임베디드 DBMS 환경에서의 데이터 동기화: DBMS 독립적인 접근 방안†

장우석*, 강영호*, 노형준*, 정병대*, 손성용*, 김상욱**

1. 서 론

하드웨어 기술과 무선 인터넷 기술의 발전에 따라 PDA(personal digital assistant), 웹패드(webpad), 스마트 폰(smart phone), PPC(pocket PC), HPC (handheld PC)와 같은 유·무선 소형 이동 단말기 (small mobile device)가 널리 퍼지는 포스트 PC 시대가 도래하고 있다. 사용자들은 장소에 관계없이 이동 단말기를 사용하여 다양한 데이터를 보다 편리하고 효율적으로 공유하기 원한다. 이와 같은 데이터 서비스 요구들을 만족시키기 위하여 소형 이동 단말기에 적합한 임베디드 DBMS(embedded DBMS)가 필요하게 되었다.

본 논문에서 사용하는 임베디드 DBMS란 위에서 말한 유·무선 소형 이동 단말기에 탑재되어, 데이터의 저장, 검색, 변경을 수행하는 소형 DBMS를 의미한다[1,2]. 사용자들은 이동 단말기에 탑재된 임베디드 DBMS를 이용하여 필요한 정보를 효과적으로 관리할 수 있다. 현재 포디홈네트(주)에서는 강원대학교 데이터 및 지식공학 연구실과 더불어 정보통신부 선도기술개발사업의 일환으로 인터넷 정보가전용 내장형 DBMS를 개발 중에 있다. 본 논문의 목적은 개발 중 획득한 일부 기술을 동일한 분야의 학자 및 개발자들과 공유하기 위한 것이다.

임베디드 DBMS 환경에서는 이동 단말기들에 저장된 정보를 공유하기 위하여 일반적으로 클라이언트-서버 구조를 이용한다[3-6]. 즉, 워크스테이션 혹은 PC와 같이 자원과 성능이 월등한 서버가 존재하고, 이동 단말기들과 같이 자원과 성능이 비교적 떨어지는 클라이언트들이 존재한다. 클라이언트들은 대부분의 시간 동안 비접속 상태로 있다가 사용자가 원하거나 정해진 시점이 되면 유·무선 통신을 통해 서버와 접속된다. 따라서 클라이언트들은 일반적으로 서버의 데이터베이스에 있는 데이터 중 필요한 일부를 자신들의 로컬 데이터베이스에 다운로드 받아 사용한다.

클라이언트-서버 구조에서 양측의 데이터베이스는 접속되지 않은 상태에서 각각 변경될 수 있으므로 일시적인 불일치성(temporary inconsistency)이 필연적으로 발생하게 된다. 데이터 동기화(data synchronization)란 중복된 데이터 사이에서 발생한 불일치성을 해결하는 수단이며[7], 이는 이동형 임베디드 DBMS를 위한 핵심적인 기능이다. 데이터 동기화의 기본적인 전략은 서버 DBMS와 클라이언트 DBMS에서 각각 수행된 변경을 탐지하여 변경된 내용들을 서로 상대측 데이터베이스에 적용하는 것이다. 임베디드 DBMS

† 본 연구는 대한민국 정보통신부 정보통신연구진흥원의 2000 ~ 2002년 선도기술개발 제 4차 사업의 일환인 인터넷 정보가전용 내장형 DBMS 개발과제(과제번호 : 2002-S-12)의 지원으로 수행되었습니다.

* 포디홈네트(주)
** 강원대학교 컴퓨터정보통신공학부

환경에서는 같은 데이터를 서버와 클라이언트가 서로 다르게 변경하는 경우가 발생하는데 이것을 충돌(conflict)이라 한다[2]. 충돌을 탐지하고 해결하는 기능은 데이터 동기화의 필수적인 기능이다.

동기화 관리자를 개발하는 데 있어 가장 중요한 요구 사항의 하나는 임베디드 DBMS 환경에서 사용하는 다양한 이기종 DBMS와 유연하게 결합하여 사용될 수 있어야 한다는 점이다. 임베디드 DBMS 환경에서는 탑재되는 플랫폼, 사용가능한 리소스, 요구되는 성능, 서비스 용도, 가격 등의 기준에 따라 각각 선정된 다수의 이기종 임베디드 DBMS가 클라이언트 내에 탑재되어 사용된다[1]. 마찬가지로 서버 내에서 사용되는 고성능 서버 DBMS도 사용자의 여건에 따라서 여러 종류가 사용될 수 있다. 다수의 이기종 DBMS가 혼재하는 환경에서 추가적인 부담 없이 동기화를 수행하기 위해서는 DBMS에 독립적으로 수행되는 동기화 관리자의 개발이 필수적이다.

여러 상용 임베디드 DBMS에서 지원하고 있는 동기화 기법들은 참고 문헌[2,7] 등에서 개략적으로 소개되고 있다. 이러한 동기화 기법들에서는 일반적으로 데이터 동기화에 필요한 레코드의 변경 정보를 DBMS 내부에서 자체적으로 저장한다. 이를 위하여 타임스탬프(time- stamp), 트랜잭션 로그(transaction log), 변경 이전 값(before-image) 등이 이용된다. 타임스탬프를 이용하는 방법은 각 레코드마다 타임스탬프 필드를 추가하고 변경이 일어날 때마다 타임스탬프를 갱신하여, 데이터 동기화 시에 타임스탬프의 대소를 비교하여 변경된 레코드를 탐지하는 방법이다. 트랜잭션 로그를 이용하는 방법은 변경 트랜잭션의 내용을 로그로 기록하여, 동기화 시에 로그를 추적하여 변경된 레코드를 탐지하는 방법이다. 변경 이전 값을 이용하는 방법은 변경이 일어난 후에도 변경되기 이전의 값을 저장하고 있다가 동기화 명령이 내려지면 이를 이용하여 변경된 레코드를 탐지하는 방법이다[2,7].

