튜플 단위 메시지 다이제스트를 이용한 이기종 모바일 데이터 동기화 기법☆

A Heterogeneous Mobile Data Synchronization Technique Using the Tuple Based Message Digest

박 성 진*
Seong-Jin Park

요 약

모바일 데이터베이스 환경에서는 서로 다른 데이터베이스 안에 동일한 데이터가 중복되는 상황이 발생하므로, 이러한 중복된 데이터의 일관성을 유지시켜 주기 위한 효율적인 동기화 기법이 요구된다. 본 논문에서는 모바일 환경하에서 클라이언트 데이터베이스와 서버 데이터베이스간의 중복된 데이터의 일관성을 유지시키기 위한 메시지 다이제스트 기반의 동기화 기법을 제안하였다. 제안한 데이터 동기화 기법은 내부적으로 튜플 단위의 메시지 다이제스트 결과를 충돌을 탐지하기 위한 정보로 사용함으로써 기존 동기화 방법에 비하여 범용성 및 확장성 측면에서 이점이 있다.

Abstract

In mobile database environments, the efficient synchronization technique is required to maintain the consistency of replicated data because the same data can be replicated between so many different databases. In this paper, we propose a message digest based synchronization technique to maintain the consistency of replicated data between client databases and a server database in mobile environments. The proposed data synchronization technique has the advantage of generality aspect and extensibility aspect by using the tuple-based message digest output to detect the data conflicts.

☞ Keyword : data synchronization, mobile database, message digest

1. 서 론

모바일 데이터베이스란 유·무선 소형 이동 단말기에 탑재되는 소형 데이터베이스를 의미하며 주로 클라이언트 시스템으로서 서버 데이터베이스와의 빈번한 데이터 동기화 작업이 요구된다. 즉, 클라이언트-서버 환경에서 양측의 데이터베이스는 오프라인 상태에서 각기 변경될 수 있으므로 일시적인 불일치성(temporary inconsistency)이 발생하게 된다. 이것을 충돌(conflict)이라 하며[1] 충돌을 탐지하고 해결하는 기능은 모바일 데이터베이스의 핵심 기능이다. 데이터 동기화(data synchronization)란 이러한 중복된 데이터 사이의 불일치성을 해결하는 과정[2]이다.

본 논문에서는 다양한 이기종 모바일 데이터베이스들이 혼재하는 환경에서 발생하는 동기화 문제를 해결하기 위하여 DBMS에 독립적인 동기화 기법을 제안하였다. 기존 동기화 기법은 특정 DBMS 종속적이기 때문에 DBMS가 변경되거나 여러 종류의 DBMS가 사용될 경우 동기화가 어려운 문제점이 있다. 제안한 데이터 동기화 기법은 동기화 대상 테이블에 대한 메시지 다이제스트 테이블의 복사본을 별도로 저장함으로써 이후에 변경되는 튜플 단위의 변경내용을 탐지하고 이에 대한 효율적인 동기화를 수행한다. 제안한

* 정 회 원 : 한신대학교 컴퓨터정보소프트웨어학부 교수
sjpark@hs.ac.kr
[2005/08/19 투고 - 2005/09/12 심사 - 2006/02/20 심사완료]
☆ 이 논문은 2006년도 한신대학교 학술연구비 지원에 의해 연구되었음

동기화 기법의 가장 큰 특징은 튜플 단위의 메시지 다이제스트를 통해 충돌 탐지를 위한 데이터의 양을 최소화하면서 어떠한 DBMS 사용 환경하에서도 적용가능한 DBMS 독립적인 데이터 동기화 기법이라는 점이다.

2. 관련 연구

기존 데이터 동기화 기법들은 기능상으로는 비슷하나, 동기화시 전달되는 메시지 구성의 최적화 정도 및 효과적인 충돌 탐지 방법에 따라 성능 차이가 발생하므로 적용 환경에 적합한 동기화 알고리즘에 대한 연구가 필요하다.

충돌을 해결하기 위한 변경 탐지에 대한 기존 연구들로는 MH-DIFF[3], ATBE[4], SCD[5] 그리고 BULD Diff[6] 등과 같은 계층형 데이터에 대한 변경 탐지 알고리즘과 관계형 데이터에 대한 변경 탐지 알고리즘[7], 그리고 유닉스 diff 유틸리티와 같은 문자열 변화 탐지 알고리즘 등이 있다. 모바일 데이터베이스 시스템을 개발하는 데 있어 가장 중요한 요구 사항의 하나는 다양한 이기종 DB와 유연하게 결합하여 사용될 수 있어야 한다는 점이다. 즉, 다수의 이기종 DBMS가 혼재하는 환경에서 추가 부담없이 동기화를 수행하기 위해서는 DBMS에 독립적으로 수행되는 동기화 기법이 요구된다.

여러 상용 DBMS에서 지원하고 있는 동기화 기법들은 일반적으로 데이터 동기화에 필요한 레코드의 변경 정보를 DBMS 내부에서 자체적으로 저장한다. 이를 위하여 변경 이전 값(Old Value)[8], 타임스탬프(TimeStamps)[9], 트랜잭션 로그(transaction log)[10] 등이 이용된다. 이러한 기존 연구에서는 레코드 변경시 타임스탬프를 변경하거나, 로그를 기록하거나, 이전 값을 저장하는 기능이 DBMS에 내부적으로 구현되어 있다. 즉, 동기화 알고리즘이 특정 DBMS에 종속되며 이 결과, 응용 환경에서 사용하는 DBMS가 바뀌는 경우 기존 동기화 알고리즘을 사용할 수 없는 문제점이 있다.

