Intranet 환경에서 복제일관성을 위한 점진적 갱신 알고리즘의 설계 및 구현

함병훈°, 서창석, 이기택, 이병욱
경원대학교 전자계산학과

Design and Implementation of Incremental Update Algorithm for Replication Consistency in Intranet

Byoung Hoon Ham°, Chang Seok Seo, Ki Taek Lee, Byung Wook Lee
Dept. of Computer Science, Kyungwon Univ.

요 약

CERN에서 1992년 웹이 발표된 이후 기업내 정보 환경에 많은 변화중 가장 두드러진 부분이 인트라넷/익스트라넷 환경으로 대표되는 기업내 정보 시스템이다. 이 환경을 통하여 수많은 사용자에게 내부의 정보를 지역과 클라이언트에 관계없이 제공하고 있으므로 기업내 데이터베이스를 연동하는 것이 중요한 비중을 차지했다. 그러나 과거의 기술로는 데이터베이스 연동시 상태정보를 지속적으로 유지할 수 없었고, 이로 인해 웹을 통한 서버와 다수의 클라이언트의 공동 작업 수행시 데이터에 대한 일관성 관리할 수 없었다.

본 논문에서는 새로운 웹과 데이터베이스의 연동을 사용하여 지속적인 상태유지를 가능하게 하고 여기에서 발생하는 공유데이터에 대한 일관성 유지를 기존 낙관적 병행제어 알고리즘인 후진(Backward) 및 전진(Foreword) 방법 대신 실시간 공유데이터의 공동작업에 적합한 점증적 갱신 알고리즘을 제안한다. 이 방법은 검증단계에서 충돌연산으로 인한 재시작을 줄이며, 사전에 불필요한 재시작을 방지하는 방법으로 자료 갱신시 다른 클라이언트들에게 변경을 통보하며, 무효화 작업을 각 클라이언트에서 수행함으로써 서버의 부하를 줄일 수 있다.

I. 서 론

현대 사회의 치열한 경쟁 속에서 성공하기 위해서는 정확한 정보를 시기 적절하게 접하고 공유할 수 있는 환경을 갖추는 것이 필수적이다. 이 문제는 새로운 이슈라기보다 컴퓨터 기술의 발전과 더불어 부단히 그 해결 방법을 모색해온 분야로써 정도의 차이는 있지만 나름대로의 진전을 이루었다. 그러나 현재까지의 구현방법은 잠재적인 비효율성과 이로 인해 발생하는 비용의 부담을 감소하기에는 역부족이었다.

한 예로 클라이언트/서버 환경의 경우 정보의 갱신과 시스템 유지보수를 담당하는 관리자들에게 과대한 업무부담을 주게 되었다. 이 문제의 해결을 위해 필요한 정보가 필요할 때 전달할 수 있을 것, 전달된 정보가 최신의 정확한 정보임을 보장할 수 있을 것, 각각의 정보는 단일 소스에서 관리될 수 있을 것, 정보의 갱신 및 관리는 그 정보를 최초로 생성하며 담당하는 사람들이 처리할 수 있을 것과 같은 기술적인 사항들이 요구되었다. 그러나 정작 가장 큰 문제는 복잡한 요구사항들에 대해 각각 별도의 해결 방법을 찾아야 한다는 것이다. 하지만 인터넷을 기반으로 한 인트라넷을 구축하면 이 문제들을 단번에 해결할 수 있다. 인트라넷의 탄생이 현실화되면서 인터넷에 관련된 다양한 기술들을 최대한 활용하여 이러한 난제를 해결할 수 있게 되었다[1]. 이러한 환경에서 웹에 기업내 데이터 저장소인 데이터베이스를 연동하는 것은 아주 중요한 기술로 인식되어 왔으며 지금도 계속 발전하는 분야이다. 기존의 웹 환경에서는 정보 공유시 데이터베이스와 웹의 연결에 CGI나 자바 서브릿 또 IDC/HTX 같은 방식을 사용하였다. 그 결과로 발생하는 웹의 특성상 트랜잭션 마다 정보 서버를 새로 접속한다는 단점이 있기 때문에 대형 데이터베이스 시스템의 경우 사용자의 질의(query)마다 시스템 연결 및 초기화에 걸리는 시간이 큰 부담이 될 수 있었고, 서버의 데이터베이스와 연결단절(*Connectless*) 문제로 인하여 지속적인 상태유지와 공유데이터의 일관성을