이러한 기존의 방법에서는 레코드 변경시 타임스탬프를 변경하거나, 로그를 기록하거나, 이전 값을 저장하는 등의 기능이 DBMS에 내부적으로 구현되어 있다. 즉, 개발하는 동기화 기능이 특정 DBMS에 종속되며 이 결과, 응용 환경에서 사용하는 DBMS가 바뀌는 경우 기존에 개발된 동기화 기능을 그대로 사용할 수 없다는 문제점이 있다. 본 논문에서는 다양한 이기종 임베디드 DBMS들이 혼재하는 환경에서 발생하는 이러한 심각한 문제를 해결하기 위하여 DBMS에 독립적인 동기화 방법을 제안한다. 제안된 방법에서는 최근 동기화 시의 동기화 대상 테이블의 복사본을 별도로 저장함으로써 이후에 변경되는 레코드에 대한 정보를 파악한다. 제안된 동기화 방법의 가장 큰 특성은 어떠한 DBMS 혹은 임베디드 DBMS가 사용되는 경우에도 제안된 동기화 기법을 그대로 적용할 수 있다는 것이다.

본 논문의 구성은 다음과 같다. 제 2장에서는 본 논문의 연구 배경으로 임베디드 환경에 대해 설명하고, 데이터 동기화의 개념과 필요성 및 데이터 동기화에 대한 기존 연구에 대해 간단히 살펴본다. 제 3장에서는 논문에서 제안하는 동기화 시스템의 구조와 기본 전략 및 제안하는 기법의 장단점에 대해 논의한다. 제 4장에서 제안하는 동기화 기법을 위한 수행 절차를 제시한다. 마지막으로 제 5장에서 결론을 내리고 향후 연구 방향을 제시한다.

2. 연구 배경

본 장에서는 당 연구의 배경으로서 임베디드 DBMS 환경, 동기화의 기본적인 개념, 기존의 임베디드 DBMS에 적용된 동기화 기법들을 소개한다.

2.1 임베디드 DBMS 환경

그림 1은 클라이언트-서버 구조를 기반으로 하는 전형적인 임베디드 DBMS 환경을 나타낸 것이다. 서버는 워크스테이션 혹은 고성능 PC등의 컴퓨터이며 여기에는 고성능 서버 DBMS가 탑재되어 있다. 서버는 항상 네트워크에 접속되어 있다. 클라이언트는 PDA, 웹패드, 스마트 폰 등과 같은 이동 단말기들이며 여기에는 임베디드 DBMS가 탑재된다. 클라이언트는 특성상 네트워크 대역폭이 크지 않으므로 대부분의 시간 동안 비접속 상태를 유지하고, 사용자가 요구하는 순간에 한하여 서버에 접속된다.

소형 이동 단말기에 임베디드 DBMS가 탑재됨으로써 응용 프로그램들은 그들이 담당하는 데이터의 원활한 처리와 공유를 할 수 있다. 물론, 임베디드 DBMS는 해당 장치에 특화되어 있는 데이터를 처리하기 위한 관리 시스템이므로 일반적인 서버 DBMS에서 가능한 모든 기능을 수행하지는 못한다. 임베디드 DBMS는 플랫폼 지원(platform support), 자원 부담(resource load), 성능 평가(performance evaluation), 요구되는 서비스(required services), 가격(price) 등의 선택 기준을

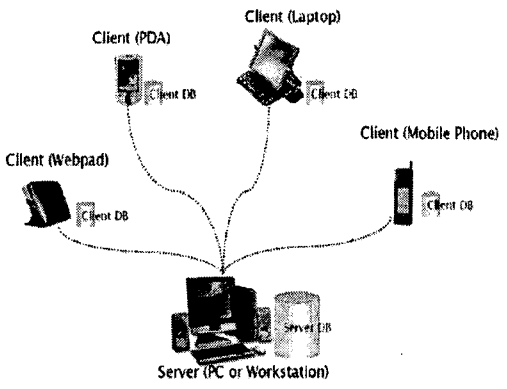


그림 1. 클라이언트-서버 구조를 기반으로 하는 임베디드 DBMS환경

통해 선정된다[1]. 또한, 고성능의 서버 DBMS도 주어진 응용 환경에 적합하도록 임베디드 DBMS에서와 마찬가지 기준을 통해 선정된다. 따라서 임베디드 환경에서는 다수의 이기종 DBMS가 혼재하게 된다.

2.2. 동기화의 개념

서버 DBMS에 저장되어 있는 데이터는 이동 단말기의 클라이언트 DBMS에 다운로드되어 사용되는 경우가 일반적이다. 또한, 클라이언트는 데이터를 공유하기 위하여 클라이언트 DBMS에 있는 데이터를 서버 DBMS로 업로드하여 이를 다른 클라이언트가 다운로드할 수 있게 한다. 이와 같은 데이터 공유 방식을 이용하면 임베디드 장치의 짧은 배터리 용량 등으로 인한 데이터 손실 위험을 방지할 수 있어 데이터 영속성(durability)도 보장할 수 있다. 따라서 임베디드 환경에서는 서버 데이터베이스와 클라이언트 데이터베이스 내에 중복(replication)이 발생한다.

