Client-Server 연구논문

분산 환경을 위한 중복데이타 서버 (replication server) 구조에 관한 연구

이종호
서울대학교

이우기
서울대학교

박주석
경희대학교

강석호
서울대학교

분산 환경을 위한 중복데이타 서버 (replication server) 구조에 관한 연구

이 종호* 이 우기* 박 주석** 강 석호*

* 서울대학교 산업공학과 ** 경희대학교 경영학과

요 약

중복데이타 서버(replication server) 는 자주 사용되는 데이타의 부분 또는 전부를 뷰 형태로 여러 지역에 중복하여 저장함으로써 최종 사용자가 원하는 데이타에 빨리 접근할 수 있도록 해준다. 또한 기본 테이블의 변화된 내용을 주기적으로 뷰에 반영함으로써 데이타 동시성의 문제를 완화하며 통신량을 감소시킬 수 있다.

본 연구에서는 기본 테이블에 일어난 변화를 저장뷰(materialized view) 에 반영시켜주기 위해 테이블 전체를 읽는 방식을 피하고 일정기간 동안 테이블에 일어난 변화가 기록된 로그(log)를 이용하는 디프런셜 갱신(differential update) 방법을 사용한다. 이 방법은 테이블의 잠금(locking) 을 피함으로 시스팀의 성능을 향상시킬 수 있다. 또한 갱신에 관련된 통신량을 최소화하기 위한 기법들을 제안한다.

위의 방법을 이용하여 분산 상황에서 조인 저장뷰(join materialized view)의 갱신을 효과적으로 지원해 주는 중복데이타 서버(replication server) 의 구조에 관해 연구한다.

I. 서 론

1. 연구 배경

분산 데이타베이스 시스템에서 performance와 availability를 높이기 위하여 데이타를 중복하여 배치하는 경우가 있다. 즉, 자주 사용되는 데이타를 여러 곳에 저장함으로써 접근 비용을 절감할 수 있고 또한 시스템이 고장났을 경우 적어도 하나의 데이타를 이용하게 될 확률을 높일 수 있다 [4].

그러나 실제로 이러한 이득을 얻기 위해서는 데이타의 mutual consistency가 보장되어야 한다. mutual consistency란 데이타의 copy가 여러 곳에 중복되어 있을 경우 이 데이타 값이 모두 동일해야 한다는 것이다. 데이타의 consistency를 유지하기 위하여 사용되는 방법에는 locking, timestamping, optimistic concurrency control 등이 있다 [1]. 가장 많이 사용되고 있는 것은 잠금(locking)으로 이는 데이타 갱신시 같은 데이타가 있는 모든 지역의 시스템을 연결하고 동시에 연결된 데이타에 다른 변화가 일어나지 못하도록 lock을 건 후 갱신하는 것이다.

그리고 분산 트랜잭션의 경우 신뢰성을 유지하기 위한 protocol로 two-phase commit, three-phase commit 등이 있다 [14]. 이 중 가장 많이 사용되는 two-phase commit 과정의 첫 단계는 데이타를 저장하고 있는 모든 지역에 갱신해도 좋은가에 대한 신호를 보내는 것이다. 그리고 모든 지역으로부터 갱신해도 좋다는 신호를 받은 후 이를 갱신하는 두번째 단계를 거친다. 이 방법은 확실하기는 하나 느리며 통신량이 많고 deadlock이

일어날 수 있다. 분산 데이타베이스의 모든 지역에서 일어나는 모든 갱신에 대해 이러한 과정이 일어나므로 심한 경우 트랜잭션이 거의 수행될 수 없게 된다 [19].

또한 같은 데이타의 copy를 가지는 지역으로의 통신이 고장났을 때 이러한 copy들 간에 mutual consistency를 유지하는 것이 어려워진다. 이와 같은 통신상의 고장을 partition failure라 한다. 이는 네트워크를 partition이라 불리는 subnetwork으로 분할 시킨다. 이러한 partition이 알려지지 않을 경우 서로 다른 partition에서 독립적으로 일어나는 갱신으로 인해 데이타의 정확성이 유지되지못할 수 있다. Partition을 처리하기 위한 protocol에는 primary site, voting, grid protocol 등이 있다 [4], [5], [12], [10], [3]. 그러나 위의 protocol은 network partition 동안 data의 availability를 심하게 제약한다 [7].

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

Virtual view는 질의어에 의해 요청될 때마다 프로그램이 실행되어 물리적으로는 화일이 존재하지 않지만 실제 존재하는 것처럼 최신의 데이타를 한시적으로 볼 수 있게 하나 저장뷰(materialized view)는 시간적 차이를 두고 갱신된 정보가 테이블 형태로 저장되어 있어 질의어에 의해 원하는 정보를 요청하여 볼 수 있다. 저장뷰(materialized view)는 꼭 현재의 데이타를 볼 필요가 없을 때 사용되며 저장을 위한 디스크 용량이 필요하지만 실행결과가 저장되어 있으므로 virtual view보다 응답시간이 짧다.

