha_s N_r, \alpha_s N_r W_B/B, U^t)]$$

$$CIO4 = C_{I/O}2 \times f(\alpha_s N_r, \alpha_s N_r W_R/B, U^t)$$

Sj에서의 비용=CIO5 + CIO6 + CIO7 + CIO8 + CIO9 + CCOM2

$$CIO5 = C_{I/O}(U_{i.ins} + U_{i.del} + U_{i.delm} + U_{i.insm}) W_R/B$$

$$CIO6 = C_{I/O}2U_i^s W_{R/B}$$

$$CIO7 = C_{I/O} \times U^{JADFj} W_{JADFj}/B$$

$CIO8 = C_{I/O} 2U^{[JADF_i} W_{JADF_i} / B$

$CIO9 = \{C_{I/O}[(H_{B-S_i} - 1) + f(U^{R_i}/U^{R_j} W_{R_j}/B, U^{j-r})\}$

$CCOM2 = 8 \times U^{[JADF_j} \times W_{mvi}/C_{comm} + 8(U^{t}_{j\,del} W_{del} + U^{t}_{j\,mod} W_{mod})/C_{comm}$

따라서 총 비용은 CIO1+CIO2+CIO3+CIO4+CIO5+CIO6+CIO7+CIO8+CIO9 이 된다.

3. PERFORMANCE의 비교

위에서 설명한 두 알고리듬에 기초한 비용 함수를 이용하여 총 비용을 비교함에 있어 아래의 값을 가정하여 계산하였다 : B=4000 bytes, $W_{R_i} = W_{R_j} = 200$ bytes, $W_B = 8$ bytes, $C_{I/O} = 25$ ms/block, $\alpha_e = 0.6$, $\alpha_p = 0.6$, $W_{ins} = 200$ bytes, $W_{del} = 8$ bytes, $W_{mod} = 100$ bytes

다음에서 테이블과 디퍼런셜 파일의 튜플 수를 달리하여 각 알고리듬의 비용을 계산하였다. 그 결과를 [표1]에서 [표4]까지 정리하였다. [표1], [표2], [표3]에서는 두 가지 기본 테이블의 튜플 수를 각각 100만, 10만으로 가정하고 각 테이블에서 유지되는 디퍼런셜 튜플의 수와 통신 속도를 달리하여, Semijoin 알고리듬을 이용하여 갱신할 때와 JADF를 이용하여 갱신할 때의 총비용 비의 계산 결과를 정리하였다. 통신 속도가 느리고, screen factor가 작을수록 그 효과는 컸다. 또한 differential tuple의 수의 변화에 따른 총비용의 비가 어떻게 변화하는 가를 알기 위하여 통신 속도를

[표 1] 총비용의 비 I (Semijoin/JADF)

s.f. \ ccom	10k	50k	100k	500k	1M	10M
0.4	390.5	131.1	74.6	22.1	15.1	8.6
0.1	453.9	155.2	94.7	40.0	32.7	26.0
0.05	520.6	177.8	113.2	55.9	48.3	41.5

(U^{R_i} : 1,000,000, Ui : 10,000, U^{R_j} : 100,000, U^{j} : 1,000)

[표 2] 총비용의 비 II (Semijoin/JADF)

s.f. \ ccom	10k	50k	100k	500k	1M	10M
0.4	91.3	34.2	19.9	6.0	4.1	2.3
0.1	104.4	38.7	24.0	8.4	8.4	6.7

(U^{R_i} : 1000,000, Ui : 50,000, U^{R_j} : 100,000, Uj : 5,000)

[표 3] 총비용의 비 III (Semijoin/JADF)

s.f. \ ccom	10k	50k	100k	500k	1M	10M
0.4	51.0	21.0	12.6	3.9	2.7	1.5
0.4	57.5	22.8	14.3	6.2	5.1	4.1

(U^{R_i} : 1000,000, Ui : 100,000, U^{R_j} : 100,000, Uj : 10,000)

1Mbps로, screen factor를 0.1로 일정하게 유지하고 그 값을 계산하여 [표4]에 정리하였다. 이는 differential tuple의 수를 적게 유지할 수록 JADF방식의 비용 효과가 크다는 것을 보여준다.