데이터가 중복되어 있고 서로 비접속인 상태에서 데이터에 대한 변경이 일어나면 데이터의 일치성(consistency)은 보장되지 않는다. 데이터 동기화(data synchronization)는 비접속 기간 동안 발생한 변경을 서로 교환하여 일치성을 보장해 주는 수단이다. 동기화 서버(sync server)는 임베디드 DBMS 환경에서 서버와 클라이언트 사이의 데이터 동기화를 담당하는 구성 요소이며, 동기화 에이전트(sync agent)는 데이터에 직접 접속하여 변경을 탐지하고 탐지된 변경을 서버로 전송해 주는 역할을 담당하는 구성 요소이다.

동기화 명령이 내려지면 우선 서버와 클라이언트의 데이터베이스에서 각각 발생한 변경이 탐지되어야 한다. 데이터 변경은 삽입(insert), 갱신(update), 삭제(delete)를 통칭한다. 변경이 탐지

된 레코드는 동기화 서버를 통해 상대방의 데이터베이스에 적용된다. 같은 레코드가 양쪽에서 서로 다르게 갱신되거나, 한 쪽에서 삭제된 데이터를 다른 쪽에서 갱신하거나, 같은 레코드 ID[1]를 가진 레코드가 양쪽에 동시에 삽입되는 경우가 발생하게 되는데, 이를 충돌(conflict)이라 한다[2]. 동기화 서버에서 충돌을 탐지하고 해결하는데, 이를 위하여 충돌 해결 규칙(conflict resolution rule)이 미리 설정되어 있어야 한다. 보통 서버와 클라이언트 중 어느 한 쪽의 변경은 무시하고 다른 한 쪽의 변경을 적용하도록 충돌 해결 규칙을 설정하지만, 특별한 경우 양 쪽 변경의 증감값의 평균을 더하거나 두 데이터를 결합(concatenation)한 값을 적용하도록 충돌 해결 규칙을 설정하기도 한다[4,8].

2.3. 기존의 동기화 기법

Oracle, Sybase, Microsoft, IBM 등과 같은 대부분의 상용 DBMS 벤더들은 고성능의 서버 DBMS와 임베디드 환경에 알맞은 클라이언트 DBMS를 함께 제공한다. 이들은 또한 동기화 서버도 함께 제공하여 서버 DBMS와 클라이언트 DBMS 간의 데이터 동기화를 가능하게 한다. 대표적인 상용 임베디드 DBMS와 동기화 서버 제품인 Oracle 9i Lite와 iConnect, IBM Everyplace와 Sync Server, Sybase Ultralite와 MobiLink & SQL Remote 등에서 사용되는 동기화 기법을 간단히 소개한다.

Oracle의 iConnect는 서버 DBMS인 Oracle Server와 클라이언트 DBMS인 Oracle Lite 사이의 데이터 동기화를 수행하는 시스템으로 두 DBMS에 통합되어 동작한다. iConnect는 사전 이미지(before-image) 테이블과의 비교를 통해 변경을 탐지하며 서버 DBMS에서 충돌을 탐지하고 충돌 시 서버와 클라이언트 중 어느 쪽의 변경을 우선적으로 적용할 것인지에 대한 규칙을 미리 정의함으로써 충돌을 해결한다.

IBM의 Sync Server는 서버 DBMS인 DB2 Universal과 클라이언트 DBMS인 DB2 Everyplace 사이에서 미드-티어(mid-tier) 시스템으로서 동기화를 수행한다. 타임스탬프를 이용한 버전 확인을 통해 변경을 탐지하고, Sync Server에서 충돌을 탐지하며, 서버-우선의 충돌 해결 규칙을 적용한다.

Sybase의 MobiLink와 SQL Remote는 Sybase의 서버 DBMS와 클라이언트 DBMS인 Sybase Adaptive Server Anywhere(Windows CE에서 사용)나 Sybase UltraLite(Palm OS 등에서 사용)와의 동기화에 이용된다. 이들은 변경 트랜잭션을 메시지에 묶어서 서로 교환하는 방법으로 동기화를 수행하며, 미리 충돌 시에 어느 쪽의 변경을 따를 것인지에 대한 규칙을 정의하여 충돌을 해결한다.

현재까지는 이와 같은 동기화 기법들이 논문의 형태로 발표되지 않았기 때문에 외부의 개발자들이 동기화 서버의 내부의 작동 방식에 대하여 자세히 파악할 수 없다. 그러나 변경 탐지 기법이 각 상용 DBMS 벤더들이 제공하는 동기화 서버마다 다르기 때문에 통합적으로 쓰일 수는 없다는 것은 분명하다. 그러나 다양한 이기종의 서버 DBMS와 클라이언트 임베디드 DBMS가 혼재되어 사용되는 환경에서 추가적인 부담 없이 하나의 동기화 서버만을 이용할 수 있도록 하기 위하여 동기화 서버의 DBMS에 대한 독립은 필수적이다. 이를 위하여 본 논문에서는 DBMS의 공통적인 특징을 이용하여 DBMS의 종류에 관계없이 수행 가능한 동기화 서버 개발 방법을 제안한다.

1) 보통 기본 키(primary key)를 이용한다.

3. 제안하는 동기화 기법의 기본 전략

본 장에서는 본 논문에서 제안하는 임베디드 DBMS 환경에서의 DBMS 독립적인 동기화 기법을 위한 전체 아키텍처와 기본 전략을 제시한다.

3.1. 동기화를 위한 전체 아키텍처

그림 2는 제안하는 동기화 기법을 위한 전체 아키텍처를 표현한 것이다.