이러한 저장뷰(materialized view)를 관리하는 것이 replication server로 이는 저장뷰(materialized view)를 여러 지역에 중복하여 저장함으로써 최종 사용자가 원하는 데이타에 빨리 접근할 수 있도록 해준다. 또한 기본 테이블의 변화된 내용을 주기적으로 뷰에 반영시켜줌으로써 데이타 동시성의 문제를 완화하며 통신량을 감소시켜 준다.

2. 연구의 목적

지금까지 이 분야에 대한 연구는 분산 상황을 고려하지 못한 것이 대부분이었고 또한 분산 상황을 고려한 경우에도 기본 테이블 하나 만을 이용하는 Selection view나 Selction_Projection view에 한정되어 있었다. 또한 여러 개의 기본 테이블을 join하는 view에 대한 효과적인 해결책은 제시하지 못하고 있다. 따라서 본 연구에서는 분산 상황하에서 조인 저장뷰(join materialized view)의 갱신을 위한 중복데이타 서버(replication server)의 구조에 관해 연구하고자 한다.

본 연구에서는 기본 테이블에 일어난 변화를 저장뷰(materialized view)에 반영시켜주기 위해 테이블 전체를 읽는 방식을 피하고 일정기간 동안 테이블에 일어난 변화를 기록하는 log를 이용하는 디프런셜 갱신(differential update) 방법을 사용한다. 이 방법은 테이블의 잠금(locking)을 피함으로 시스팀의 성능을 향상시킬 수 있다. 또한 갱신에 드는 비용을 줄이기 위해 이를 주기적으로 갱신하는 방법을 택하였다. 마지막으로 갱신시 필요한 통신량을 최소화 하기 위해 기본 테이블에서 일어난 변화 중에서 저장뷰(materialized view)에 반영되어야 할 부분만을 screen test를 거친 후 저장뷰(materialized view)가 있는 지역으로 보낸다.

위의 방법을 이용하여 분산 상황에서 저장뷰(materialized view)를 효과적으로 갱신하기 위한 효과적인 구조와 알고리즘을 찾고자 한다.

II. 기존의 연구 현황 및 문제점

저장뷰(materialized view)를 사용할 때 문제가 되는 것이 기본 테이블에서의 변화를 view 상에 어떻게 반영시켜주는가 하는 것이다. 이에 대한 간단한 해결책으로 기본 테이블에 갱신이 일어났을 때마다 테이블을 다시 읽어 view를 새로이 저장하여 반영시키는 방법(full refresh)이 있다 [13], [2]. [13]에서는 베이스 테이블에 별도의 field(Addr, Prev Addr, Time Stamp)를 두어 이를 이용하여 테이블에 일어난 변화를 알아 변화된 부분만을 저장뷰(materialized view)에 반영시켜 줌으로써 통신량을 최소화하도록 하였다. 그러나 이 방법을 사용할 경우 전체 테이블에 lock을 걸어야 하므로 다른 사용자는 이 테이블을 이용할 수 없게 되어 비효율적이다. 또한 이 방법은 각기 다른 갱신 시간을 가지는 여러 저장뷰(materialized view)를 지원해 주지 못한다.

이와 다른 방법으로 기본 테이블을 읽어 변화를 알아내는 것이 아니라 데이타베이스 recovery시 사용되는 log를 읽어 저장뷰(materialized view)를 갱신하는 디프런셜 갱신(differential update) 방법이 있다 [11], [17]. Kahler와 Risnes는 log 화일을 생성하는 두가지 계획을 실험하였다. 첫번째 것은 테이블에 발생한 변화를 기록한 연속적인 log 화일을 이용하는 것으로 log 화일 중 불필요한 부분을 삭제하는 과정이 없는 경우이다. 두번째 것은 tuple identifier를 이용, 불필요한 부분을 삭제하여 condensed log을 만들어 이를 사용하여 저장뷰(materialized view)를 갱신함으로써 전달되는 정보를 줄이며 잠금(locking)을 피할 수 있다. 또한 기본 테이블보다 훨씬 적은 크기인 log를 이용하므로 처리속도를 높일 수 있다. 그러나 log의 각 tuple들이 트랜잭션 시간에 의해 정열되어 있지 않아 각각의 view

갱신시 condensed log을 다시 읽어야하는 단점이 있다 [11].

Segev와 Park은 분산 환경하에서의 differential file을 이용하여 임의의 갱신 시간을 가지는 저장뷰(materialized view)에 관해 연구하였다. 이들은 duplicate elimination, screen test, postscreening elimination을 통하여 갱신에 필요한 통신량을 최소화하였다. 그러나 이 연구는 Selection view나 Selection_Projection view에 한정되어 join을 허용하는 일반적인 view의 갱신을 지원해 주지는 못하였다 [17].

