메시지 다이제스트를 이용한 모바일 데이타베이스 동기화 알고리즘

(A Synchronization Algorithm for Mobile Database using Message Digest)

문 창 주 † 최 미 영 †† 김 상 민 ††† 정 진 환 ††††
(Chang-Joo Moon) (Mi-Young Choi) (Sang-Min Kim) (Jin-Hwan Jung)

요 약 본 논문에서는 데이타베이스 서버와 모바일 데이타베이스 사이의 데이타 동기화를 위하여 메시지 다이제스트 기반의 SAMD 알고리즘을 제안한다. SAMD 알고리즘은 메시지 다이제스트 테이블을 이용하여 데이타베이스 서버 측과 모바일 데이타베이스 측에 이미지를 만들고 이 두 개의 이미지를 비교하여 동기화가 필요한 로우들을 선별한다. 두 이미지의 값이 다르다면 동기화 정책에 따라 동기화를 진행한다. SAMD 알고리즘은 특정 데이타베이스 밴더에 종속된 기술이나, 트리거, 스토어 프로시져, 타임스탬프등을 사용하지 않고 관계형 데이타베이스의 기본적인 SQL 기능만을 사용한다. 따라서 특정 밴더의 데이타베이스에 종속적이지 않음으로 어떠한 데이타베이스 서버와 모바일 데이타베이스 서버의 조합에도 사용 가능 하다. 또한 성능면에서도 기존의 제품에 비해서 우수한 성능을 나타낸다.

키워드 : 동기화, 모바일 데이타베이스, 메시지 다이제스트

Abstract This paper proposes the SAMD(Synchronization Algorithms based on Message Digest) algorithm based on message digest in order to data synchronization between a database server and a mobile database. The SAMD algorithm makes the images at the database server and the mobile database using message digest tables and compares two images in order to select the rows needed to synchronization. If two images are different, the synchronization is progressed according to synchronization policy. The SAMD algorithm does not use techniques which are dependent the some database vendor and also triggers, stored procedures and timestamps. The SAMD only uses the standard SQL functions for the synchronization. Therefore the SAMD algorithm is used in any combinations of the database server and the mobile database because the SAMD algorithm is not depended on some database venders. Also the SAMD algorithm shows better performance compared with existing synchronization products.

Key words : Synchronization, Mobile Database, Message Digest

1. 서 론

최근 들어 모바일 통신 기술과 장비들의 발전으로 모바일 통신 분야가 새로운 컴퓨팅 환경으로 등장 하였으며 PDA(personal digital assistant), Mobile phone, HPC(hand-held PC), Pocket PC 등의 소형 모바일 단말기 빠른 속도로 보급되고 있다. 다양한 모바일 단말기들과 무선 인터넷 기술이 결합되면서 모바일 단말기를 통한 정보 처리가 가능 하게 되었고 다양한 비즈니스 모델들이 등장하고 있다[1].

모바일 단말기들은 컴퓨팅 파워가 약하고 배터리에 의존적이다. 또한 통신 대역폭도 유선에 비해 좁기 때문에 지속적인 통신 접근이 어렵다[2,3]. 따라서 이들은 대용량의 데이타를 처리하거나 데이타베이스 서버와 지속적인 연결을 유지하기 힘들다. 이러한 이유 때문에 모바일 단말기들은 안정적인 데이타 처리를 위하여 모바일 데이타베이스를 가지고 있다. 모바일 단말기들은 온라인

· 본 연구는 정보통신부 및 정보통신연구진흥원의 대학 IT연구센터 육성 지원사업의 연구결과로 수행되었음
† 정 회 원 : 건국대학교 항공우주정보시스템 교수
cjmoon@konkuk.ac.kr
†† 비 회 원 : 고려대학교 컴퓨터학과
michelle@paran.com
††† 비 회 원 : 건국대학교 컴퓨터시스템전공
sooniettl@nate.com
†††† 비 회 원 : 건국대학교 컴퓨터소프트웨어전공
love_jjh@hanmail.net
논문접수 : 2006년 11월 1일
심사완료 : 2007년 4월 6일

(on-line) 상태에서 데이타베이스 서버로부터 제한된 데이타의 복사본을 내려 받는다. 그 다음 오프라인(off-line) 상태에서 내려 받은 데이타들을 이용하여 다양한 업무를 처리한다. 이러한 네트워크가 단절된 상태에서의 작업은 이동성 지원을 위한 중요 요소이다[4]. 네트워크가 단절된 상태에서의 작업은 필연적으로 데이타베이스 서버와 모바일 데이타베이스 사이에 데이타 불일치를 발생시킨다. 따라서 모바일 컴퓨팅 환경에서 데이타 불일치를 해결하고 데이타의 무결성을 보장하는 동기화는 반드시 해결해야 하는 중요한 주제이다[5].

기존의 데이타베이스 밴더들은 모바일 환경에서 데이타 동기화를 위한 다양한 솔루션들을 제공하고 있다. 이들 솔루션들은 Sybase의 MobiLink & SQL Remote[6], IBM의 Sync Server[7], Informix의 Cloudsync[8], Oracle의 iConnect[9], PointBase의 UniSync 등이다. 그러나 이들 솔루션들은 서버 측 데이타베이스의 메타데이타와 같은 데이타베이스 의존적인 정보를 사용하거나 서버 측 데이타베이스가 제공하는 특정 기능을 사용함으로 서버 측 데이타베이스에 독립적이지 못하다. 즉 동기화를 데이타베이스 서버가 담당하며 모바일 데이타베이스는 반드시 서버 측 데이타베이스와 동일한 밴더의 제품이어야 한다. 미들티어에 별도의 동기화 서버를 운영하는 솔루션은 서버 측 데이타베이스에 독립적이기는 하지만 모바일 데이타베이스에 종속적이다. 즉 반드시 동기화 서버와 모바일 데이타베이스가 동일 밴더의 제품이어야 한다. 또한 비즈니스 애플리케이션 작성시 동기화를 위하여 특정 라이브러리를 사용하거나 코드를 변경하는 제한사항들이 존재한다. 이러한 문제들 때문에 모바일 비즈니스 시스템의 확장성(extensibility), 적응성(adaptable)과 융통성(flexibility)이 현격하게 감소하게 된다. 다양한 자원들이 공존하여 이질성이 높은 모바일 컴퓨팅 환경에서 이러한 특성의 감소는 효과적인 모바일 비즈니스 시스템 구축을 어렵게 하며 특정 밴더의 영향력만을 증가 시킨다.

본 논문에서는 위에서 언급된 문제를 해결하기 위하여 메시지 다이제스트 기반의 동기화 알고리즘(Synchronization Algorithms based on Message Digest : SAMD)을 제안한다. SAMD는 ISO(International Standard Organization)에서 명시된 SQL 표준의 질의문과 데이타 조작어(DML: Data Manipulation Language)만을 사용하여 동기화 문제를 해결함으로 데이타베이스 서버와 모바일 데이타베이스의 종류에 상관없이 어떠한 조합으로도 동기화가 가능하여 확장성, 적응성, 융통성을 제공한다. SAMD는 데이타베이스 서버와 모바일 데이타베이스 사이에 중복된 로우들을 메시지 다이제스트를 하여 양측에 이미지를 만들고 이 두개의 이미지를 비교하여 동기화가 필요한 로우들만 선별한다. 만일 동일 로우에 대하여 양측의 메시지다이제스트 값이 상이하면 중복된 로우의 값의 변화가 있는 것임으로 SAMD를 이용하여 동기화를 해야 한다. 메시지 다이제스트는 주로 보안 프로토콜에서 전송되는 데이타의 변조 여부를 판단하기 위해서 사용된다. 이 과정에서 커다란 양의 데이타가 작은 양이 데이타로 변경되므로, 저장 공간의 낭비를 최소화 하고 데이타 불일치를 감지하는 과정을 간단하게 할 수 있다. 메시지 다이제스트 함수는 제한된 자원에서도 빠르게 작동하기 때문에 자원이 제한된 모바일 단말기에 부담을 적게 준다. 또한 메시지 다이제스트 테이블들을 모두 데이타베이스 서버에 위치하게 하여 모바일 단말기가 SAMD를 이용한 동기화를 위하여 부가적으로 저장장소를 낭비하는 것을 방지 하였다. 또한 SAMD는 성능면에서도 기존의 솔루션보다도 우수한 결과를 나타낸다.