동기화는 하나의 서버 DBMS와 다수의 클라이언트 DBMS들을 대상으로 한다. 동기화를 위해 동기화 서버와 동기화 에이전트가 존재한다. 동기화 서버[2]는 서버 DBMS와 클라이언트 DBMS를 연결하고, 변경된 데이터를 받아서 충돌을 탐지하고 해결한 후, 상대편 DBMS의 데이터베이스에 그 변경을 반영하는 역할을 한다. 동기화 에이전트는 서버 DBMS와 클라이언트 DBMS에 각각 존재하며, 동기화 대상이 되는 테이블을 검색하여 변경을 탐지하고 동기화에 필요한 정보인 연결 정보, 인증 정보, 레코드 필터링 조건 등을 관리한다. 동기화 서버와 에이전트는 자신들의 역할을 수행하기 위해 필요한 데이터를 저장하기 위해 별도의 데이터베이스를 이용할 수 있는데 이를 동기화 데이터베이스(sync DB)라고 정의한다.

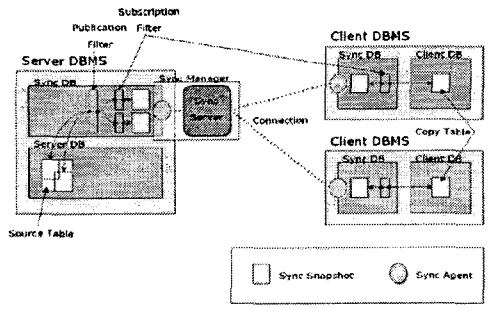


그림 2. 동기화를 위한 전체 아키텍처

원본 테이블(source table)은 서버 측에 존재하는 동기화의 대상 테이블을 의미하고, 사본 테이블(copy table)은 클라이언트 측에 존재하는 동기화의 대상 테이블을 의미한다. 동기화가 수행되기 위해서는 우선 서버 DBMS의 원본 테이블이 지정되어야 한다. 일반적으로 동기화는 발행-인용(publish- subscribe) 방식으로 수행되는데, 서버의 원본 테이블에서 발행(publish)할 영역에 대한 조건과 함께 클라이언트의 사본 테이블에서 인용(subscribe) 할 영역에 대한 조건을 설정한다. 이 두 조건을 동시에 만족하는 내용이 최종 동기화 대상이 된다. 이와 같이 조건을 설정하여 동기화에 필요한 영역을 지정하는 것을 필터링(filtering)이라고 한다[4,5].

서버와 클라이언트 각각의 동기화 에이전트는 원본 테이블과 사본 테이블의 최근 동기화 시의 내용을 별도로 저장해 둔다. 별도로 저장된 최근 동기화 시의 테이블 내용을 본 논문에서는 스냅샷(snapshot)[3]이라고 정의한다. 동기화 명령이 내려지면 우선 변경된 레코드를 탐지해야 하는데, 이 때 각 DBMS에서 따로 레코드의 변경을 기록해 두지 않고도 스냅샷과의 비교를 통해 변경된 레코드를 찾아낼 수 있다[4]. 스냅샷을 이용한 변경

2) 동기화 서버와 서버 데이터베이스를 혼동해서는 안 된다. 서버 데이터베이스는 동기화 할 원본 데이터가 들어 있는 데이터베이스를 말하고, 동기화 서버는 실제로 동기화 작업이 이루어지는 곳을 말한다. 동기화 서버는 서버 데이터베이스와 떨어져서 동기화 작업을 수행할 수도 있다.

3) 동기화를 소개한 여러 문헌에서 여러 가지 의미로 스냅샷이라는 용어를 사용했다. 예를 들어 오라클[3]에서는 클라이언트에 복제된 원본 테이블의 복제본을 가리켜 스냅샷이라고 하고, 사이베이스[6]에서는 동기화 대상이 되는 테이블 전체를 클라이언트로 복제하는 것을 스냅샷 방식이라고 한다. 일반적으로 스냅샷이라는 용어는 특정한 시각의 테이블의 상태를 말한다.

4) 참고 문헌 [7]에서 Oracle의 동기화 모듈인 iConnect가 사전 이미지(before-image) 테이블을 이용하여 변경을 탐지한다는 것에 대해 언급하고 있다. iConnect의 변경 탐지 방식은 이전 동기화 시의 스냅샷 테이블을 이용하는 본 논문에서 제시한 변경 탐지 방법과 비슷하다. 그러나 iConnect의 방식은 DBMS (Oracle Lite) 내부에서 동기화를 위해 지원하는 기능으로, 본 논문에서 주장하는 바와 같이 DBMS와 동기화 모듈의 독립성을 위한 동기화 모듈의 기능이 아니라는 점에서 다소 차이가 있다.

의 탐지가 바로 데이터 동기화가 DBMS에 독립적으로 동작할 수 있도록 하는 핵심적인 개념이다.

3.2. 기본 전략

클라이언트 측의 동기화 에이전트는 필터링 조건과 함께 서버 데이터베이스로의 연결 정보인 네트워크 주소, 데이터베이스 이름 및 인증 정보 등을 함께 관리한다. 이들 정보는 크지 않으므로 동기화가 실행될 때 클라이언트 자신의 연결 정보와 함께 동기화 서버로 전송된다. 물론 동기화 서버가 클라이언트마다 필터링 조건 정보와 연결 정보를 직접 관리할 수도 있지만, 동기화 서버의 부담을 줄이기 위해 클라이언트가 늘어남에 따라 가변적으로 변하는 이러한 정보들을 클라이언트 각자가 저장하는 것이 더 합리적이다.