한편 저장뷰(materialized view)의 갱신 시간에 따라 다음 3가지로 나누어질 수 있다. 첫째 기본 테이블상에서의 변화가 일어나자마자 저장뷰(materialized view)를 갱신하는 것이다 [2], [9]. 둘째 저장뷰(materialized view)에 관한 질의가 들어올 때까지 갱신을 연기하는 것이다 [8], [15]. 마지막으로 저장뷰(materialized view)를 주기적으로 갱신하는 것이다 [17].

위의 3가지 방법은 각기 장단점을 가지고 있으며 시스템의 여건과 갱신 비용을 고려하여 하나를 선택할 수 있다. 이 중 저장뷰(materialized view)를 주기적으로 갱신하는 방법은 항상 최신의 갱신된 기본 테이블의 상태를 보여주는 것이 아니라 정기적으로 갱신된 기본 테이블의 결과에 대한 정보를 가지고 있다가 사용자의 요구가 있을 때 바로 전에 갱신된 상태를 보여주는 것이다. 따라서 이 방법은 사용자가 꼭 현재의 자료를 볼 필요가 없을 경우 사용될 수 있다. 또한 이 방법은 갱신을 위한 refresh 작업의 시간적인 조건을 어떻게 주느냐에 따라 다양하게 이용되어질 수 있는 장점이 있다. 즉 refresh 작업에 대한 시간 조건을 매우 빠르게 주어 최신의 데이타를 보게 할 수 있다. Segev 와 Fang은 최적 갱신시간에 대한 연구를 하였다 [16].

한편 현재 나와 있는 DBMS 중 저장뷰(materialized view)의 갱신을 지원하고 있는 것으로는 IBM Data Propagator Relational, Sybase Replication Server, Oracle Version 7 Simple Snapshot, OpenIngres Replicator이 있다 [6]. 이 중 IBM Data Propagator Relational은 분산 상황이 아닌 한 지역내에서 디프런셜 갱신(differential update) 방법을 이용한 조인뷰(join view)의 갱신을 지원하고 있다. 그리고 Oracle Version 7 Simple Snapshot은 디프런셜 갱신(differential update) 방법을 사용하지 않고 full refresh 기법을 사용하여 조인뷰(join view)의 갱신을 지원한다. 또한 Sybase Replication Server는 조인뷰(join view)가 아닌 하나의 테이블에서의 변화를 반영하여 저장뷰(materialized view)의 갱신을 지원한다 [7].

III. 중복데이타 서버(replication server)의 구조

1. 디프런셜 갱신(differential update) 방식의 개요

본 연구에서는 Segev와 Park이 제안한 디프런셜 갱신(differential update) 방식을 채택하여 조인 저장뷰(materialized view)를 갱신하고자 한다. Segev와 Park의 연구는 불필요한 differential tuple의 제거를 위한 여러 기법을 제안하여 기존의 다른 연구보다 더 나은 성능을 보여 주었다 [17].

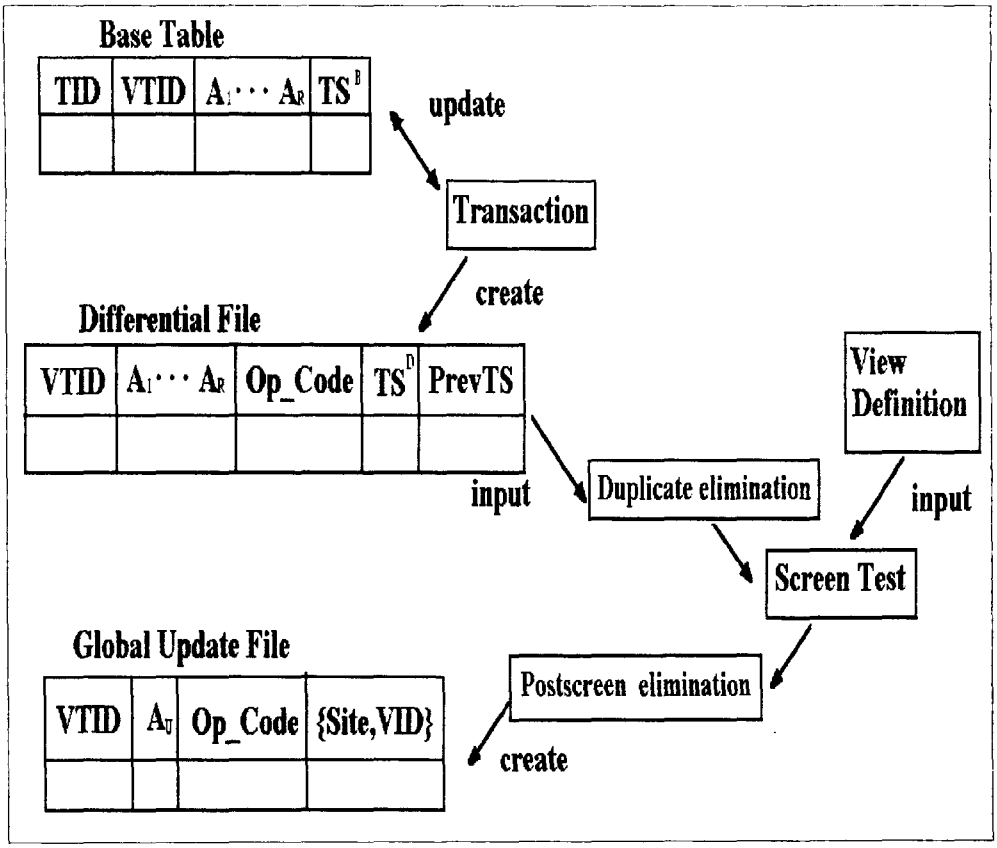
디프런셜 갱신(differential update) 방식은 가장 최근에 저장뷰(materialized view)에 행해진 refresh후에 테이블에 발생한 변화만을 기록한 differential file을 이용한다. Differential file을 사용하는 이유는 저장뷰(materialized view)의 갱신시 통신량을 최소화하고 기본 테이블 전체를 건드리는 회수를 줄여 시스템의 성능을 향상시키기 위함이다.