보장하기가 힘들었다. 이러한 단점을 해결하기 위하여 JDBC, ODBC/OLE-DB/ADO와 같은 데이터베이스 미들웨어와 자바애플릿 또는 액티브X[7][8] 같은 새로운 직접 연결기술이 개발되었고 이것을 사용함으로써 브라우저 내에서 동적으로 데이터베이스의 정보를 처리할 수 있고 각각의 상태정보를 유지할 수 있게 되었으며 그 결과 기존 분산환경에서 대두되었던 클라이언트 서버간 중복데이터간의 일관성 문제가 웹 환경의 클라이어트와 서버간에도 발생하게 되었다. 이를 해결하기 위한 방안으로 본 논문에서는 기존의 낙관적 병행제어기법을 응용한 점진적 갱신 알고리즘으로 Intranet상에서 발생하는 중복데이터의 일관성유지 문제를 개선하고자 한다.

II. 관련연구

2.1 Web과 데이터베이스연동

웹과 데이터베이스의 연동은 크게 간접연결(State-less)과 직접연결(State Oriented)의 2가지 방식으로 나누어질 수 있다.

2.1.1 간접연결(Stateless)

간접 연결 방법(Stateless)이란 웹브라우저와 데이터베이스가 웹을 경유해 연결되는 방법을 말하며 간접연결 도구로는 CGI, 서블릿, 웹 확장 등을 사용한다. 이 방법을 이용하면 단순한 조회나 검색 같은 경우 유용한데, 이는 간접 연결 방법이 쉽고 빠르게 작성할 수 있기 때문이다. 그러나 브라우저와 데이터베이스간의 연결이 지속되지 않고 상태 정보가 유지되지 않는다. 따라서 한 웹 페이지에서 다른 웹 페이지로 이동할 경우 이전의 상태 정보를 잃어버리게 되므로 데이터베이스에 새로운 연결을 맺어야한다. 즉 전자 상거래처럼 상태 정보를 유지해야 할 경우나 지속적으로 데이터베이스를 조작할 필요가 있는 경우에는 적합하지 않다.

2.1.2 직접연결(State Oriented)

직접 연결 방법(State Oriented)은 웹브라우저의 응용 프로그램과 데이터베이스 간의 연결이 지속되고, 상태 정보가 유지된다. 따라서 사용자와의 지속적인 대화형 응용 시스템을 작성할 수 있다. 직접 연결 방법의 특징은 웹브라우저 내의 응용 프로그램이 데이터베이스와 직접 통신할 수 있다는 것이다. 이를 위해서는 웹브라우저 내에 자바 애플릿이나 액티브X와 같이 다운로드 가능한 동적인 응용 프로그램이 필요하다.

1) SUN Soft의 연결방법

자바 솔루션을 사용할 경우 사용되는 기술들로는 JDBC나 JDBC미들웨어, 자바 애플릿, 자바빈스 등을 들 수 있으며, JDBC는 자바에서 제공하는 데이터베이스 연결 기능으로 CLI(Call Level Interface)를 준수하기 때문에 매쏘드 호출 차원에서 데이터베이스를 조작할 수 있도록 한다. 또한 JDBC 미들웨어는 JDBC를 multi-tier 차원에서 관리 할 수 있게 해주기 때문에 JDBC를 보다 광범위한 영역에서 사용할 수 있게 해주며, 클라이언트 측면에서는 웹서버에 다운로드 받은 후 JDBC에게 직접 연결해 작동하는 JDBC 애플릿이 위치한다[7]. 따라서 JDBC애플릿의 사용으로 데이터베이스에 지속적인 연결과 상태정보를 유지할 수 있다.