필터링을 통해 원하는 데이터가 서버에서 클라이언트로 복제(replication)되면, 복제된 당시의 스냅샷이 동기화 에이전트가 관리하는 데이터베이스에 저장된다. 스냅샷은 서버와 클라이언트에서 동기화의 대상이 되는 원본/사본 테이블과 대응되어 최근 동기화 이후에 발생한 변경을 탐지하는 데 이용된다. 동기화 서버를 통해 동기화 명령이 서버 데이터베이스와 클라이언트 데이터베이스를 관리하는 동기화 에이전트에 내려지면, 각각의 에이전트는 최근 동기화 된 시점부터 당시까지 원본/사본 테이블에 어떤 레코드가 삽입, 갱신, 삭제되었는지를 스냅샷과의 비교를 통해 탐색한다. 스냅샷에는 존재하지 않고 원본/사본 테이블에만 존재하는 레코드는 새로 삽입된 레코드이고, 스냅샷에만 존재하고 원본/사본 테이블에는 존재하지 않는 레코드는 삭제된 레코드이며, 스냅샷과 원본/사본 테이블 양측에 존재하면서 그 내용이 달라진 레코드는 갱신된 레코드이다. 물론 양측에 존재하고 내용이 달라지지 않은 레코드는 변경이 일어나지 않은 레코드이다.

삽입된 레코드, 갱신된 레코드, 삭제된 레코드는 동기화 에이전트에서 검색되어 각각 임시 테이블에 저장된다. 즉, 삽입 테이블, 갱신 테이블, 삭제 테이블이 만들어진다. 이들을 통칭하여 변경 테이블이라 하는데, 변경 테이블은 동기화 에이전트가 관리하는 동기화 데이터베이스에 저장된다. 참고로 삭제 테이블의 경우에는 기본 키를 제외한 필드들은 의미가 없으므로 기본 키만으로 테이블을 구성할 수 있다.

서버와 클라이언트 양측에서 변경 탐지가 끝나면 변경된 레코드들로 이루어진 변경 테이블들이 동기화 서버로 전송된다. 동기화 서버에서는 충돌을 탐지하는데, 충돌이란 양쪽에서 동일한 레코드 ID(또는 기본 키)를 가진 레코드에 대해 서로 다른 변경을 했을 경우에 발생한다. 충돌은 네 가지[5] 종류로 나눌 수 있다. 양쪽에서 서로 다른 값으로 갱신한 경우를 갱신-갱신 충돌(update-update conflict)이라 하고, 서버에서 갱신된 레코드를 클라이언트에서는 삭제한 경우를 갱신-삭제 충돌(update-delete conflict)이라 하고, 서버에서 삭제된 레코드를 클라이언트에서는 갱신한 경우를 삭제-갱신 충돌(delete-update conflict)이라 하고, 양쪽에서 기본 키가 같은 서로 다른 레코드를 삽입한 경우를 삽입-삽입 충돌(insert-insert conflict) 이라 한다[2,7].

동기화 서버에서 충돌을 해결하기 위해 여러 가지 규칙을 설정해야 한다. 충돌이 일어난 레코드에 대하여 양측의 변경을 동시에 반영하기가 불가능하므로 클라이언트 측의 변경을 무시하고 서버 측의 변경을 따르거나 서버 측의 변경을 무

5) 참고 문헌 [2]과 [7]에서는 충돌을 삽입 충돌, 갱신 충돌, 삭제 충돌의 세 가지로 분류했다. 본 논문에서 말하는 갱신-삭제 충돌과 삭제-갱신 충돌을 함께 묶어서 삭제 충돌로 분류했다. 충돌의 개념은 비슷하지만 본 논문에서는 충돌 탐지 방법을 구분해서 설명하기 위해 갱신-삭제 충돌과 삭제-갱신 충돌을 따로 분류했다.

시하고 클라이언트 측의 변경을 따르는 것 중에 하나를 보통 선택한다. 그러나 특별한 경우 양쪽에서 일어난 변경이 동시에 반영되는 경우도 발생한다. 숫자 데이터에서 양쪽의 증감치를 합하여 갱신하기도 하고, 문자 데이터의 경우 두 데이터를 결합하여 갱신하기도 하는 특별한 옵션을 선택할 수도 있다[4,8]. 충돌이 발생한 모든 레코드들에 대하여 동기화를 요청한 사용자에게 어떤 변경을 적용할 것인지 선택하도록 하는 방법도 가능하다. 이와 같이 충돌이 해결된 변경 레코드는 각각 상대측 데이터베이스에 적용된다. 이상 없이 적용이 완료되면 테이블의 내용과 동일하게 스냅샷을 갱신하고 동기화 작업을 마친다.

동기화 작업에 의해 삽입, 갱신, 삭제되는 동작들은 하나의 트랜잭션으로 간주하여 실행되어야 한다. 따라서 동기화 작업 진행 중에 시스템 장애나 오류가 생기는 경우에는 서버나 클라이언트에 동기화의 대상이 되는 데이터 중 일부만 반영되고 일부는 반영되지 않기 때문에 데이터의 일관성에 문제가 발생하게 된다. 이러한 문제가 발생하는 경우에는 트랜잭션 롤백을 통해 동기화 이전 상태로 복구시킴으로써 데이터베이스를 안전하게 보호할 수 있다. 시스템 장애나 오류가 해결되면 사용자는 다시 동기화 명령을 내려 데이터를 동기화 시킬 수 있다.