Differential file은 기본 테이블에 발생하는 모든 변화를 변화가 발생한 시간과 함께 기록하는 file로 일종의 log이다. Differential file을 사용할 때 각 tuple을 분별하기 위한 key로는 tuple 생성시 DBMS가 부여하는 tuple identifier VTID를 이용한다. 그리고 tuple에 발생한 변화를 나타내주기 위해 사용되는 attribute로 op_code가 있다. 새로이 입력이 될 경우 ins, 삭제가 되면 del를 적어주는 것이다. tuple의 attribute에 변경이 발생할 경우 del_m과 ins_m을 연이어 사용하여 이를 표시해 줄 수 있다.

주기적으로 저장뷰(materialized view)를 갱신하기 위해서는 일정 기간 동안의 differential file을 읽어 변화된 부분만을 반영해준다. 이 때 한 tuple에 여러번의 변화가 발생하였다면 처음과 마지막을 제외한 부분을 반영해줄 필요가 없다. 따라서 이 부분을 제거해야 하는데 이 과정이 duplcate elimination이다.

Duplicate elimination을 거친 tuple들은 screen test를 거치게 된다. screen test는 기본 테이블에서 유도된 view의 정의를 이용하여 tuple에 발생한 변화를 저장뷰(materialized view)에 반영시켜줄지 여부를 결정하는 것이다.

Screen test를 통과한 tuple들은 VTID에 따라 sorting하게 된다. sorting을 하는 이유는 같은 VTID를 가지는 tuple의 op_code를 합쳐 하나로 만들기 위함이다. 이것이 postscreening elimination 과정이다. 위의 모든 과정은 저장뷰(materialized view)를 갱신할 때 통신량을 최소로 만들기 위해 만들어진 것이다. 최종적으로 postscreeen elimination을 거친 tuple들은 Global Update File(GUF)를 생성하여 변경사항을 저장뷰(materialized view)가 있는 여러 지역으로 보내 view의 내용을 갱신하게 된다. (그림 1 참조)



TID : physical identifier of a tuple
VTID : unique identifier of a tuple
A : attribute
TS^B : the time the base table was last updated
TS^D : the time the differential file was appended to DF
op_code : operation code (ins,del,ins_m,del_m)
VID : view I.D.

[그림 1] Materialized view 갱신 절차

2. 조인 저장뷰(Join materialized view)의 갱신을 지원하는 중복데이타 서버(replication server)의 구조

조인 저장뷰(Join materialized view)를 이용할 때 가장 큰 문제가 되는 것은 join 하여 저장뷰(materialized view)를 생성한 후 어떻게 기본 테이블의 변화를 반영하느냐는 것이다. 또한 새로운 tuple이 입력되거나 기존의 tuple이 삭제 또는 변경되었을 경우 이 tuple들이 새로이 join되어 저장뷰(materialized view)에 추가되어야 하는지 여부를 결정해주어야 한다.

이 문제를 해결하기 위해서 위에서 설명한 GUF를 이용한다. Join에 참여하는 각 테이블은 differential file을 유지하고 또한 여기서 유도되는 view 관리를 위해 GUF를 생성한다. GUF 중 op_code가 ins인 tuple의 경우 join되는 상대 테이블이 위치한 지역으로 보내어 join한 후 기존의 저장뷰(materialized view)에 추가되어야 한다. 또한 join시 사용되는 attribute가 변경되어 op_code가 mod인 tuple도 또한 새로이 join되어 추가되어야 한다. 이외의 tuple들은 새로이 join할 필요없이 기존의 저장뷰(materialized view)가 위치한 site에 직접 GUF를 생성하여 보내 이를 갱신한다. 즉, GUF의 VTID와 같은 VTID를 가지는 tuple에 대해 delete와 modification을 행할 수 있다.