본 논문의 구성은 다음과 같다. 2장에서는 관련연구로서 데이타 동기화에 대한 기존 연구에 대해서 소개하고, 본 논문의 알고리즘에서 사용된 메시지 다이제스트의 개념에 대해서 설명한다. 3장에서는 데이타베이스 서버와 모바일 클라이언트 간의 동기화가 필요한 환경인 동기화 프레임워크에 대해서 설명하고, 4장에서 제안된 메시지 다이제스트 기반의 동기화 알고리즘을 설명한다. 5장에서는 SAMD를 구현하여 성능평가와 정성평가를 실시한다. 마지막으로 6장에서는 본 논문에 대한 결론을 내리며, 향후 연구 과제를 제시한다.

2. 관련연구

2.1 데이타 동기화

데이타 동기화는 크게 publish/subscribe 복제 관리, 데이타 동기화 절차, 불일치 탐지 및 해결 등의 3단계로 나누어진다. 일반적으로 모바일 데이타베이스 환경에서는 publish/subscribe 모델에 따라 중앙 서버 데이타베이스와 연동한다. 중앙 서버 데이타베이스는 모바일 데이타베이스에서 데이타를 사용할 수 있도록 데이타를 제공하고(publish), 모바일 데이타베이스는 중앙 서버 데이타베이스가 제공하는 데이타의 일부를 복제하여(subscribe) 사용한다[1,10].

publish/subscribe 모델에 의해 데이타가 모바일 데이타베이스로 복제되어 관리되면 상호간의 데이타 일치성 유지가 필요하다. 즉, 서버나 모바일 데이타베이스에서 데이타 변경이 발생하면 상호간에 데이타를 동일하게 유지시키는 동기화 작업이 필요하다. 데이타 동기화는 일반적으로 모바일 데이타베이스와 서버 간에 양방향으로 진행된다. 모바일 데이타베이스에서 변경이 발생하고 이를 서버에게 전달하여 데이타 동기화를 요구하는 경우와

서버에서 변경이 발생하여 이를 모바일 데이타베이스에게 전달하여 데이타 동기화를 요구하는 경우가 있다.

모바일 데이타베이스의 요구에 의한 동기화는 다음과 같은 절차로 이루어진다. 먼저 모바일 데이타베이스에서는 동기화가 마지막으로 이루어진 이후에 발생한 변경 사항을 결정한다. 변경이 발생하는 경우는 데이타를 새로 생성하거나 기존의 데이타를 갱신하는 경우, 데이타를 삭제하는 경우가 있다. 변경 사항 결정은 before-image table을 이용하는 방식 혹은 변경 연산에 대한 로깅을 이용하는 방식 등이 있다. 다음에 변경사항을 서버에 전달한다. 변경사항 전달은 일련의 데이타 변경 과정을 순서대로 전달할 수도 있고, 변경 과정을 통합하여 전달할 수도 있고, 응용 로직 자체를 전달하는 방법 등이 있다. 서버에서는 동기화가 마지막으로 이루어진 이후에 무슨 변경이 발생했는지 결정하고, 데이타 불일치를 탐지 및 해결한 후 서버의 데이타를 변경한다. 서버는 최종 데이타를 모바일 데이타베이스에 전달하여 모바일 데이타베이스의 데이타를 서버 데이타와 일치시킨다.

2.2 메시지 다이제스트

메시지 다이제스트는 임의의 길이의 메시지를 고정 길이 해시 값으로 사상시키는 함수인 단 방향 해시 함수로 구성된다. 해시 함수는 다양한 임의의 크기의 메시지 M을 입력으로 받아서 해시 코드 H(M)로 나타내는 고정 길이 결과를 만들어낸다. MAC(Message Authentication Code)과는 달리 해시 코드는 키를 사용하지 않으며, 입력 메시지만을 이용하는 함수이다. 이러한 해시 코드를 메시지 다이제스트, 혹은 해시 값이라고 부른다. 메시지 다이제스트는 메시지의 모든 비트에 대한 함수이며, 오류 검출 능력을 갖는다. 메시지 내의 비트가 하나라도 변하면 결과 메시지 다이제스트에도 변화가 나타난다[11].

메시지 다이제스트 h는 다음과 같은 형식을 갖는 함수 H에 의해 생성된다.

$$h = H(M)$$

M은 임의의 길이의 메시지이며, H(M)는 고정 길이 메시지 다이제스트이다. 메시지가 정확하다고 판단되면 메시지 다이제스트를 원본에 덧붙인다. 수신자는 메시지 다이제스트를 다시 계산함으로써 메시지를 인증한다. 메시지 다이제스트 그 자체로는 안전하다고 할 수 없으므로, 메시지 다이제스트를 보호하기 위한 어떤 수단이 요구된다.

메시지 다이제스트를 생성하기 위해 이용되는 해시 함수의 목적은 파일, 메시지, 혹은 데이타 블록의 "지문"을 만들어 내는 일이다. 효과적인 메시지 인증을 위해서 해시 함수 H는 다음의 원칙을 지켜야 한다[12].

1) H는 어떤 크기의 데이타 블록에도 적용될 수 있다.
2) H는 고정 길이 결과를 만든다.
3) H(x)는 주어진 x에 대해서 상대적으로 계산하기 쉬우므로, 하드웨어 및 소프트웨어 구현이 현실적이다.
4) 어떤 값 h에 대해서도 H(x)=h인 x를 찾는 것은 계산 불가능하다. 이것을 one-way 특성이라고 한다.
5) 어떤 블록 x에 대해서도 H(y)=H(x)이며 $y \neq x$인 y를 찾는 것은 계산 불가능하다. 이것을 weak collision resistance라고 한다.
6) H(x)=H(y)인 (x, y) 쌍을 찾는 것은 계산 불가능하다. 이것을 "strong collision resistance"라고 한다.

3. 동기화 프레임워크

3.1 동기화 모델

그림 1은 모바일 비즈니스 환경에서 동기화 서버를 이용한 일반적인 동기화 프레임워크이다. 전체의 프레임워크는 데이타베이스 서버, 동기화 서버(AnySyn) 그리고 모바일 데이타베이스가 내장된 여러 개의 모바일 단말기로 구성 된다.

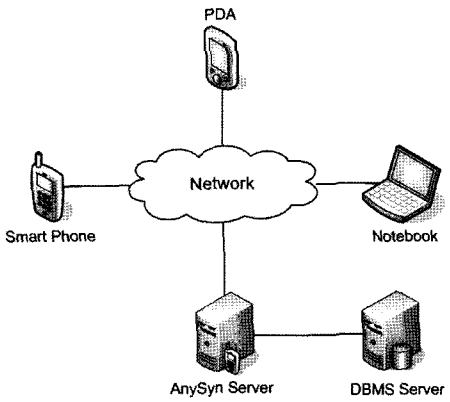

그림 1 모바일 환경의 동기화 프레임워크

데이타베이스 서버는 비즈니스에 필요한 모든 데이타들이 관리되며 모바일 데이타베이스는 모바일 단말기 사용자가 필요로 하는 데이타의 복사본을 데이타베이스 서버로부터 다운받아 사용한다. 동기화 서버는 이들 사이에 위치하여 동기화를 실행하며 동기화에 필요한 부가적인 정보를 관리한다. AnySyn 동기화 서버는 SAMD 알고리즘을 기반으로 동기화 작업을 수행하는 동기화 서버이다. AnySyn에는 동기화 정책이 설정되어 동기화에 적용되고 커넥션 풀을 운영하여 데이타베이스서버 접근에 따른 부담을 최소화 한다. 모든 모바일 단말기는 별도의 툴킷을 사용하여 유선으로 AnySyn 동기화 서버에 접근하여 동기화를 수행한다.