2) MicroSoft의 연결 방법

JDBC와 마찬가지로 CLI(Call Level Interface)를 지향하는 Microsoft의 대표적인 솔루션이 ODBC이다. ODBC 솔루션은 OLE(Object Linking & Embedding)와 COM(Component Object Model) 기술의 데이터베이스 확장 판인 OLE DB를 하위 구조로 하고, 이를 ActiveX 환경에서 쉽게 사용할 수 있는 인터페이스인 ADO(ActiveX Data Object)를 제공하고 있다[8]. 이 방법도 SUN Soft계열의 연결방법과 마찬가지로 데이터베이스에 지속적인 연결과 상태정보를 유지할 수 있다[8].

2.2 병행제어(Concurrency Control)

병행제어는 두 개 이상의 트랜잭션이 데이터베이스의 한 자료를 동시에 접근하는 경우 발생하는 트랜잭션의 충돌을 막기 위하여 트랜잭션의 직렬성을 유지하는 방법이다. 특히 자료를 갱신하는 연산들이 중복될 경우에 오류가 발생하는데, 이들을 해결하기 위한 대표적인 방법으로 데이터를 배타적으로 접근하게 하는 록킹기법과 타임 스템프 순서화 기법, 낙관적 병행기법이 있다.

2.2.1 낙관적 병행제어(Optimistic Concurrency Control)

낙관적 병행제어 기법은 트랜잭션이 실행되는 동안에는 아무런 검사를 실시하지 않으며 갱신은 실행이 다 끝날 때까지 데이터베이스의 데이터에 직접 반영시키지 않는다.

트랜잭션이 실행되는 동안 데이터 충돌이 발생되었을 경우 블록킹을 사용하지 않으면서 트랜잭션들을 병행적으로 수행하도록 한다. 또한 교착상태가 발생하지 않기 때문에 2단계 록킹 기법과 같이 교착상태를 발견하기 위한 오버헤드가 없다[4]. 또한 낙관적 병행제어기법의 각 트랜잭션들은 판독 단계와 평가 단계, 기록단계의 세 단계로 수행되며, 기록 보다 판독이 많은 응용자 트랜잭션의 길이가 짧은 응용, 충돌이 적은 응용 등에서 좋은 성능을 나타내며 충돌발생시 보다 적은 트랜잭션의 철회 문제와 짧은 시간에 많은 트랜잭션의 완료에 대한 스케줄링이 중요한 관건이 되고 있다.

2.3 낙관적 병행제어 상태

2.3.1 판독 단계(Read Phase)

트랜잭션이 필요한 데이터 아이템은 데이터베이스로부터 판독해서 트랜잭션의 작업 구역에 지역 사본을 만들어 관리하며 트랜잭션의 모든 갱신은 이 사본에 대해서만 실행하고 실제 데이터베이스에 대해서는 실행하지 않는다.

2.3.2 검증 단계(Validation Phase)

판독 단계에서 사본에 반영된 트랜잭션의 실행 결과를 데이터베이스에 그대로 반영할 때 직렬 가능성의 위배 여부를 검사하는 단계로 후진(Backward) 및 전진(Forward) 검증방법이 존재하며[10], 검증 단계는 낙관적 병행제어의 가장 중요한 부분으로 Conflict Operation을 발견하고 이를 해결하는 두 가지 검증방법을 위한 다수의 알고리즘이 있다.

2.3.3 쓰기 단계(Write Phase)

검증 단계를 통과하면 트랜잭션의 실행결과는 실제로 데이터베이스에 반영된다. 그렇지 않으면 트랜잭션은 취소되고 다시 시작해야 한다.

2.4 트랜잭션의 검증(Validation of Transaction)

낙관적 병행제어의 검증단계에는 이미 종료된 트랜잭션들을 대상으로 하는 후진(Backward) 검증방법과 아직 종료되지 않은 실행 중인 트랜잭션들과의 충돌 관계를 파악하는 전진(Forward) 검증 2가지 방식이 존재한다[10].