이 때 GUF의 schema는 다음과 같다.

GUF(VTID, Au, Op_Code, {Site,VID})

Au : Op_Code가 ins일 경우 해당 attribute에 대한 모든 입력 값

Op_Code가 del일 경우 공집합

Op_Code가 mod일 경우 새로 입력되는 attribute에 대한 값

{Site,VID} : { 해당 저장뷰(materialized view)가 위치한 지역,

view ID }

한편 join되는 상대 테이블이 위치한 지역에는 새로이 join되어야 하는 tuple들이 모이게 된다. 만일 여러 지역의 테이블과 join이 이루어지는 경우라면 join이 이루어지는 지역에서는 다른 지역에서 온 tuple들의 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.

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

Site_ID : Differential file이 생성된 지역의 고유 ID

VTID : 한 tuple의 고유 ID

Join_Attribute : Join시 사용되는 attribute

Pointer : 저장뷰(materialized view)에서 필요로 하는 attribute로

이루어진 differential file의 record와 연결시켜 주는 포인터

위의 방법을 사용하여 join할 경우 join은 다음 두가지로 나누어 처리해야 한다.

1) Join이 foreign key(composite key일 수 있음)에 의하여 이루어지는 경우

2) Join이 일반 attribute에 의해 이루어지는 경우

앞으로의 설명을 위해 다음 notation을 정의한다. 또한 다음 사항을 가정한다. (가정 : 각 site에서는 join을 이용한 저장뷰(materialized view) 생성에 관한 정보를 모두 가지고 있다. 즉 join에 참여하는 relation의 위치와 view의 정의에 관한 정보를 가지고 있다.)

■ General Notation :

Rk : relation (k = i 이면 Rk는 foreign key를 가지는 relation,
k= j 이면 Rk는 Ri와 join되는 relation, $i \neq j$)

DF_{Ri} : Ri의 differential file

GUFi : Ri의 GUF

JADFj : Rj와 join되는 JADF

Si : Ri가 위치한 site

Sm : 저장뷰(materialized view)가 위치한 site

U_i : DF_{Ri}의 tuple 수

1),2) 두 경우 모두 differential file에서 join되어야 할 tuple을 join에 참여하는 relation이 위치한 지역으로 보내 JADF를 생성한다. 1)번의 경우 Sj에서만 DF_{Ri}을 이용하여 JADF를 만들어 수 있다. Ri에 새로운 tuple이 입력되었을 경우 join시 이에 대응하는 tuple이 referential integrity에 의해 Rj에 반드시 존재하므로 JADF를 만들 수 있는 것이다. 또한 foreign key가 변경된 tuple(이 경우 op_code는 mod)에 대해서도 JADF를 만들어야 한다. 이와 달리 relation Rj에서는 새로운 tuple이 입력되었더라도 JADF를 만들 필요가 없다. 왜냐하면 이경우 새로이 입력된 tuple에 상응하는 relation Ri의 tuple이 아직 생성되지 않았기 때문이다. 만약 이 둘이 동시에 생성되더라도 relation Ri에 의해 생기는 JADF로 join이 이루어지므로 문제가 되지 않는다.

다음 Relation Ri에 의해 site2에 생긴 JADF의 join attribute 즉 foreign key와 DF_{Rj}의 primary_key와 먼저 비교한다. 만약 두 부분이 일치한다면 join을 위해 Rj 즉, 전체 기본 테이블을 search할 필요가 없으므로 처리시간이 줄어들게 된다. 일치하는 부분이 없다면 Rj를 search할 수 밖에 없다. 이를 하나의 알고리즘으로 만들면 다음과 같다.

■ 알고리즘 JADF_JOIN_MV ([그림 2], [그림 3] 참조)

1) DF_{Ri}에서 U_i개의 tuple을 읽는다.

2) Duplicate elimination과 screen test를 한다.

3) 위의 tuple들을 sorting한 후 postscreening elimination을 하여 GUFi를 만든다.

4) GUFi의 모든 tuple에서 다음 Sub_Join 알고리즘을 수행한다.

[Sub_Join 알고리즘]

i) Op_Code가 ins이고 foreign key를 가지는 relation이면 Sj에 보내 JADFj를 만든다.

i-1) JADFj와 같은 Join Attribute를 가지는 tuple을 DF_{Rj} 상에서 택해 join한다. (단 같은 VTID가 있을 경우 Time Stamping이 가장 늦은 tuple과 join한다. op_code가 del인 것은 제외한다.) DF_{Rj}상에 없으면 Rj전부를 search하여 join한다.

i-2) Join된 tuple을 screen test한 다음 GUF를 만들어서 Sm에 보낸다.

ii) Op_Code가 mod이면 다음 단계로 나뉜다.