3.2 불일치

불일치는 데이타베이스 서버의 원본 데이타 혹은 모바일 데이타베이스의 복사본 데이타가 변화에 의해서 서로 다른 값을 가지고 있는 상태를 의미한다. 양측은 독립적인 데이타의 추가, 삭제, 변경 등의 변화가 가능함으로 필연적으로 불일치가 일어난다. 표 1은 하나의 로우에 대해서 불일치가 일어나는 모든 경우를 나타낸다.

표 1에서 가능한 경우는 모두 16가지이지만, 이들 중 case 6, 7, 8, 10, 14는 ADD 연산이 포함되므로, 실제로 하나의 로우에 대해서 일어날 수 없다. 예를 들어 Case 7의 경우 서버 측에서 추가된 로우가 클라이언트에서 변경된 로우와 다른 로우이므로 불일치로 볼 수 없다. 이 경우 Case 3과 Case 5가 각각 일어난 것과 동일하다. 다른 4가지의 경우도 이와 유사함으로 SAMD는 이들 5개의 case에 대해서는 고려하지 않는다.

4. SAMD 동기화 알고리즘

4.1 SAMD의 목표

SAMD 동기화 알고리즘은 확장성, 적응성과 융통성을 보장하기 위하여 아래와 같은 요구사항을 만족한다.

1) 데이타베이스에 대하여 독립적.
   특정 데이타베이스에 종속적인 메타데이타나 내부 기능을 사용하지 않는다.
2) 표준 SQL문만을 사용하여 동기화.
   ISO에서 명시된 표준 SQL의 질의문과 데이타 조작어만 사용하여 동기화 한다. 따라서 트리거를 이용한 어떠한 데이타 조작도 불가능 하다.
3) 데이타베이스 서버의 데이타 테이블의 구조를 변경 불가.
   동기화에 필요한 데이타를 추가하기 위해서 기존의 데이타 테이블의 구조를 변경하지 않는다. 즉 동기화 기능이 기존의 데이타 테이블 구조와는 독립적으로 실행되어야 한다. 따라서 타임 스템프등과 같은 정보가 데이타테이블에 추가 될 수 없다.
4) 애플리케이션에 구현 시 제한사항 추가 불가
   동기화 기능을 위하여 애플리케이션에 추가적인 어떤 일을 하거나 특정 라이브러리를 사용해야 하는 제약 사항이 있으면 안 된다.

기존의 상용 데이타베이스들의 동기화 방법은 위와 같은 특징을 모두 만족시켜주지는 못한다.

4.2 알고리즘의 기본개념

제안하는 SAMD 동기화 알고리즘은 데이타들의 이미지를 생성하기 위하여 메시지 다이제스트 함수를 사용한다. 보안 분야에서 메시지 다이제스트 함수를 이용한 데이타 무결성 보장 방법은 그림 2와 같다.

송신측은 해쉬 함수를 사용하여 전송하고자 문서에

표 1 불일치 분석

C	모바일 DB	DB 서버	C	모바일 DB	DB 서버	C	모바일 DB	DB 서버	C	모바일 DB	DB 서버
1	UC	UC	5	UC	ADD	9	UC	MOD	13	UC	DEL
2	ADD	UC	6	ADD	ADD	10	ADD	MOD	14	ADD	DEL
3	MOD	UC	7	MOD	ADD	11	MOD	MOD	15	MOD	DEL
4	DEL	UC	8	DEL	ADD	12	DEL	MOD	16	DEL	DEL

(C:Case, UC:Unchange, ADD:Addition, MOD:Modification, DEL:Deletion)

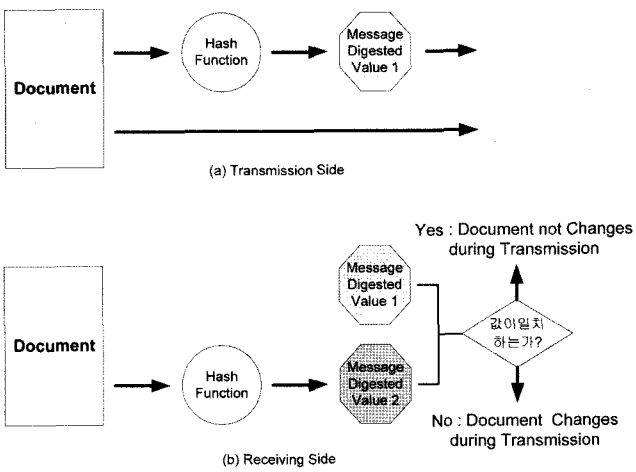

그림 2 메시지 다이제스트

대한 메시지 다이제스트 값 1을 생성한다. 이 과정에서 문서는 크기가 작은 메시지로 변환된다. 여기서 사용하는 해쉬함수는 충돌을 회피할 만큼의 충분한 안전성을 가지고 있다. 송신측은 문서와 메시지 다이제스트 값 1을 수신측에 보낸다. 수신측과 송신측은 동일한 해쉬 함수를 공유하고 있음으로 수신측은 이 함수를 전송받은 문서에 적용하여 메시지 다이제스트 값 2를 생성한다. 만일 두 값이 동일하면 전송된 문서는 전송되는 동안 변질되지 않았다는 것을 보장 할 수 있다. 이러한 방법은 문서보다는 간단한 메시지 다이제스트 값을 비교하여 문서의 변질을 확인할 수 있음으로 매우 효율적인 방법이다.

이 방법을 관계형 데이타베이스에 적용하여 두 테이블의 로우 사이의 데이타 동일성을 검사하는 방법은 그림 3과 같다.

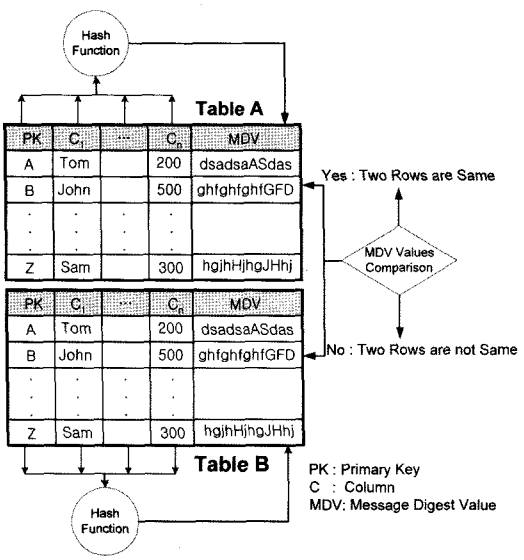

그림 3 데이타 테이블에서의 메시지 다이제스트

두 개의 테이블A와 테이블B에서 두 로우의 메시지 다이제스트 값이 동일하면 두 로우의 데이타는 동일하다. 만일 두 값이 동일하지 않다면 두 로우는 하나 이상의 상이한 컬럼 값을 가지고 있는 것이다. 따라서 이러한 방법은 두 로우 사이의 불일치를 검출하는데 유용하게 사용될 수 있다. 기본 키(Primary Key)를 기반으로 하여 불일치가 존재하는 로우가 식별이 되면 동기화 정책에 의해서 결정된 동기화 방향으로 로우를 복사하면 된다. 이러한 동기화 방법은 데이타베이스 내부의 함수, 로그, 메타데이타 등에 의존하지 않고 변경된 로우를 식별함으로 데이타베이스에 독립적인 동기화를 가능하게 한다.