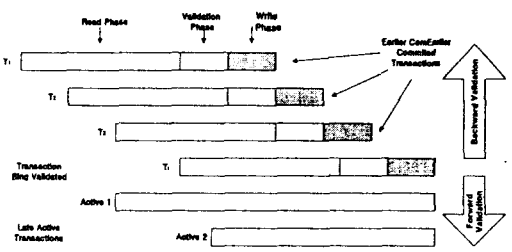


그림 2.1 트랜잭션들의 검증

2.4.1 후진 검증(ackward Validation)

후진 검증 기법에서의 데이터 충돌은 검증하고 있는 트랜잭션(Validation Transaction)의 읽기 세트(Read Set)와 완료된 트랜잭션의 쓰기 세트(Write)를 비교하여 검출한다. 즉 [그림 2.1]에서 검증할 트랜잭션 Tj의 읽기 세트와 이미 수행이 완료된 T1과 T2의 쓰기 세트를 가지고 검사한다. 충돌 해결방법은 이미 완료된 트랜잭션이 직렬화 순서상 검증하고 있는 트랜잭션보다 앞서기 때문에 검증하고 있는 트랜잭션을 재시작 하는 것이다.

2.4.2 전진 검증(Forward Validation)

전진 검증 기법에서는 트랜잭션의 검증이 현재 수행중인 트랜잭션들에 대해 이루어진다. 이것은 검증하고 있는 트랜잭션이 직렬화 순서상 읽기 단계에서 동시 수행중인 모든 트랜잭션들보다 앞선다는 가정에 기초한다. 따라서 데이터 충돌의 검출은 검증하고 있는 트랜잭션의 쓰기 세트와 활동중인 트랜잭션의 읽기 세트를 비교하여 수행된다. 즉 [그림 2.1]에서 검증할 트랜잭션 Tj의 쓰기 세트와 동시에 수행중인 읽기 단계의 트랜잭션 Active1과 Active2의 읽기 세트를 가지고 검사하게 된다. 이러한 경우 데이터 충돌발생시 검증하고 있는 트랜잭션이나 혹은 읽기 단계에서 충돌할 트랜잭션을 재시작 함으로써 해결할 수 있고 이 방법을 사용한 알고리즘들로는 낙관적 취소, 낙관적회생, 낙관적 대기, Wait-50 같은 것들이 있다.

2.5 후진 검증 알고리즘

2.5.1 낙관적 취소(Optimistic Abort)

충돌이 일어난 판독 단계의 트랜잭션들이 검증 단계의 트랜잭션보다 우선 순위가 모두 높다면 검증 단계의 트랜잭션이 즉시 취소된다. 그렇지 않은 경우 판독 단계의 트랜잭션들이 모두 취소되는 두 가지 경우로 처리하는 방식의 알고리즘이다.

```
if CHP transactions in All conflict set then
        restart the validation transaction
else
        restart transactions in conflict set;
        commit the validation transaction;
end;
```

그림 2.2 낙관적 취소 충돌 해결 방법

2.5.2 낙관적 희생(Optimisti Sacrifice)

충돌이 일어난 판독 단계의 트랜잭션들중 하나라도 검증 단계의 트랜잭션보다 우선 순위가 높다면 검증 단계의 트랜잭션이 취소된다. 그렇지 않은 경우에는 판독 단계의 트랜잭션들이 모두 취소되므로 낭비적인 취소가 많이 발생할 수 있는 처리 방식의 알고리즘이다.

```
if CHP transactions in conflict set then
        restart the validation transaction
else
        restart transactions in conflict set;
        commit the validation transaction;
end;
```

그림 2.3 낙관적 희생 충돌 해결방법

2.5.3 낙관적 대기(Optimisti Wait)