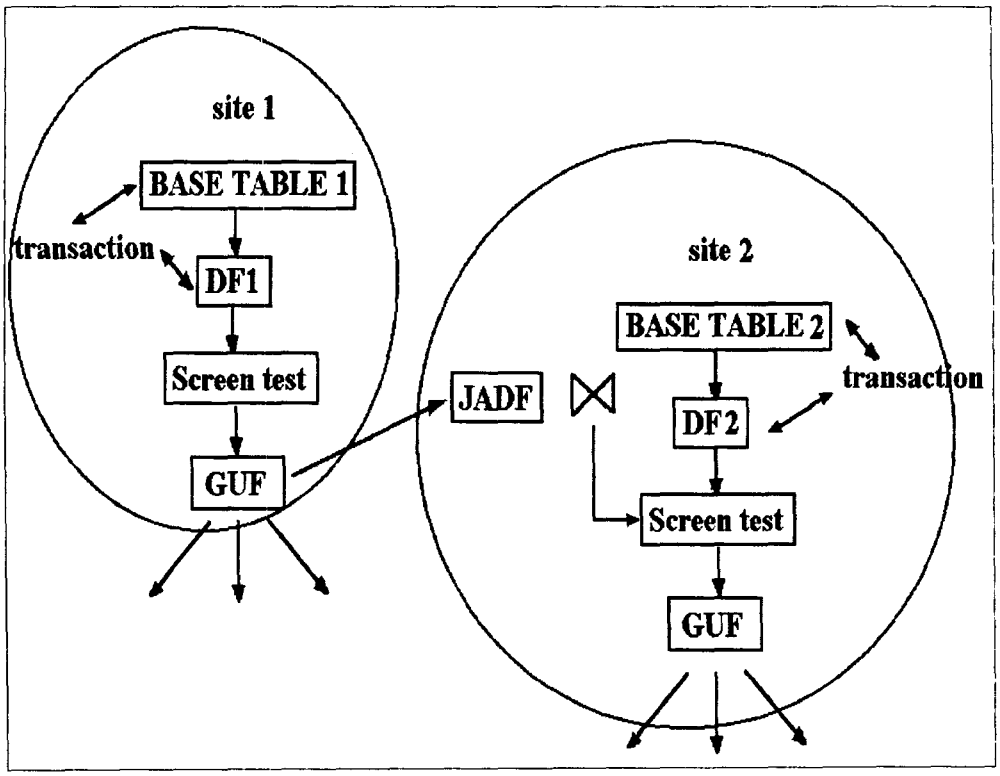
1) 저장뷰(materialized view)에서 사용되는 일반 attribute가 변경되었을 경우 Sm에 보낸다.

2) modification이 foreign key(= join attribute)의 변경이라면 Sj에 보내 JAIDFj를 만든다.

Sub_Join 알고리즘 i-1)로 간다.

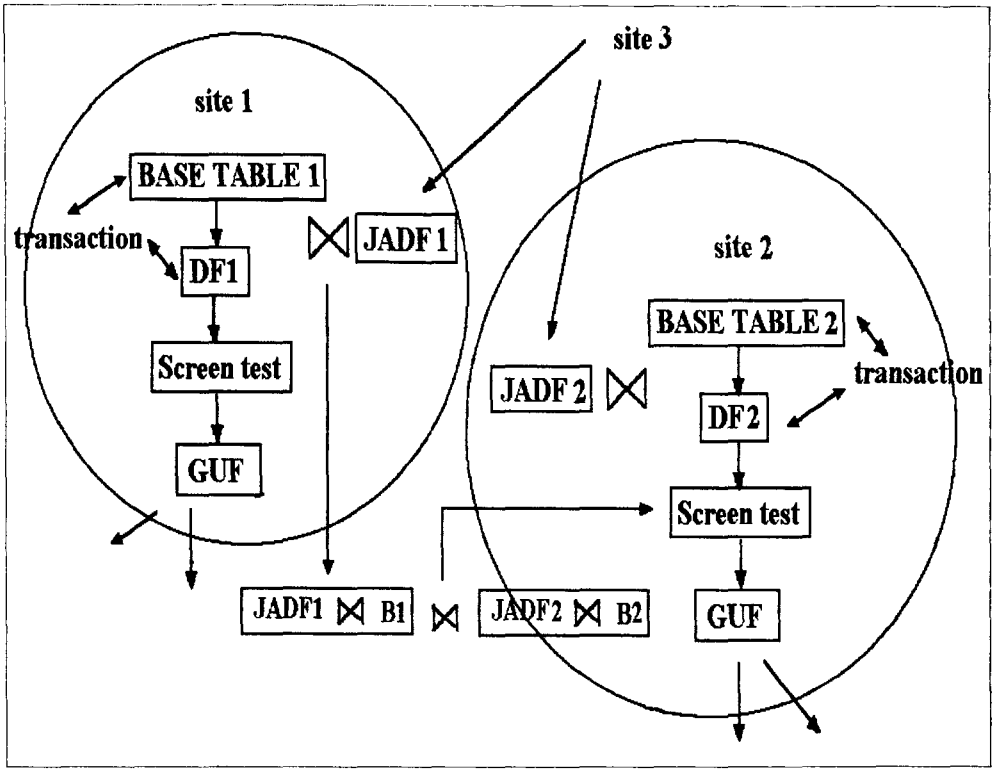
iii) Op_Code가 del이면 Sm에 보낸다.