또한 전체 비용 중에서 통신이 차지하는 비

[표 4] 총 비용의 비 IV(Semijoin/JADF)

d.f.수 / s.f.	Ui:1만 Uj:1천	Ui:5만 Uj:5천	Ui:10만 Uj:1만	Ui:20만 Uj 2만	Ui:50만 Uj:5만
0.1	32.7	8.4	5.1	3.3	1.7

(U^{Ri} 1000,000, U^{Rj} 100,000, ccom 1M bps)

[표 5] 총비용 중 통신이 차지하는 비율(%)

사용 알고리듬 \ S.F		0.1	0.4
Semijoin		19	47
JADF_JOIN	Ui : 1만 Uj : 1천	0.6	0.8
	Ui : 10만 Uj : 1만	1.0	1.4
	Ui : 50만 Uj : 5만	1.8	3.4

(U^{Ri} : 1000,000, U^{Rj} : 100,000, ccom:1M bps)

율을 구하면 [표5]와 같다. 디퍼런셜 파일을 원래 튜플 수의 1/2정도로 크게 유지할 때에도 Semijoin을 사용하는 것 보다 JADF 알고리듬의 경우 통신비용이 매우 적어진다는 것을 알 수 있다.

또한 통신비용이 어느 정도 절감되는 지를 알아보기 위해 통신비용의 비를 계산하여 정리하면 [표6]과 같다. 이는 JADF 알고리듬을 이용하여 Materialized 뷰를 갱신할 경우 통신비용의 절감이 매우 큼을 보여준다.

Ⅳ. 결론 및 추후 연구 과제

본 논문에서는 분산 상황에서 조인 저장뷰의 갱신을 효과적으로 지원하기 위해 JADF라는 새로운 알고리듬을 제시하였다. 그 의미는 다음과 같다.

1) 디퍼런셜 파일을 이용하여 조인함으로써 기본 테이블의 잠금(locking)을 최대한 줄여 시스템 성능을 향상시켰다. 이 특성은 분산 트랜잭션의 완료율을 향상시키므로 분산 데이타베이스를 현실화할 수 있는 중요한 기반이 된다는 것을 의미한다.
2) 디퍼런셜 파일 중 조인에 필요한 부분만을 선택하여 통신량을 최소화하였다.
3) JADF라는 새로운 구조를 제안하여 하나의 테이블과 이와 조인되는 다수의 테이블 간의 조인 시 이를 주기적으로 한꺼번에 처리함으로써 효율성을 높였다.
4) 본 알고리듬에 따른 비용 함수를 계산한 결과 원래의 테이블에 대한 디퍼런셜 튜플의 비가 적을수록, 그리고 screen factor의 크기가 적을수록 비용 절감의 효과가 컸다. 또한 통신 속도 가 느릴수록 그 효과는 커졌다. JADF조인 알고리듬을 사용할 경우 Semijoin 보다 통신비용의 비중이 적었고 통신비용 절감이 상대적으로 컸다. 추후의 연구 과제로는 Semijoin 이외의 다른 알고리듬과 비교하여 본 알고리듬의 효율을 입증하는 것과 위의 알고리듬을 DBMS의 하나의 모듈로 구현하는 것이다.

참 고 문 헌

Bernstein,P.A.,Hadzilacos,V., and Goodman,N. Concurrency Control and Recovery in Database Systems, Addison-Wesley, 1987.

Blakery,J.A., Larson,P. and Tompa,F.W., "Efficiently updating materialized views," *in Proc. ACM-SIGMOD Conf. Management of Data*, Washington, DC, May 1986.

Cheung,S.Y., Ammar,M.H., and Ahamad,M., "The Grid Protocol: A High Performance Scheme for Maintaining Replicated Data," *IEEE Transactions on Knowledge and Data Engineering*, vol.4,no. 6,Dec.1992.