충돌이 일어난 판독 단계의 트랜잭션들의 우선순위가 높은 경우 검증 단계의 트랜잭션은 우선순위가 높은 트랜잭션들이 먼저 종료될 수 있도록 기회를 부여하기 위해 대기되는 처리 방식의 알고리즘이다. 이것은 우선 순위가 높은 트랜잭션의 처리를 선호하는 시스템에 적합한 반면 대기되는 트랜잭션은 우선 순위가 더 높은 트랜잭션들에 의해 나중에 취소될 가능성이 많게 된다.

```
while CHP transactions in conflict set do
        wait;
restart transaction in conflict set;
commit the validating transaction
```

그림 2.4 낙관적 대기 충돌 해결방법

2.5.4 Wait-50

낙관적 대기 기법이 문제점을 개선하고자 제안된 것으로 충돌이 일어난 트랜잭션들중 50% 이상이 검증단계에 있는 트랜잭션보다 우선순위가 높으면 검증단계의 트랜잭션이 대기된다. 그렇지 않은 경우 즉 우선순위가 높은 비율이 50% 미만이면 검증 단계의 트랜잭션이 자신의 기록 단계로 전환되며 현재 충돌중인 트랜잭션들을 모두 취소시킨다[11]. Wait-50 기법은 낙관적 대기 기법에 비하여 트랜잭션 처리 시스템의 성능을 향상시킨 알고리즘이다.

```
while CHP transactions in conflict set and HPpercent >- 50 do
        wait;
restart transaction in conflict set;
commit the validating transaction
```

그림 2.5 Wait-50 충돌 해결방법

앞의 낙관적 취소와 회생 방법에서는 충돌이 일어난 트랜잭션들의 낭비적인 취소가 자주 발생함으로써 트랜잭션 처리 시스템의 성능을 저하시키게 된다. 그리고 낙관적 취소와 회생 방법에 비하여 성능이 향상된 낙관적 대기와 Wait-50 방법에서는 검증 단계의 트랜잭션이 대기되는 동안 새로운 충돌이 발생할 수 있다. 이러한 현상은 동시에 실행되는 트랜잭션들의 수가 증가하는 시스템일수록 충돌로 인한 트랜잭션들의 취소량이 증가하게 될 것이므로 결국 트랜잭션 처리 시스템의 성능을 저하시키는 요인이 된다. 검증 단계에서 대기 현상이 발생하는 낙관적 동시실행 제어 프로토콜들은 트랜잭션 처리 시스템의 성능 향상 및 분산 환경에 적용을 위한 개선이 필요하다.

III. 제안 알고리즘

기존에 연구된 알고리즘들이 후진(Backward)검증과 전진(Forward)검증으로 나누어 트랜잭션을 관리하였다는 점에 착안하여 본 논문에서는 Backward와 Forward 방법을 통합하여 중복데이터의 일관성을 유지하는 알고리즘을 설계 및 구현하였다.

3.1 시스템 구성 및 절차

시스템구조는 서버측 시스템으로 Microsoft Window NT 4.0과 IIS를 Web Server로 MS SQL6.5를 DBMS 사용하였고, 클라이언트측 시스템으로 Window 95와 IE 4.0을 브라우저로 ActiveX Control과 ADO를 사용하여 RDS를 구현하여 State Oriented 환경을 구축한 후 제안 알고리즘을 적용하였다.

상태연결(State Oriented) 방법을 사용하여 Web에서 클라이언트와 서버를 연결하므로 써 Web에서 상태의 지속적인 연결이 유지되고, 이에 따른 중복데이터 일관성 문제를 다룬다. 각각의 클라이언트는 브라우저를 통하여 그들이 원하는 데이터를 서버에 요청할 수 있으며, 요청된 데이터는 클라이언트의 캐쉬에 보관된다. 각각의 클라이언트는 캐쉬에 저장된 데이터를 가지고 갱신작업을 수행하며 그들이 필요한 시간에 갱신 검증 요청을 서버에 보낸다. 서버는 각 클라이언트로부터의 검증 요청을 받아서 검증작업을 수행한 후 검증을 요청한 클라이언트와 해당 클라이언트들에게 통보하며, 각 클라이언트는 통보된 내용에 따라 각각의 작업을 수행한다. 여기서 점진적 갱신(Incremental Update)이란 한 클라이언트에서 작업을 완전히 종료하지 않은 상태에서 부분적으로 변경된 부분을 서버측에 통지하여 검증 받는 작업을 수행하는 것을 말한다. 이는 중복된 데이터를 가지고 공동작업을 하는 경우 효과적이다. 그리고 제안한 알고리즘의 시스템 구조도는 [그림 3.1]과 같다.