5) 저장뷰(materialized view)를 구성하는 Sm site의 B^+ tree에서 해당 tuple을 찾아 갱신한다.



(DF : Differential File)

[그림 2] JADF를 이용한 저장뷰(materialized view)의 갱신
(foreign key를 이용한 join)



[그림 3] JADF를 이용한 join 저장뷰(materialized view)의 갱신
(composite key 이용)

2)번의 경우는 1)과 달리 join에 참여하는 모든 relation의 differential file 중 새로이 입력된 tuple이나 join attribute값이 변경된 tuple에 의해 JADF를 생성한다. 왜냐하면 일반 attribute를 이용하여 join을 하기 때문에 앞에서 사용된 referential integrity rule을 이용할 수 없기 때문이다. 또한 JADF와 differential file만을 이용하여 join할 수 없으므로 테이블 전체를 search하여 join해야 한다.

3. 예 제

위에서 설명한 방법을 다음 다음 예제에 적용하였다.

: join에 참여하는 relation은 둘이며 foreign key를 이용하여 join할 경우

(예제를 위해 사용되는 각 relation의 schema는 아래와 같다.)

EMP(VTID, ENO, ENAME, JNO)

ENO : Employee 번호

ENAME : Employee 이름

JNO : Employee가 참여하고 있는 PROJECT 번호)

PROJECT(VTID, JNO, JNAME, BUDGET)

JNO : PROJECT 번호

JNAME : PROJECT 명

BUDGET : PROJECT의 예산

단 밑줄은 각 relation의 primary key이며 음영을 둔 부분은 foreign key이다. 또한 relation EMP는 site1에 relation PROJECT는 site2에 저장되어있다. EMP와 PROJECT를 이용한 저장뷰(materialized view) MV1은 다음과 같이 정의된다.

```
CREATE VIEW MV1 AS
SELECT EMP.ENAME,PROJECT.JNAME, PROJECT.BUDGET
FROM EMP, PROJECT
WHERE EMP.JNO = PROJECT.JNO and EMP.AGE > 30
```

Differential file을 이용하여 materialized view를 구축하기전에 각 relation의 상태와 이를 이용하여 생기는 MV1은 아래와 같고 site 3에 저장된다고 가정한다.

EMP

VTID1	ENO	ENAME	AGE	JNO
1	E1	LEE	31	J1
2	E2	KIM	25	J2
3	E3	PARK	36	J2
4	E4	YUN	43	J3
5	E5	BAE	29	J5
6	E6	SIN	37	J3
7	E7	HAN	28	J2

PROJECT

VTID2	JNO	JNAME	BUDGET
101	J1	CAD	100
102	J2	CAM	300
103	J3	OR	150
104	J4	LP	240
105	J5	MRP	450

MV1

VTID1	VTID2	ENAME	JNAME	BUDGET
1	101	LEE	CAD	100
3	102	PARK	CAM	300
4	103	YUN	OR	150
6	103	SIN	CAM	300

[그림 4] 예제를 위한 테이블과 이를 이용한 materialized view

site 1에서 EMP에 대한 갱신으로 다음과 같은 differential file이 생성되었다.

VTID1	ENO	ENAME	AGE	JNO	OP_CODE
8	E5	CHOI	35	J4	ins
3	E3	PARK	36	J2	del$_m$
3	E3	PARK	36	J4	ins$_m$
5	E5	BAE	29	J5	del$_m$
5	E5	BAE	29	J4	ins$_m$
5	E5	BAE	29	J4	del
1	E1	LEE	31	J1	del

[그림 5] EMP의 differential file

위의 differential file에 duplicate elimination을 할 경우 VTID가 5인 tuple중 op_code가 ins$_m$인 것([그림 5]에서 음영은 둔 tuple)은 필요없게 되어 삭제할 수 있다. 그 결과는 [그림 6]과 같으며 screen test를 통과한 tuple은 [그림 7]과 같다.