본 논문에서 제안한 동기화 기법의 가장 큰 장점은 동기화 기능이 DBMS의 어플리케이션으로 동작하므로 DBMS의 수행 방식에 영향을 주지 않고 독립적으로 수행될 수 있다는 것이다. 모든 DBMS가 공통적으로 지원하고 있는 API인 SQL, JDBC, ODBC등을 이용함으로써 사용자가 지정한 데이터베이스와 테이블 구조에 수정을 가하지 않고 이기종 DBMS와 연동이 가능하도록 하였다.따라서 본 논문에서 제시된 DBMS 독립적인 기법을 이용하면 유연성(flexibility)과 상호 운용성(interoperability)을 가진 동기화 시스템을 제작할 수 있다. 단, 제안하는 방법은 스냅샷을 저장하기 위한 별도의 저장 공간이 필요하므로 다소의 저장 공간 부담[6]을 감수해야 한다. 그러나 임베디드 DBMS 환경에서 다양한 이기종의 DBMS가 사용된다는 점을 고려하면, DBMS 독립적인 방식이라는 장점을 수용하기 위하여 이러한 저장 공간의 부담은 감수할 만한 것이라고 판단된다.

4. 동기화 절차

본 장에서는 제안하는 기법을 위한 동기화의 절차에 관하여 보다 구체적으로 설명한다. 동기화 절차는 크게 동기화 설정과 동기화 실행으로 구분된다. 동기화 설정은 동기화에 필요한 다양한 정보들을 사전에 설정하는 것을 의미한다. 즉, 동기화 설정은 동기화 실행을 위한 준비 과정이라 할 수 있다. 동기화 실행은 동기화 서버가 동기화를 요청하여 변경된 데이터가 서버와 클라이언트간에 상호 교환됨으로써 양측의 데이터가 동일하게 되는 것을 의미한다.

4.1. 동기화의 설정

알고리즘 1은 동기화 설정 과정을 순서대로 나타낸 것이다.

단계 1의 네트워크 주소는 클라이언트와 서버가 통신하기 위한 기본 정보로 사용된다. 서버는 자신의 클라이언트의 주소를 모두 저장하여 관리해야 하고, 클라이언트는 서버의 주소만 저장하여

6) 스냅샷의 크기는 클라이언트에 복제된 사본 테이블과 비슷하다. 클라이언트는 대상 테이블에 필요한 저장공간 만큼씩의 공간이 더 필요하고, 서버는 클라이언트의 수 만큼의 스냅샷을 저장하기 위한 공간이 필요하게 된다. 만일 서버나 클라이언트가 읽기 전용(read-only)이라면 저장 공간의 부담은 더 줄어들 수 있다. 읽기 전용인 테이블은 변경이 발생하지 않기 때문에 변경 탐지를 위한 스냅샷이 필요하지 않다.

1. 서버와 클라이언트의 네트워크 주소를 각각 설정한다.
2. 동기화 대상이 되는 데이터베이스와 테이블의 이름을 각각 설정한다.
3. 동기화 서버와 에이전트에서는 데이터베이스에 접속하기 위한 인증 정보를 설정한다.
4. 서버는 발행 필터링 조건을 설정하고 클라이언트는 인용 필터링 조건을 지정한다.
5. 충돌 해결 규칙을 설정한다.
6. 서버의 동기화 대상 데이터를 클라이언트로 다운로드한다.
7. 양측의 동기화 에이전트는 스냅샷을 복사하여 각각의 동기화 데이터베이스에 저장한다.

알고리즘 1. 동기화 설정 과정

관리한다. 이들 네트워크 정보는 동기화 에이전트에 의하여 관리되며, 동기화가 시작되면 동기화 서버로 전송된다. 단계 2에서는 원본 테이블의 이름과 원본 테이블이 포함된 데이터베이스의 이름, 사본 테이블의 이름과 사본 테이블이 포함된 데이터베이스의 이름이 지정된다. 이들 이름도 네트워크 주소와 같이 동기화 에이전트에서 관리된다.

단계 3의 인증 정보는 지정된 데이터베이스와 테이블에 동기화 에이전트가 접근 가능하게 하기 위한 것이다. 동기화 에이전트는 서버 DBMS와 클라이언트 DBMS에서 동기화의 대상으로 지정된 데이터베이스와 테이블에 대한 읽기-쓰기 권한을 가지고 있어야 한다. 인증 정보 역시 각 동기화 에이전트에서 관리된다. 단계 4에서는 동기화의 대상이 되는 서버의 원본 테이블에서 필요한 발행 필터링 조건과 클라이언트의 사본 테이블에서 필요한 인용 필터링 조건이 지정된다. 각 필터링 조건들은 행 필터링(row filtering)과 열 필터링(column filtering)으로 이루어져 있다. 서버의 동기화 에이전트는 발행 필터링 조건을 관리하고 클라이언트의 동기화 에이전트는 발행 필터링 조건과 인용 필터링 조건을 모두 관리한다.

단계 5의 충돌 해결 규칙은 각 클라이언트마다 설정된다. 충돌 해결 규칙은 클라이언트의 사정에 따라 변경될 수 있으므로 동기화 에이전트에서 관리하다가 동기화가 실행될 때 동기화 서버로 전송되어 충돌 해결에 이용된다. 필요한 모든 데이터가 설정되면 단계 6에서 데이터가 서버에서 클라이언트로 전송되어 복제된다. 이 작업을 통해 동기화를 위한 최초의 복제가 이루어진다. 단계 7에서 서버와 클라이언트의 동기화 에이전트는 원본 테이블과 사본 테이블의 스냅샷을 복사하여 동기화에 필요한 데이터가 저장되는 동기화 데이터베이스에 저장한다. 저장된 스냅샷은 다음 동기화 시의 변경 탐지에 이용된다.