4.3 SAMD 동기화 알고리즘

그림 4는 SAMD 동기화 알고리즘이 적용되는 데이타베이스 서버와 모바일 데이타베이스의 테이블 구조를 나타낸다. 두 데이타베이스 모두 데이타 테이블(DSDT : Database Server Data Table, MCDT : Mobile Client Data Table)과 메시지 다이제스트 테이블(DSMDT : Database Server Message Digest Table, MCMDT : Mobile Client Message Digest Table)을 가지고 있다. 데이타 테이블은 비즈니스 데이타가 들어있는 테이블이고 메시지 다이제스트 테이블는 데이타 테이블의 데이타들을 메시지 다이제스트한 값을 보관하는 테이블이다. 따라서 하나의 데이타 테이블은 하나의 메시지 다이제스트 테이블과 논리적으로 쌍을 이룬다. 메시지 다이제스트 테이블의 구조는 데이타 테이블의 기본키(PK : Primary Key) 칼럼, 메시지 다이제스트 값(MDV : Message Digest Value) 칼럼, 플래그 칼럼으로 이루어진다. 플래그 컬럼은 해당 로우에 불일치가 발생함을 나

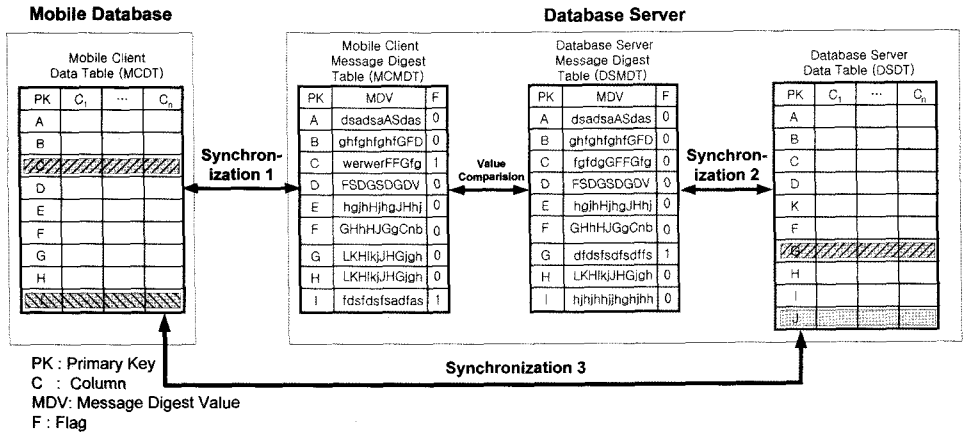

그림 4 테이블 구조

타냄으로 동기화가 필요한 로우를 식별 하는데 사용된다. 그림 4에서 PK 값이 A인 로우의 경우 메시지 다이제스트 값이 동일함으로 동기화할 필요가 없다. 그러나 PK값 C 로우의 경우는 MCMDT의 플래그 값이 1 임으로 동기화가 필요하다. 이러한 동기화 과정은 3.2절에서 언급된 모든 불일치를 해결하며 로우 단위로 이루어진다. 예를 들어 C 로우의 불일치의 경우 모바일 데이타베이스에서 데이타베이스 서버 측으로 동기화가 일어나며 MCDT의 C 로우 값으로 DSDT의 C 로우 값이 대치된다.

그림 4와 같이 동기화 알고리즘은 동기화 1, 동기화 2 부분과 동기화 3 부분으로 나누어진다. 동기화 1, 동기화 2 부분은 데이타 테이블과 메시지 다이제스트 테이블을 동기화하는 부분이다. 따라서 두 동기화 알고리즘은 알고리즘 적용 대상만 상이하고 알고리즘은 동일하다. 이 부분에서는 데이타 테이블의 각 로우값을 가지고 메시지 다이제스트 값을 만들고 이 값을 메시지 다이제스트 테이블의 메시지 다이제스트 값과 비교한다. 만일 값이 동일하면 데이타의 변화가 없음으로 동기화가 필요치 않다. 만일 값이 상이하다면 데이타 테이블의 값의 변화가 있음으로 메시지 다이제스트 테이블의 값을 새로운 다이제스트 값으로 갱신하고 플래그 값을 1로 변경한다. 플래그 값은 동기화가 필요한 로우를 식별하기 위한 정보로 사용된다. 데이타베이스 서버 안에서 하나의 DSDT당 하나의 DSMDT가 존재한다. 그러나 MCMDT는 크기는 DSMDT에 비해 작지만 모바일 단말기 숫자만큼의 MCMDT를 가지게 된다. 모바일 단발기의 동기화 요청이 있을 때마다 DSDT의 전체 로우에 대해여 동기화 2를 실시하는 것은 매우 비효율 적이다. 동기화 1을 먼저 실시하여 동기화가 필요한 로우를 식별하고 동기화 2에서는 식별된 로우에 대해서만 동기화가 일어나게 하였다. 동기화 3 부분은 두 메시지 다이제스트 테이블의 플래그 값을 사용하여 불일치의 종류를 분석하고 불일치별로 두 데이타 테이블 사이에서 동기화 작업을 수행 한다. 동기화 작업 후에는 메시지 다이제스트 테이블에서 동기화된 로우의 플래그 값을 0으로 한다.

대부분의 모바일 단말기들은 제한된 자원을 가지고 있음으로 동기화 과정에서 최대한 모바일 단말기에게 부담을 주면 안된다. 따라서 그림 4에서처럼 메시지다이제스트 테이블을 모두 데이타베이스 서버측에 위치시켜 모바일 단말기의 저장장소 낭비를 방지하였다. 동기화 1 과정에서 네트워크 사용으로 인한 부담이 있지만 MCDT의 데이타의 양이 서버에 비하여 작다. 그리고 모바일 단말기의 동기화를 위해 필요한 데이타가 유선통신을 통하여 배치처리가 가능한 SQL질의를 사용하여 AnySyn 동기화 서버로 한번에 전달되고 그 다음부터는 모바일 단말기에 부담이 없음으로 동기화 1 단계에서의 네트워크 사용으로 인한 부담을 줄일 수 있다.

SAMD 동기화 알고리즘은 다음과 같은 제한사항을 준수해야 한다.

1) 모든 데이타베이스 테이블은 기본키를 가지고 있다.
2) 데이타 테이블의 기본키와 메시지 다이제스트 테이블의 기본키는 동일 로우에 대하여 동일 값을 가진다.
3) 모바일 데이타베이스와 데이타베이스 서버에 각각 새로운 로우 삽입할 때 동일한 기본키 값을 가지지 못한다.

관계형 데이타베이스 모델은 테이블마다 기본키가 있고 로우를 식별하기 위해서 사용됨으로 조건 1)과 조건 2)는 기본적인 조건이다. 조건 3)은 모발 데이타베이스와 데이타베이스 서버 사이의 동기화 과정에서 기본키에 의한 무결성 충돌이 일어나지 않는 것을 의미한다. 실제로 양쪽에서 동일 기본키 값의 삽입이 가능하지만 이것은 애플리케이션 수준의 처리나 관리자의 정책에 의해서 충분히 해결 할 수 있음으로 고려하지 않는다.

그림 5는 SAMD동기화 알고리즘의 순서도이다.