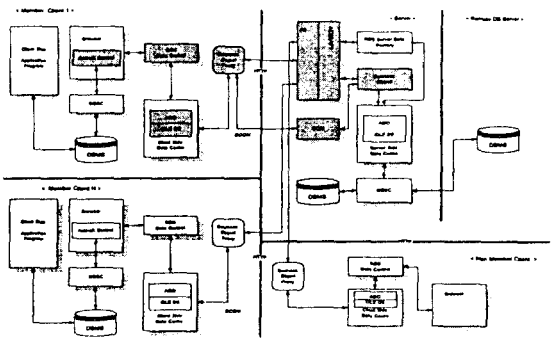


그림 3.1 Web에서 Integrated Database Syetem

본 연구에서는 타임스템프나 우선순위를 사용하지 않고 Row Id를 사용하여 트랜잭션 사이의 충돌을 검사하는 방법으로 사용한다. 따라서 타임스템프 사용할 때 클라이언트/서버 사이에 시간 동기화를 처리할 필요가 없고 우선순위를 사용하지 않기 때문에 작업중인 모든 트랜잭션들의 우선 순위를 검사하지 않아도 된다. 단지 Row Id만 가지고 병행처리를 수행하며, 클라이언트의 데이터 요청시, 현 Server에 있는 데이터와 Row ID를 같이 전송하며, Row Id를 이용하여 병행처리를 수행한다.

3.2 제안 알고리즘의 구성

3.2.1 판독 단계(Read Phase)

각 클라이언트에서 판독요청을 하면 서버는 판독요청을 한 각 클라이언트의 IP Address와 요청 테이블을 기록한 후 해당 클라이언트에서 요청한 데이터를 전송한다.

3.2.2 데이터 갱신(Data Update Phase)

서버로부터 판독요청에 의하여 전송 받은 각 클라이언트 데이터를 가지고 원하는 레코드의 갱신작업을 수행한다. 갱신을 원하는 레코드가 무효화된 상태이면 더 이상 갱신작업을 수행할 수 없다. 이 상태에서 갱신을 원하면 서버에 있는 갱신된 해당 데이터를 재전송 받아야 한다

3.2.3 갱신 검증 단계(Update Validation Phase)

갱신작업을 끝낸 클라이언트가 서버에 검증을 요청하는 단계로, 전송된 Row_ID로 서버의 해당 레코드

의 Row_Id와 비교하여 해당 레코드의 유효성을 판단한다.

검증결과 해당 레코드가 무효화 하다고 검증되면 갱신검증을 요청한 클라이언트에게 해당 레코드 무효화 통보를 보내며 무효화 통보를 받은 클라이언트는 갱신 작업을 취소한다.

3.2.4 쓰기 단계(Write Phase)

검증이 완료된 해당 레코드는 서버측 레코트를 갱신하고 해당 레코드의 Row_ID를 재 할당한다. 서버측 작업이 완료된 후 갱신검증을 요청한 해당 클라이언트에게 Row_ID를 보내고, 또한 서버측에 기록된 Table을 참조하여 갱신된 레코드를 참조한 해당 클라이언트들에게 갱신된 레코드의 Row_ID를 보낸다.