Davison,S.B.,Garcia-Molina,H., and Skeen,D., "Consistency in partitioned networks," *ACM Comput.Survey*, vol.17, no.3, Sep. 1985.

Gilfford,D.K., "Weighted Voting for Replicated data" in Proc.7th Symposium on Operating Systems Principles, 1979.

Goldring,R. "A Discussion of Relational Database Replication Technology," InfoDB Spring 1994.

Gorelik,A.,Wang,Y. and Deppe,M. "Sybase Replication Server" *in Proc. ACM-SIGMOD Int. Conf. Management of Data*, May 1994.

Hanson,E.R., "A Performance analysis of view materialization stratigies," *in Proc. ACM-SIGMOD Conf.* Management of Data, May 1987.

Horowitz,S. and Teitelbaum,T., "Generating editing environment based on relations and attributes," *ACM Trans. Programming Language Syst.*, vol. 8, oct. 1986.

Jajodia,S. and D.Mutchler, "A Hybrid Replica Control Algorithm Combining Static and Dynamic Voting," *IEEE Transactions on Knowledge and Data Engineering*, vol.1,no.4,Dec.1989.

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.

Kumar,M., "Performance Analysis of a Hierarchical Quorum Consensus Algorithm for Replicated Objects" in Proc. Distributed Computing System, 1990.

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.

Özsu,M.T., and Valduriez,P., Principles of Distrib-

uted Database Systems, Prentice-Hall, 1991.

Roussopoulos,N. and Kang,H. "Principles and Techniques in the design of ADMS+/-," *IEEE Computer*, Dec. 1986.

Segev,A. and Fang,W., "Optimal update policies for distributed materialized views," Dep. Comput. Sci. Res., Lawrence Berkeley Lab., CA, Tech. Rep. LBL-26104, 1988.

Segev,A. and Park,J., "Updating Distributed Materialized Views," *IEEE Transactions on Knowledge and Data Engineering*, Vol.1 ,no.2, June 1989.

Shmueli, O., and Itai, A., "Maintenance of views," *in Proc. ACM-SIGMOD Conf. Management of Data*, Boston, MA, 1984.

Th , L., " Distribute Data Without Choking The Net," *Datamation*, Jan. 1994.

Tompa,F.W. and Blakeley,J.A., "Maintaining Materialized Views without Accessing Base Data," *Inf. Syst.* vol. 13 1988.

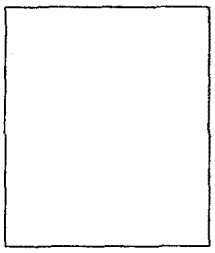
Yao,S.B., "Approximating block accesses in database organizations," *Commun. ACM, No. 148*, vol. 20, Apr. 1977.

◇ 저자소개 ◇

공동저자 이우기는 서울대 산업공학과(학사), (주)금성사에서 근무했고, 서울 대 산업공학과에서 석사학위를 취득했으며, 현재 서울대 산업공학과 박사과 정에 재학중이다. 주요관심분야는 분산데이타베이스, MIS, Manufacturing Database, Concurrent Engineering 등이다.

공동저자 강석호는 서울대 문리대(학사), University of Washington(석사), Taxas A&M University(박사), MIT(객원교수), 현재 서울대 산업공학과 교 수, 최고산업전략과정(AIP) 주임교수, 상공자원부 공업기반기술개발 기획평 가위원. 주요관심분야는 지능형생산시스팀, MIS 및 Database System.

공동저자 박주석은 서울대 산업공학과(학사), KAIST 산업공학과(석사), U.C. Berkerly MIS(박사), Lawlence Berkerly Laboratory 연구원, 현재 경희대학 교 경영학과 부교수, 한국경영정보학회 이사, 한국데이타베이스학회 이사. 주 요관심분야는 분산데이타베이스, MIS, CASE



공동저자 이종호는 현재 기아자동차에 근무중이다. 서울대 산업공학과에서 학사와 석사학위를 취득했다. 주요관심분야는 분산 데이타베이스, Data Replication 등이다.