VTID1	ENO	ENAME	AGE	JNO	OP_CODE
8	E5	CHOI	35	J4	ins
3	E3	PARK	36	J2	del$_m$
3	E3	PARK	36	J4	ins$_m$
5	E5	BAE	29	J5	del$_m$
5	E5	BAE	29	J4	del
1	E1	LEE	31	J1	del

[그림 6] duplicate elimination 후의 EMP differential file

VTID1	ENO	ENAME	AGE	JNO	OP_CODE
8	E5	CHOI	35	J4	ins
3	E3	PARK	36	J2	delm
3	E3	PARK	36	J4	insm
1	E1	LEE	31	J1	del

[그림 7] Screen test 후 AGE가 30보다 큰 differential file

위의 differential file에 postscreening elimination을 할 경우 다음의 결과를 얻을 수 있다.

VTID1	ENO	ENAME	JNO	OP_CODE
8	E5	CHOI	J4	ins
3	E3	PARK	J4	mod
1	E1	LEE	J1	del

[그림 8] postscreening elimination 후의 differential file

Op_code가 ins인 tuple과 mod이고 foreign key값이 변경된 tuple을 이용하여 생기는 JADF는 다음과 같다.

SITE_ID	VTID1	J_A
S1	8	J4
S1	3	J4

→

ENAME
CHOI
PARK

[그림 9] site 2에 생기는 JADF

또한 op_code가 del인 tuple에 대해서는 GUF을 만들어 MV1이 위치한 site3에 직접 보낸다. 이 예에서 생기는 GUF는 (1; del; {S3,MV1})와 같다.

위의 조건하에서 새로이 추가되고 삭제된 부분을 기존의 materialized view에 반영하기 위한 과정은 아래와 같다.

1. EMP의 differential file 중 op_code가 del인 것은 GUF를 materialized view가 위치한 site로 보내 이와 VTID가 일치하는 것을 모두 지운다.

VTID1	VTID2	ENAME	JNAME	BUDGET
3	102	PARK	CAM	300
4	103	YUN	OR	150
6	103	SIN	CAM	300

[그림 10] GUF에 의해 변경된 materialized view

2. EMP의 differential file 중 opcode가 ins인 것과 op_code mod이고 foreign key값이 변경된 것을 site2에 보내 JADF를 만든다.

3. site2에서 JADF와 B의 differential file와 비교하여 JADF의 join attribute와 differential file의 primary key가 일치하면 이를 join하고 screen test 한 후 materialized view가 위치한 지역으로 보내 추가한다. 일치하는 것이 없을 경우 table B를 모두 search하여 위와 같은 작업을 한다.

VTID1	JNO	OP-CODE
8	J4	ins
3	J4	mod

→

ENAME
CHOI
PARK

⋈

VTID2	JNO	JNAME	BUDGET
104	J4	LP	240

[그림 11] JADF를 이용한 join

VTID1	VTID2	ENAME	JNAME	BUDGET
4	103	YUN	OR	150
6	103	SIN	CAM	300
8	104	CHOI	LP	240
3	104	PARK	LP	240

[그림 12] 예제의 최종 materialized view

IV. 결론 및 추후 연구 과제

본 논문에서는 분산 상황에서 조인 저장뷰(join materialized view)의 갱신을 효과적으로 지원해 주는 중복데이타 서버(replication server)의 구조에 관해 연구하였다. 본 논문에서 제시한 알고리즘의 특징은 다음과 같다.

1) differential file을 이용하여 join함으로써 기본 테이블의 잠금(locking)을 최대한 피해 시스템 성능을 향상시켰다.
2) differential file 중 join에 필요한 부분만을 선택하여 통신량을 최소화 하였다.
3) JADF라는 새로운 구조를 제안하여 하나의 테이블과 이와 join되는 다수의 테이블간의 join시 이를 주기적으로 한꺼번에 처리함으로써 효율성을 높였다.

추후 연구과제로는 본 논문에서 제시한 알고리즘이 어느정도의 성능 향상을 가져오는지 알아보는 것이다. 이를 위해 비용함수를 세워 위의 알고리즘을 사용하지 않고 저장뷰(materialized view)를 갱신하는 여러가지 경우와 상호 비교해야 할 것이다.

참고 문헌

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

[2] 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

[3] 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

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

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

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

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

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

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

[10] 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

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

[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] Özsu,M.T., and Valduriez,P., Principles of Distributed Database Systems, Prentice-Hall,1991

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

[16] 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

[17] Segev,A. and Park,J., "Updating Distributed Materialized Views," IEEE Transactions on Knowledge and Data Engineering,Vol.1 ,no.2,june 1989.

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

[19] Thé, L., " Distribute Data Without Choking The Net" Datamation,jan., 1994

[20] Tompa,F.W. and Blakeley,J.A., "Maintaining Materialized Views without Accessing Base Data," Inf. Sys. vol. 13 1988