S1~S3 단계는 그림 4의 동기화 2 과정을 나타낸다. DSDT와 DSMDT를 양쪽 외부조인(fullouterjoin)했을 때 허상 로우(dangling row)들을 이용하여 S1, S2, S3 단계를 적용할 DSDT의 로우들을 식별할 수 있다.

S1 단계는 DSDT의 로우가 수정된 경우 DSMDT와 동기화 하는 과정이다. DSDT 테이블의 로우에 대하여 메시지 다이제스트 값을 계산하여 DSMDT의 MDV 컬럼의 값과 비교한다. 두 값이 동일한 경우는 DSDT 로우가 아무런 변화가 없는 것이고 값이 다른 경우는 DSDT의 로우가 변경된 것이다. 따라서 DSDT의 이 로우의 메시지 다이제스트 값으로 DSMDT의 MDV 컬럼의 값을 교체한 후 플래그 컬럼을 1로 변경한다.

S2~S3 단계는 DSDT에서 로우가 삽입 또는 삭제된 경우를 동기화 하는 과정이다. 삽입의 경우 DSMDT에 삽입된 로우의 기본키와 메시지 다이제스트한 값을 삽입한다. 삭제의 경우는 DSMDT에 삭제된 로우의 MDV 컬럼의 값만을 NULL값으로 변경한다. 만일 NULL값으로 하지 않고 DSMDT에서 해당 로우를 삭제하게 되면 DSDT에 로우를 삽입한 것과 구분이 되지 않는다.

S4~S6 단계는 동기화 1 과정을 나타내는 과정이다. 이 단계는 MCDT와 MCMDT를 동기화 하는 과정으로 기본 알고리즘은 S1~S3 단계와 동일하다. 그러나 MCDT와 MCMDT는 상이한 밴더의 데이타베이스 내부의 데이타 테이블이고 물리적으로도 분리되어 있음으로 S1~S3 단계에서처럼 직접적인 양쪽 외부조인이 불가능하다. 따라서 이 경우는 임시 테이블을 만들어 잠시

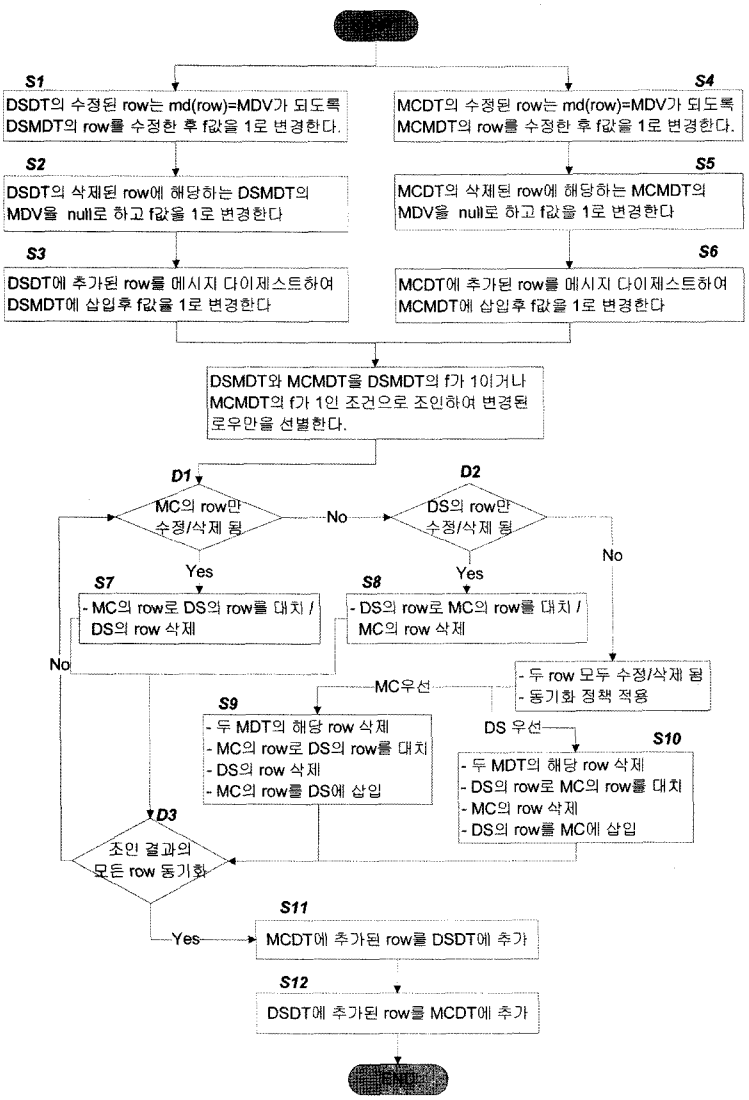

그림 5 SAMD 동기화 알고리즘

서버측으로 MCDT의 테이블의 내용을 복사한 후 동기화 1 과정을 수행 후 복사된 데이타를 삭제 한다. 이 과정이 데이타베이스 밴더에 대해서 SAMD 동기화 알고리즘의 독립성을 보장해주는 과정이다.

S7~S12 단계는 그림 4의 동기화 3 과정을 나타낸다. DSMDT와 MCMDT를 양쪽 외부조인(fullouterjoin)했을 때 허상 로우(dangling row)와 DSMDT와 MCMDT의 프래그를 이용하여 동기화 대상의 로우와 불일치의 종류를 식별하고 DSDT와 MCDT을 동기화 과정이다.

S7 단계는 MCDT에서 수정 혹은 삭제된 로우를 DSDT에 동기화하는 단계이다. D1 조건에서 MCMDT의 플래그 값이 1이고 DSMDT의 플래그 값이 0인 로우를 찾는다. 이 로우는 MCDT에서 수정 혹은 삭제가 일어난 것을 의미한다. 만일 MCMDT의 MDV 컬럼이 널이면 MCDT에서 삭제가 일어난 것을 의미하고 그렇지 않으면 수정이 일어난 것을 의미한다. 삭제의 경우는 DSDT, DSMDT, MCMDT에서 MCDT의 삭제된 로우와 일치하는 로우를 삭제하면 된다. 수정의 경우는 MCDT의 로우 값으로 DSDT의 로우값을 대치하고 MCMDT의 로우 값으로 DSMDT의 로우값을 대치한다. 동기화 후에 DSMDT와 MCMDT의 동기화된 로우의 플래그 값은 0으로 수정한다. 이 단계가 완료되면 C3, C4 불일치가 해결된다.

S8 단계는 DSDT에서 수정 혹은 삭제된 로우를

MCDT에 동기화 하는 단계이다. 이 단계는 S7단계의 알고리즘과 동일하며 단지 동기화가 S7과는 반대로 DSDT에서 MCDT쪽으로 일어난다. 이 단계가 완료되면 C9, C13 불일치가 해결된다.

S9, S10 단계는 DSDT와 MCDT에서 모두 수정 혹은 삭제가 일어난 경우 동기화 정책에 따라 DSDT에서 MCDT쪽으로 동기화 하던가 그 반대 방향으로 동기화 하는 단계이다. DSMDT와 MCMDT의 플래그 값이 모두 1인 로우들만이 동기화의 대상이 된다. S9, S10 단계에서는 표 2와 같은 4가지의 경우가 고려되어야 한다.