Row_ID를 받은 각 클라이언트들은 각각의 상황에 따라 작업을 수행하는데, 갱신 검증요청 단계에 있는 클라이언트는 갱신된 데이터를 기록하고 Row_Id를 할당하여 작업을 종료하며, 갱신검증 단계가 아닌 다른 단계에 있는 클라이언트들은 클라이언트의 해당 Row_ID와 비교하여 해당 레코드의 유효성을 검증한 후 다르면 무효화 처리를 한다.

3.2.5 제안 알고리즘의 전체 구성도

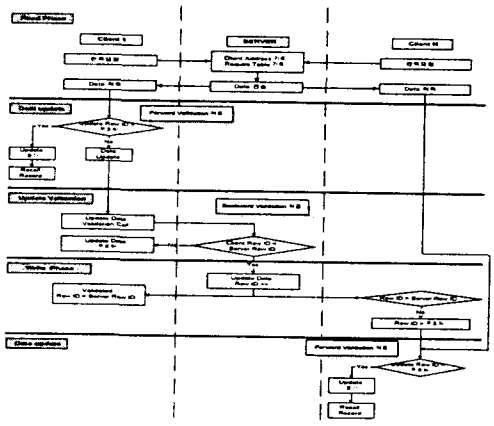


그림 3.1 제안 알고리즘 전체 구성도

3.2.6 알고리즘 구현부

앞에서 제시한 개념들로 프로그램을 구현 후 기존 알고리즘과 제안 알고리즘을 상대적으로 비교하였다.

1) 클라이언트 1에서 갱신작업 수행

기존 알고리즘과 제안 알고리즘은 [그림 3.2] 와 같이 클라이언트 1 과 클라이언트 2로 각각 실행되며 각각 클라이언트 1에서 같은 레코드에 대하여 갱신작업을 수행한다.

주) 프로그램은 브라우저 상에서 Connect oriented 방법을 사용하여 구현되었기 때문에 모두 공유데이터에 대한 일관성이 유지된다.

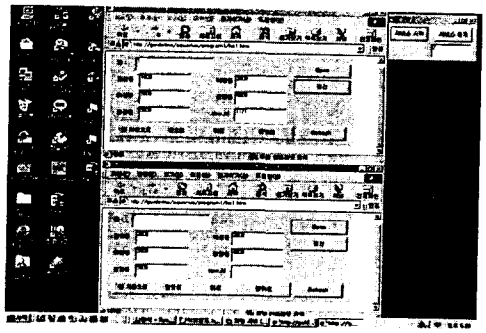
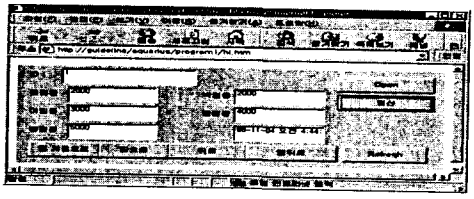
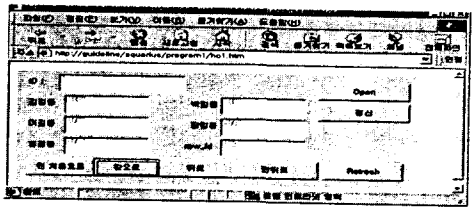


그림 3.2 프로그램 실행

2) 클라이언트 2에서 클라이언트 1의 수정된 데이터에 대한 수정 후 갱신 작업 수행

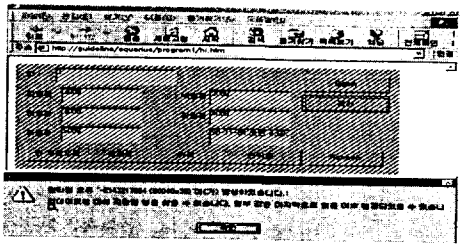


가. 변경된 데이터에 대한 갱신작업 수행 가능

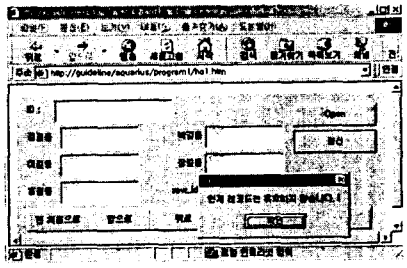


나. 변경된 데이터에 대한 갱신작업 수행 불가 (해당 레코드를 Read Only로 열어 단지 보여주기만 한다.)