4.2. 동기화의 실행

동기화의 실행은 서버나 클라이언트 둘 중 어느 곳의 동기화 에이전트를 통해서든 동기화 서버로 요청될 수 있다. 동기화가 요청되면 아래의 알고리즘 2가 실행된다. 알고리즘 2는 동기화 실행 과정을 순서대로 나타낸 것이다.

단계 1에서는 동기화 서버가 동기화 명령을 아직 받지 못한 상대편의 동기화 에이전트로 동기화 명령을 전송한다. 단계 2에서는 동기화 에이전트가 스냅샷 테이블과 동기화를 수행하고자 하는 테이블을 비교하여 변경을 탐지한다. 이들 변경된

1. 동기화 서버는 동기화 명령을 상대측 동기화 에이전트로 전송한다.
2. 각 동기화 에이전트가 스냅샷과 현재 테이블을 비교함으로써 변경 탐지를 수행한다.
3. 동기화 서버는 서버와 클라이언트의 변경 테이블을 비교하여 충돌을 검사하고, 탐지된 충돌을 정해진 규칙에 따라 해결한다.
4. 동기화 서버는 충돌이 해결된 변경 내용을 상대측 데이터베이스에 반영한다.
5. 동기화 에이전트는 동기화 데이터베이스의 스냅샷에 변경된 내용을 반영한다.

알고리즘 2. 동기화 실행 과정

레코드는 동기화 에이전트에 의해 임시로 만들어진 변경 테이블에 저장되는데, 이를 위해 동기화 데이터베이스가 이용된다.

단계 3에서 변경 테이블끼리 서로 비교하여 충돌을 탐지한다. 삽입-삽입 충돌은 서버와 클라이언트 양측의 삽입 테이블의 기본 키를 비교하여 서로 같은 기본 키를 가진 레코드가 존재하는지 검사하여 찾는다. 삭제-갱신 충돌은 서버 측의 삭제 테이블과 클라이언트 측의 갱신 테이블의 기본 키를 비교하여 같은 레코드가 양 테이블에 존재하는지 검사하여 찾는다. 갱신-삭제 충돌도 마찬가지로 서버 측 갱신 테이블과 클라이언트 측 삭제 테이블의 기본 키를 비교하여 같은 레코드가 양 테이블에 존재하는지 검사하여 찾는다. 갱신-갱신 충돌도 마찬가지로 양측의 갱신 테이블의 기본 키를 비교하여 같은 기본 키가 있는지를 검사해서 찾는다.

발견된 충돌을 해결하기 위해 충돌 해결 서버와 클라이언트 양측의 임시 변경 테이블을 변경한다. 충돌이 일어난 레코드에 대하여 서버 측의 변경에 따르는 방식에서는 클라이언트 측 변경 테이블의 충돌된 레코드를 삭제한다. 마찬가지로 클라이언트 측의 변경을 따르는 방식에서는 서버 측 변경 테이블의 충돌된 레코드를 삭제한다. 증감값을 이용하거나 두 값이 결합된 값을 이용하여 충돌을 해결할 경우[4,9]에는 양 측 변경 테이블의 충돌된 레코드들을 함께 갱신한다.

단계 4에서 충돌이 해결된 변경 테이블을 서로 상대방의 원본/사본 테이블에 반영한다. 삽입 테이블에 있는 레코드들은 원본/사본 테이블에 삽입하고, 갱신 테이블에 있는 레코드들은 원본/사본 테이블에서 삭제하고, 삭제 테이블에 있는 레코드들도 원본/사본 테이블에서 삭제한다. 반영이 끝나면 서버 데이터베이스의 원본 테이블과 클라이언트 데이터베이스의 사본 테이블은 동일해진다. 단계 5에서는 동기화 에이전트는 마지막으로 양측의 스냅샷을 갱신한다. 갱신된 스냅샷은 다음 동기화가 실행될 때 변경 탐지에 이용된다.

4. 결 론

임베디드 DBMS는 소형 이동 단말기에 탑재되어 데이터의 저장, 검색, 변경을 수행함으로써 응용 프로그램이 보다 편리하고 효율적으로 정보를 이용할 수 있도록 한다. 이러한 임베디드 DBMS는 일반적으로 고성능, 대용량의 서버 DBMS에 클라이언트로서 연결되어 데이터를 다운로드 받아 사용한다. 이와 같은 임베디드 환경에서는 대부분의 경우 클라이언트와 서버가 접속되지 않은 상태에서 데이터 변경이 이루어진다. 따라서 서버와 클라이언트간에는 일시적인 데이터 불일치성이 필연적으로 발생한다. 데이터 동기화란 서버와 클라이언트에서 비접속 기간동안 각각 발생한 변경 내용들을 서로 비교·교환하여 데이터 불일치성을 해결하는 작업을 말한다.

임베디드 DBMS 환경에서는 다양한 이기종 DBMS가 사용된다. 그러나 현재 DBMS 벤더들마다 별도의 동기화 방식을 이용하기 때문에 이기종 DBMS간의 동기화에는 추가적인 부담이 필요하다. 본 논문에서는 이러한 추가적인 부담 없이 동기화를 수행하도록 하기 위하여 DBMS 독립적으로 수행되는 동기화 방법을 제안하였다. 본 논문에서 제안한 방법은 동기화 당시의 테이블의 복사본, 즉 스냅샷을 동기화를 위해 별도로 저장하는 것이다. 스냅샷을 이용하면 동기화가 DBMS의 응용 프로그램처럼 동작하도록 할 수 있기 때문에 DBMS의 내부 구조에 변화를 주지 않고 동기화 서버를 구현할 수 있다. 스냅샷을 이용하는 방법의 단점인 클라이언트 측의 저장 공간 부하는 장점인 DBMS에 독립성에 비하여 감수할 수 있

는 수준으로 판단된다. DBMS에 독립적인 특징은 임베디드 DBMS와 동기화 서버를 이용한 응용 프로그램이 유연성과 상호 운용성을 가질 수 있도록 해 준다.