표 2 삭제와 삽입에 따른 경우 분석

경 우	모바일 데이타베이스	데이타베이스 서버
1	삭 제	삭 제
2	수 정	삭 제
3	삭 제	수 정
4	수 정	수 정

S9단계는 MCDT에서 DSDT쪽으로 동기화가 일어난다. 경우 1은 DSDT와 MCDT의 동일 로우가 모두 삭제되었음으로 DSMDT와 MCMDT의 동일 로우도 모두 삭제한다. 경우 3은 경우 1의 작업을 수행하고 추가적으로 DSDT에서 MCDT에서 삭제된 로우와 동일 로우를 삭제하면 된다. 두 경우가 완료되면 C12, C16 불일치가 해결된다. 경우 2, 4는 MCMDT의 수정된 로우로 DSMDT의 동일 로우를 수정한다. 경우 2는 DSDT의 로우가 삭제되었음으로 MCDT의 로우값을 DSDT에 삽입한다. 경우 4는 DSDT의 로우가 변경되었음으로 MCDT의 로우값으로 DSDT의 동일 로우값을 수정한다. 두 경우가 완료되면 C11, C15 불일치가 해결된다.

S10 단계는 DSDT에서 MCDT쪽으로 동기화가 일어난다. S9단계와 알고리즘은 동일하고 동기화 방향만 다르다. S10 단계가 완료되면 C15, C16, C12, C11 불일치들이 해결된다.

S11 단계는 MCDT에 삽입된 로우를 DSDT에 반영하는 단계이다. MCMDT의 플래그 값이 1이고 허상 튜플(dangling tuple)인 로우에 적용된다. MCDT와 MCMDT에 삽입된 로우들을 DSDT와 DSMDT에도 삽입한다. 이 단계가 완료되면 C2,C6,C10,C14 불일치가 해결된다. S12 단계는 DSDT에 삽입된 로우를 MCDT에 반영하는 단계이다. 알고리즘은 S11과 동일하며 동기화 방향만이 다르다. 이 단계가 종료되면 C5,C6,C7,C8 불일치가 해결된다.

SAMD 알고리즘이 수행되면 동기화 과정을 통하여 표 1에서 분석된 모든 불일치가 해결된다. 따라서 SAMD 알고리즘은 모든 발생 가능한 불일치를 동기화 할 수 있다.

4.3 SAMD의 트랜잭션과 병행 처리

그림 6은 SAMD 알고리즘의 트랜잭션 처리 방법을 보여준다. SAMD 알고리즘의 앞부분에 있는 동기화 1과 동기화 2를 트랜잭션 처리 단위로 하였다. 동기화 1은 MCDT의 모든 로우들에 대해여 메시지 다이제스트 값을 만들어 값이 변경된 로우들을 식별하는 과정으로 한 트랜잭션 안에 처리 되는 것이 합당하다. 왜냐하면 동기화1 과정 중에 실패(Fail)가 일어나고 다시 동기화 1을 수행할 때 실패가 일어난 로우부터 다시 동기화를 수행한다면 두 동기화 사이에 실패 이전에 동기화 되었던 MCDT의 로우가 변경되지 않았다는 것을 보장하지 못하기 때문이다. 동기화 2도 동기화 1과 동일한 이유에서 동기화 단위로 사용 된다.

SAMD 알고리즘이 시작되면 동기화 1이 성공했는지를 확인하고 동기화가 실행되지 않았으면 동기화 1을 수행한다. 동기화 1이 수행 중에 실패가 일어나면 롤백(Rollback)이 일어나 동기화 1이 수행되기 전의 상태로 되돌아간다. 동기화 1이 성공적으로 수행이 되면 그 다음 단계인 동기화 2로 이동한다. 동기화 2단계에서도 동기화 1과 동일한 방식으로 트랜잭션 처리를 한다. 만일 어떤 이유에 의해서 SAMD를 처리하는 프로세서가 문

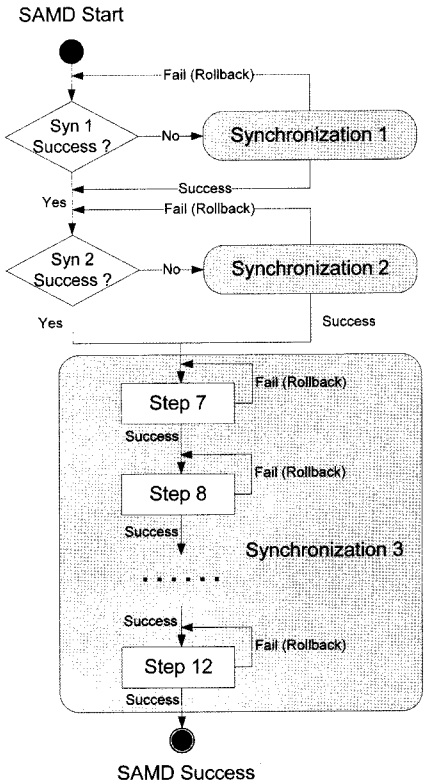

그림 6 SAMD의 트랜잭션 처리

제를 일으켜 다시 동기화가 시작 되는 경우 SAMD는 트랜잭션 처리 정보를 유지하고 있기 때문에 불필요한 동기화 단계를 반복하지 않는다. 예를 들어 동기화 2 단계에서 문제가 일어나 SAMD 처리 프로세서가 멈추었고 다시 시작되는 경우 이미 동기화 1단계는 성공적으로 수행되었음으로 다시 수행되지 않고 그 다음 단계인 동기화 2단계로 이동된다.

동기화 3단계에서는 그림 5의 SAMD 알고리즘에서 직사각형으로 표현되는 각 단계를 트랜잭션을 처리하는 단위로 사용하였다. 동기화 1과 동기화 2는 동기화를 위한 준비 작업에 해당하고 실질적으로 동기화가 이루어지는 단계는 동기화 3이다. 동기화 3에서 표 1에서 분석한 모든 불일치가 해결이 된다. 따라서 이 모든 단계를 트랜잭션 단위로 하면 너무 범위가 크고 실패 발생 시 동기화가 성공적으로 이루어진 로우들도 불필요하게 실패 이전의 불일치 상태로 되돌아간다. 예를 들어 S9 단계에서 실패가 일어났어도 S7과 S8단계에서 동기화한 내용을 롤백 하지 않아도 된다. 왜냐하면 실패 후 다시 동기화 3 과정 수행 시 S7과 S8에서 동기화된 로우는 이미 동기화되었음으로 동기화 대상 로우에서 제외되기 때문이다. 따라서 그림 6에서 동기화 1과 동기화 2의 우측에 있는 동기화 여부를 묻는 조건과 이 조건에 응답을 하기위한 정보를 유지 하지 않아도 된다. 또한 긴 트랜잭션 처리 기간 동안에 서버 DSDT의 로우들을 잠금(Locking)하고 있기 때문에 여러 모바일 기기들이 동기화를 요청하는 경우 병행처리를 힘들게 한다. 동기화 3에서 각 단계는 성공적으로 동기화가 되면 그다음 단계로 이동이 되고 실패 하는 경우 해당 동기화 단계를 다시 수행 하게 된다.

이러한 트랜잭션 처리를 위하여 배치(Batch)의 개념을 사용 하였다. 각 동기화 단위에서 필요한 모든 SQL 문장들을 하나의 배치로 묶어 배치 단위로 SQL문장을 수행 하였다. 이들 문장들은 모두 성공하여 데이타베이스에 반영되거나 아니면 동기화 전 단계로 롤백 된다.