3) 클라이언트 2에서 갱신 버튼을 눌렀을 경우오류 메시지를 출력하여, 공유 데이터에 대한 일관성을 유지한다.



가. 갱신 버튼 클릭시 해당 데이터에 대한 검증 작업을 수행하게 된다. 즉 데이터에 대한 검증이 갱신작업 후에 이루어지게 됨으로 무효한 데이터에 대한 갱신이 발생할 수 있고 불필요한 낭비적 작업이 발생한다.



나. 갱신 버튼 클릭시 해당 레코드가 무효함을 메시지로 출력하며 전 단계에서 무효한 레코드에 대한 갱신 작업을 불가능하게 만들었으므로 낭비적인 작업의 수행이 발생하지 않는다.

4. 클라이언트 2는 "확인"을 Click한 후 Reload하거나 다시 개방하면 갱신된 데이터를 받을 수 있다.

V. 결 론

제안 알고리즘은 검증단계에서 충돌연산으로 인한 재시작을 줄이며, 사전에 불필요한 재시작을 방지한다. 그리고 자료 갱신시 다른 클라이언트들에게 변경을 통보하여 무효화 작업을 각 클라이언트에서 수행하여 서버의 부하를 줄인다.

일관성 수준에서 볼 때, 레코드 식별자로 일관성을 관리하기 때문에 병행 효율이 증가하며, 갱신된 데이터를 통보하여 무효화 작업을 수행함으로 충돌연산이 발생 할 수 있는 트랜잭션의 수를 줄인다. 또한 실시간 공유 데이터의 갱신 작업에서 일관성을 유지할 수 있으며, 특정 데이터에 대한 빈번한 갱신작업이 발생하지 않는 한 기존 알고리즘보다 효율이 향상된다.

앞으로 본 논문을 바탕으로 기존 서버의 데이터만이 아닌 웹에서 클라이언트의 개별 데이터베이스와의 연동문제와 이에 따른 공유자료의 중복 일관성유지 문제를 해결하는 알고리즘의 연구가 필요하다.

참 고 문 헌

[1] http://www.infoage.co.kr/internetmag Interent Magazine 96. 4

[2] http://www.utoronto.ca/webdocs/CGI/cgi1.html CGI

[3] http://jserv.javasoft.com/products/java-server The Java Servlet API,

[4] Oracle Web Server, http://www.oracle.com

[5] NSAPI, http://www.webadvisor.com/nsapi.html

[6] ISAPI, http://www.webadvisor.com/isapi.html

[7] http://java.sun.com/productis/java-blend Java blend

[8] ADO and SQL Server Developers Guide, MS document

[9] George Coulouris, Jean Dollimore, Distributed System Concepts and Design, Addison-wesley, 1994

[10] Kung, H. T., & Robinson, J. T., On Optimistic Methods for Concurrency Control. ACM Transaction on Database Systems 2, 213-226, 1981

[11] J. Haritsa, M. Carey and M. Livny, "Dynamic real-time optimistic concurrency control." Proc. 11th IEEE Real-Time Systems Symposium, Lake Buena Vista, Florida, U.S.A., December, pgs. 94--103. 1990

[12] Kwok-wa Lam, Kam-yiu Lam "Optimistic Concurrency Control Protocol for Real-time Databases" J.Systems software 119-131, 1997

[13] J. Robinson, "Design of Concurrency controls for transaction processing systems" Carnegie Mellon University, 1982

[14] S. Greenberg and D. Marwood, "Real-time groupware as a distributed system Concurrency control and its effects on the interface" ACM CSCW' 94 1994

[15] Stefanoceri & Giuseppe Pelagatti "Distributed Databases Principles & Systems"

[16] OLE_DB http://www.oledb.com/