제안된 방법은 클라이언트가 저장공간에 대해 심각한 제약이 없는 경우 DBMS에 대한 독립성을 잘 보장하고 있다. 향후 연구 방향으로는 DBMS 독립성을 보장하면서 저장 공간 부하를 줄여 나가는 방법을 고안하는 것을 고려하고 있다.

참 고 문 헌

[1] Michael A. Olson, "Selecting and Implementing an Embedded Database System," IEEE Computer, Vol. 33, No. 9, pp 27-34, September 2000.

[2] 김상욱, 오세봉, 김만순, 박정일, 상용 내장형 DBMS 환경을 위한 Synchronization 기법에 관한 연구, 정보통신 선도기반 기술개발사업 인터넷 정보가전용 임베디드 DBMS(과제번호 : 2000-S-169) 기술보고서, 2001년 10월.

[3] Oracle, Oracle 9i Lite Administration Guide 5.0.1, A White Paper.

[4] Pointbase, Pointbase UniSync Developer's Guide, Version 4.2, A White Paper.

[5] IBM, DB2 Sync Server Administration Guide 7.2.1, A White Paper.

[6] Sybase, Synchronization Technologies for Mobile and Embedded Computing, A White Paper.

[7] 이상윤, 박순영, 이미영, 김명준, "이동 DBMS의 데이터 동기화 기술 분석," 데이터베이스 연구회지, 17권 3호, pp. 29-41, 2001년 9월.

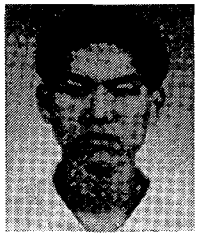
[8] Jim Gray, Pat Helland, Patrik O'Neil, Dennis Shasha, "The Dangers of Replication and a Solution," In *Proc. Int'l. Conf. on Management of Data*, ACM SIGMOD, pp. 173-182, 1996.

장 우 석

• 2001년 2월 : 포항공과대학교 컴퓨터공학과 졸업(학사)
• 2001년 2월~현재 : 포디홈네트(주) 부설연구소 연구원
• 관심분야 : DBMS, 임베디드 DBMS, 홈 네트워킹
• E-mail : linus@4dhome.net

강 영 호

• 2002년 2월 : 숭실대학교 컴퓨터학부 졸업(학사)
• 2002년 1월~현재 : 포디홈네트(주) 부설연구소 연구원
• 관심분야 : DBMS, 임베디드 DBMS, 홈 네트워킹
• E-mail : tristan@4dhome.net

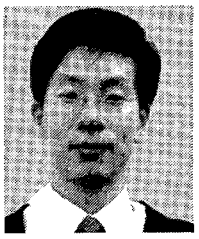


노 형 준

- 2000년 2월 : 포항공과대학교 컴퓨터공학과 졸업(학사)
- 2000년~2001년 04월 (주)포시에스 웹 솔루션 개발팀
- 2001년 4월~현재 현재 포디홈네트(주) 부설연구소 전임 연구원
- 관심분야 : DBMS, 임베디드 DBMS, 홈 네트워킹
- E-mail : urmine@4dhome.net

정 병 대

- 1999년 2월 : 포항공과대학교 컴퓨터공학과 졸업(학사)
- 2001년 2월 : 포항공과대학교 컴퓨터공학과 졸업(석사)
- 2001년 2월~2001년 10월 : 포디홈네트(주) 기업부설연구소 연구원
- 2001년 10월~현재 : 포디홈네트(주) 부설연구소 DBMS 팀 팀장
- 관심분야 : DBMS, 임베디드 DBMS, 홈 네트워킹
- E-mail : nicolas@4dhome.net



손 성 용

- 1990년 : 한국과학기술원 졸업(학사)
- 1992년 : 한국과학기술원 졸업(석사)
- 2000년 : Univ. of Michigan 졸업(공학박사)
- 1992년 1월~1995년 5월 : LG Software 근무
- 1995년 11월~1996년 8월 : 생산기술연구원 근무
- 2000년~현재 : 현재 포디홈네트(주) 부설연구소 연구소장
- 관심분야 : 홈 네트워킹, DBMS
- E-mail : xtra@4dhome.net

김 상 욱

- 1989년 2월 : 서울대학교 컴퓨터공학과 졸업(학사)
- 1991년 2월 : 한국과학기술원 전산학과 졸업(석사)
- 1994년 2월 : 한국과학기술원 전산학과 졸업(박사)
- 1991년 7월~8월 : 미국 Stanford University, Computer Science Department, Summer Student
- 1994년 2월~1995년 2월 : KAIST 정보전자연구소 위촉 연구원
- 1999년 8월~2000년 8월 : 미국 IBM T.J. Watson Research Center, Post-Doc.
- 1995년 3월~현재 : 강원대학교 컴퓨터정보통신공학부 부교수
- 관심분야 : DBMS, 저장 시스템, 주기억장치 DBMS, 임베디드 DBMS, 데이터 마이닝
- E-mail : wook@kangwon.ac.kr