AnySyn 서버는 쓰레드와 전통적인 생산자/소비자 알고리즘을 사용하여 병행처리를 한다. AnySyn 서버는 비동기식(asynchronous) 방식으로 모바일 단말기로부터의 동기화 요청을 처리한다. 즉 AnySyn 서버는 요청이 없는 경우는 대기상태에 있다가 동기화 요청이 있는 경우만 동기화 작업을 수행한다. 3장에서 제시한 동기화 프레임워크에서 모바일 단말기가 온라인 상태에서 동기화를 원하는 시점에서만 동기화가 일어남으로 일반적인 비즈니스를 고려 할 때 특정 시점에 동기화 요청이 집중 될 수 있다. 따라서 지속적으로 동기화 요청을 계속해서 확인 하는 동기식(synchronous)방법은 적당하지 않다. 모바일 단말기의 동기화 요청이 오면 이 사실을 AnySyn 서버에 알리고 동기화 큐(Queue)로 들어간다. 따라서 동기화 요청이 몰리는 경우 동기화 요청들은 자신의 차례가 올 때까지 동기화 큐에서 대기 하게 된다. 모든 동기화 요청이 동일한 중요성이 있다고 가정하여 큐에 들어온 순서대로 동기화가 이루어지도록 하였다. AnySyn 서버는 모바일 단말기로부터 동기화 요청이 오면 동기화 큐에 대기중인 동기화 요청들에게 SAMD 알고리즘을 수행하는 하나의 쓰레드를 할당하여 동기화를 수행한다.

만일 두 개의 쓰레드가 동일한 서버의 로우에 대하여 쓰기 작업을 하는 경우는 데이타베이스관리시스템(DBMS)이 제공하는 잠금(Locking)방법에 의하여 동기화를 처리한다. 즉 한 트랜잭션이 로우에 대해서 쓰기 작업을 하고 있으면 다른 트랜잭션은 이 쓰기 작업 트랜잭션이 종료된 후에 로우에 대해서 쓰기 작업을 할 수 있다. 동기화 1은 MCDT와 MCMDT와 일대일로 동기화가 일어남으로 쓰기작업에 의한 잠금으로 동기화 1이 지연되는 경우는 없다. 동기화 2와 동기화 3는 각 모바일 클라이언트들이 중복되게 DSDT의 로우의 복사본을 가지고 있지 않다면 쓰기작업에 의한 잠금으로 동기화 2와 동기화 3가 지연되지 않는다. 대부분의 비즈니스 환경의 경우 모바일 클라이언트들은 자신이 다룰 수 있는 자신만의 데이타 복사본만을 내려 받는다. 만약 모바일 클라이언트들이 중복되게 DSDT의 로우의 복사본을 가지고 있어도 트랜잭션 단위가 크지 않기 때문에 짧은 시간만을 지연하게 된다. 비즈니스 로직상 모든 경우에 모바일 클라이언트들이 중복되게 DSDT의 로우의 복사본을 가지고 있지 않다면 동기화 3 전체를 하나의 트랜잭션 단위로 처리 할 수 있다. 이 경우 동기화 과정에서 발생할 수 있는 네트워크 트래픽을 최소화시켜 동기화 시간을 단축할 수 있다.

5. 구현 및 비교 평가

5.1 구현 내용

SAMD 알고리즘은 Java(JDK 1.5)를 이용하여 구현하였으며 데이타베이스 연결은 JDBC를 이용하였다. 데이타베이스 서버는 Oracle9i를 사용하였으며 모바일 데이타베이스는 Sybase의 UltraLite를 사용 하였다. 메시지다이제스트를 위해서는 JLock 자바 보안 패키지[13]를 사용하였다. SAMD 동기화 알고리즘의 테스트에 사용된 주요 클래스는 표 3과 같다.

먼저 Create_s_dt 클래스를 사용하여 무작위로 생성된 로우들을 데이타베이스 서버에 삽입하고 SAMD 클래스를 이용하여 동기화 과정을 거쳐 모바일 데이타베이스에도 데이타베이스 서버와 동일한 로우가 존재하게 한다. 그 다음 TableModify 클래스를 이용하여 표 1에

표 3 구현 클래스

클래스	내용
SAMD	동기화 알고리즘의 실제 구현 클래스이다.
Create_s_dt	난수 발생기를 이용하여 임의의 문자열 데이타를 DSDT에 삽입한다.
TableModify	임으로 DSDT와 MCDT에 삽입, 삭제, 수정의 변화를 주어 16가지의 불일치 상황을 만든다.
VerifySync	동기화가 성공적으로 이루어졌는지를 검사 한다.

서 언급된 불일치 상황을 임으로 생성한다. 다시 SAMD 클래스를 이용하여 동기화를 하고 VerifySync 클래스를 사용하여 동기화가 성공적으로 이루어 졌는지를 검증한다. 검증과정에서 SAMD 동기화 알고리즘은 16개의 불일치 상황을 모두 해결하였다.

5.2 성능 평가

SAMD 동기화 알고리즘의 성능을 비교하기 위하여 기존의 밴더들의 제품들을 표 4와 같이 분석하였다. 1번, 3번, 5번의 경우 모바일 데이타베이스와 데이더베이스 서버가 반드시 동일 밴더야 함으로 밴더에 제약을 받지 않는 SAMD 와는 비교 대상이 아니다. 이들을 밴더의 고유 기술을 사용함으로 SAMD 동기화 알고리즘과 동일 조건에서 평가가 어렵다. 2번, 4번, 6번의 경우 데이타베이스 서버가 모두 범용 데이타베이스임으로 SAMD와의 비교 대상이 된다. 이중에서 2번의 Sybase의 MobiLink가 가장 대표적인 제품임으로 MobiLink와 성능 비교를 실시하였다.

성능 평가 조건은 SAMD와 MobiLink 모두 데이타베이스 서버는 Oracle 9i를 사용했으며 모바일 데이타베이스는 Sybase의 UltraLite를 사용하였다. 두 데이타베이스는 네트워크의 변수를 제거하기 위하여 같은 머쉰에 설치하였다. 먼저 Create_s_dt 클래스를 사용하여 무작위로 생성된 5000건의 로우를 Oracle에 삽입하고 각각의 동기화 과정을 거쳐 UltraLite에도 5000건의 로우가 삽입되게 하였다. 그 다음 TableModify 클래스를 이용하여 Oracle와 UltraLite에 각각 100건의 로우가 삽입되게 하고 또한 UltraLite에 100건의 로우가 삭제되고 100건의 로우가 수정 되도록 하였다. 수정, 삭제되는 데이타는 5000건의 로우중 골고루 배치되도록 조정하였다. 성능평가에서는 동일한 테이블 수정이 되었을 때 동기화 하는 시간을 측정 하여 비교하였다. 5000건의 데이타 중 400건의 데이타 변화는 일반적인 모바일 컴퓨팅 환경을 측정하기에는 무리가 없는 데이타 환경이다. 성능을 평가한 결과는 표 5의 성능평가 결과와 같이 SAMD가 MobiLink에 비하여 평균 0.64초 정도 빠른 성능을 보였다.

표 5 성능평가 결과

No	SAMD	MobiLink	성능차이 (SAMD-MobiLink)
1	1.53	2.29	- 0.76
2	1.52	2.31	- 0.79
3	1.53	1.87	- 0.34
4	1.50	2.31	- 0.81
5	1.53	2.30	- 0.77
6	1.55	1.89	- 0.34
7	1.53	2.30	- 0.77
8	1.53	1.94	- 0.41
9	1.71	2.32	- 0.61
10	1.53	2.29	- 0.76
평균	1.55	2.18	- 0.64

(단위 : 초)

MobiLink는 표 1과 같은 동기화가 필요한 상황에 대하여 이벤트를 발생시키고 이벤트에 대하여 동기화 SQL혹은 JAVA를 이용하여 스크립터를 각각 작성해야 하는데 이 작업은 매우 번거롭고 불편하여 익숙하기 까지 상당한 시간이 필요하다. 또한 표 1의 일부 경우는 지원하지 않는다. 그래서 표 1에서 명세한 다양한 경우에 대해서 평가하지 못하고 위에서 언급한 4가지 경우에 대해서만 성능평가를 실시하였다.

SAMD의 경우 별도의 스크립터를 작성할 필요가 없어 사용자 편의성에서도 장점을 가지고 있다. 또한 MobiLink의 경우 Sybase의 데이타베이스가 아닌 다른 데이타베이스 서버와 동기화를 위해서는 데이타베이스 서버측에 별도의 테이블, 프로시듀어, 트리거등을 생성해야 한다. 그러나 SAMD는 이러한 작업이 필요 없다. MobiLink를 사용하는 경우 반드시 UltraLite나 Sybase ASA와 같은 Sybase의 모바일 데이타베이스만을 사용

표 4 모바일 데이타베이스 동기화 제품 분석

No	제품	밴더	모바일 DB	DB 서버	구조
1	SQL Remote	Sybase	ASA (Adaptive Server Anywhere)	ASE (Adaptive Server Enterprise)	DBMS와 통합
2	MobiLink		UltraLite, Sybase ASA	범용 DBMS	Mid-tier
3	DB2 Everyplace Sync Server	IBM	DB2 Everyplace	DB2 Universal	Mid-tier
4	Cloudsync	informix	Cloudscape	범용 DBMS	Mid-tier
5	iConnect	Oracle	Oracle Lite	Oracle DBMS	DBMS와 통합
6	UniSync	PointBase	PointBase Micro	범용 DBMS	Mid-tier

해야 하지만 SAMD의 경우 데이타베이스서버, 모바일 데이타베이스 종류에 상관없이 사용가능 하다. 따라서 성능뿐만 아니라 사용자 편의성 측면에도 장점을 가지고 있다. 그러나 SAMD는 별도의 메시지다이제스트 테이블을 유지함으로 저장 공간의 효율성 측면에는 단점을 가지고 있다. 그러나 대용량의 저장 공간의 비용이 저렴하고 확장이 용이함으로 커다란 문제점이 되지는 않는다.

성능평가에 있어 SAMD는 동기화를 위한 핵심 부분만 구현이 되었고 제품이 가져야 하는 여러 가지 요소들을 구현 되지 않았다. 또한 실제 환경에 적용하기 위한 일반화가 부족하다. 따라서 상용제품인 MobiLink와 비교하는 것은 어느 정도 무리가 있다. 하지만 SAMD의 성능이 이러한 요소를 고려하더라도 MobiLink에 비하여 많이 뒤지지 않고 위에서 언급된 것과 같이 사용자편의성, 간단한 동기화 작업, 밴더 독립성 등의 장점을 가지고 있다.

5.3 정성 평가

제안된 알고리즘은 4장에서 명시한 알고리즘의 목표를 준수하도록 설계되었다. 언급된 목표들은 기존의 상용 동기화 제품들이 갖고 있는 문제점들로써, 이 목표를 달성하지 못하면 동기화 알고리즘의 적용에 범용성을 갖지 못한다. 현재 널리 사용되고 있는 Sybase의 MobiLink & SQL Remote[5], IBM의 Sync Server[6], Informix의 Cloudsync[7], Oracle의 iConnect[8]에 대하여, 명시된 알고리즘 목표를 기준으로 본 논문에서 제안된 알고리즘과 비교하였다. 다음 표는 각각 알고리즘 설계 목표와 각 제품 및 제안된 알고리즘의 목표 준수 여부를 나타내고 있다.

표 6 알고리즘 설계 목표

	알고리즘 설계 목표
R1	데이타베이스에 대하여 독립적
R2	표준 SQL문만을 사용하여 동기화
R3	데이타베이스 서버의 데이타 테이블의 구조를 변경 불가
R4	애플리케이션에 구현 시 제한사항 추가 불가

6. 결론 및 향후 연구

본 논문에서는 데이타베이스 서버와 모바일 데이타베이스 간의 데이타 동기화를 위하여 메시지 다이제스트 기반의 SAMD 동기화 알고리즘을 제안하였다. SAMD 알고리즘은 기본적인 관계형 데이타베이스의 SQL 기능만을 가지고 동기화를 수행함으로써 특정 밴더의 데이타베이스에 종속적이지 않고 어떠한 데이타베이스 서버와 모바일 데이타베이스 서버의 조합에도 사용 가능 하다. 따라서 모바일 비즈니스 시스템 구축시 확장성(extensibility), 적응성(adaptable) 과 융통성(flexibility)을 보장해 준다. 또한 성능 면에서도 기존의 제품에 비해서 우수한 성능을 보인다.

향후 과제로는 SAMD 알고리즘을 온라인 실시간 동기화 알고리즘으로 확대하는 작업이 필요하다. 또한 동기화를 위하여 다양한 사용자와 관리자 GUI 등의 개발이 필요하다.

참 고 문 헌

[1] 김상욱, 오세봉, 손성용, 이진호, "임베디드 데이타베이스 환경에서의 동기화를 위한 프레임워크", 정보과학회지, 제20권, 제7호, pp. 14-21, 2002.

[2] Tomasz Imielinski and B. R. Badrinath, "Mobile wireless computing: challenges in data management," Communications of the ACM, Volume 37 Issue 10, pp. 18-28, 1994.

[3] Barbara, D., "Mobile Computing and Databases - A Survey," IEEE Transactions on Knowledge and Data Engineering, Vol. 11 No. 1, pp 108-117, 1999.

[4] EPFL, U. Grenoble, INRIA-Nancy, INT-Evry, U. Montpellier, "Mobile Database: a Selection of Open Issues and Research Direction," SIGMOD Record, Vol.33, No.2, pp. 78-83, June, 2004.

[5] 최미선, 김영국, "이동(Mobile) 데이타베이스 개요 및 연구 현황", 데이타베이스 연구회지, 제17권, 제3호, pp. 3-16, 2001.

[6] "Synchronization Technologies for Mobile and Embedded Computing," A Whitepaper from Sybase, Inc.

[7] "DB2 solutions for mobile computing," IBM's White Paper.

[8] "Informix CloudSync," Informix's White Paper.

[9] 최윤석, "인터넷 환경을 위한 Oracle 8iLite", www.oracle.com/kr.

[10] 이상윤, 박순영, 이미영, 김명준, "이동 데이타베이스의 데이타 동기화 기술 분석", 데이타베이스 연구회지, 제17권, 제3호, pp. 29-41, 2001.

[11] William Stallings, "Cryptography and Network Security," 2003.

표 7 기존 제품과 SAMD의 설계목표 준수

	MobiLink	SQL Remote	Sync Server	Cloudsync	iConnect	SAMD
R1	○	X	X	○	X	○
R2	X	X	X	X	X	○
R3	○	○	○	○	○	○
R4	○	○	○	X	○	○

[12] Nechvatal, J., "Public Key Cryptography," 1992.
[13] Java Cryptography Library, J/LOCK, "http://www.stitec.com/products21.htm"

문 창 주

1997년 고려대학교 컴퓨터학과 학사. 1999년 고려대학교 컴퓨터학과 석사. 2004년 고려대학교 컴퓨터학과 박사. 2005년 고려대학교 정보보호대학원 연구교수. 2006년 건국대학교 컴퓨터응용과학부 조교수 2006년~현재 건국대학교 항공우주정보시스템공학과 조교수. 관심분야는 분산객체, 시스템통합, 정보보호, 데이타베이스

최 미 영

1990년 숭실대학교 전자계산학과 학사 1992년 포항공과대학 전자계산학과 석사 1993년 포스데이타. 1998년 (주)핸디소프트. 1999년 위세정보기술. 2006년 (주)엔코아 정보컨설팅. 2005년~현재 고려대학교 컴퓨터학과 박사과정. 관심분야는 데이타베이스, XML

김 상 민

2007년 건국대학교 컴퓨터응용과학부 학사 관심분야는 데이타베이스

정 진 환

2007년 건국대학교 컴퓨터응용과학부 학사 관심분야는 데이타베이스