타임스탬프 방식은 서버와 클라이언트의 동기화 대상이 되는 각 레코드에 타임스탬프 필드를 추가하는 것이다. 이때, 타임스탬프는 서버에 존재하는 레코드의 최종 변경 시점을 표현하는 값이다. 클라이언트는 서버로부터 데이터를 다운로드할 때, 각 레코드와 대응되는 타임스탬프 값을 함께 다운로드 받는다. 이 값은 클라이언트의 DBMS 내에서 유지되다가 동기화시에 충돌 감지를 위하여 사용된다. 변경 이전의 값을 사용하는 방식은 클라이언트가 서버로부터 다운로드한 변경 전의 각 레코드 값을 이용한다. 클라이언트에서 변경이 일어난 후에도 이 값은 클라이언트의 DBMS 내에서 그대로 유지되다가, 동기화 시에 충돌 감지를 위하여 사용된다. 이 방식은 두 가지 버전의 값을 모두 유지해야 하므로 저장 공간에 오버헤드가 크고, 타임스탬프 방식은 타임스탬프 전송에 따른 네트워크 부하가 크다. 그리고, 두 동기화 기법 모두 특정 DBMS에 종속적이기 때문에 다양한 이기종의 모바일 데이터베이스가 사용될 수 있는 시스템의 동기화 기법으로는 적합하지 않다.

제안한 동기화 기법은 가장 최근 동기화 시점의 데이터들을 메시지 다이제스트로 요약한 복사본을 각각 DB에 저장함으로써 DBMS에 종속되거나 기존 응용프로그램 혹은 테이블에 영향을 주지않고 독립적으로 데이터의 변경 유무, 충돌 여부를 파악하여 동기화시킬 수 있다.

3. 메시지 다이제스트 기반의 동기화 모델

3.1 고려 사항

일반적으로 모바일 기기안의 데이터는 모바일 관계형 데이터베이스 안에 저장되어 있으며 서버의 데이터는 각종 관계형 데이터베이스 혹은 그룹웨어(관계형 데이터베이스 개념 지원)안에 저장되어 있다. 제안한 동기화 알고리즘은 다음과 같

은 특성을 만족해야 한다.

첫째, 동기화 알고리즘은 변화된 데이터만 감지하여 동기화함으로써 동기화에 따른 시스템의 연산 부담을 최소화하여 처리속도를 향상시킬 수 있어야 한다.

둘째, 동기화 알고리즘은 데이터베이스의 공통의 기본 인터페이스만을 사용함으로 동기화되는 데이터(데이터베이스)에 종속되지 않아야 한다.

셋째, 동기화 알고리즘으로 인해 테이블 구조에 제한을 가하거나 기존의 테이블 구조에 변경이 발생해서는 안 된다. 즉, 기존의 데이터베이스 시스템에 어떠한 변경도 요구하지 않아야 한다.

3.2 메시지 다이제스트

제안한 동기화 알고리즘은 메시지 다이제스트 MD4[11]를 기반으로 한다. MD4는 보안 표준의 하나로 일종의 해쉬 알고리즘으로 그 목적은 해쉬함수의 목적과 동일하다. 입력으로 임의의 길의(arbitrary length)의 메시지를 받아서 출력으로 128비트의 지문(fingerprint)형 메시지 다이제스트를 생성한다. 즉, 생성된 메시지 다이제스트는 (그림 1)과 같이 많은 양의 데이터(메시지)를 단방향 함수를 이용하여 작은 메시지로 작성된 것이다. MD4는 다음과 같은 해쉬 함수의 조건을 모두 충족한다.

(조건1) 임의의 길이의 메시지를 입력으로 받을 수 있어야 한다.

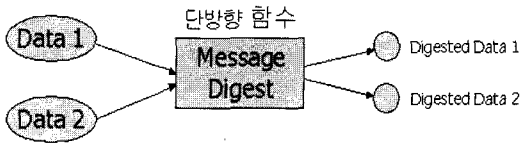
(조건2) 고정된 길이의 출력을 만들어야 한다.

(조건3) 모든 X에 대하여 f(X)의 계산이 쉬워야 한다.

(조건4) 주어진 X에 대해서 원래의 X를 구할 수 없어야 한다.

(조건5) f(X) = f(Y)인 X, Y를 구하기가 어려워야 한다.

MD4 알고리즘은 패딩 비트를 더하는 단계, 메시지의 길이를 더하는 단계, MD 버퍼를 초기화하는 단계, 16-워드 블록의 메시지 프로세싱 단계,



(그림 1) 메시지 다이제스트

출력 단계의 5 단계의 동작들을 통해 다이제스트를 생성한다.

(그림 1)에서 보는 것처럼 메시지 다이제스트 함수를 적용해서 얻게 되는 메시지 다이제스트 값인 DigestedData1과 DigestedData2의 값이 동일하면 원본 데이터인 Data1과 Data2는 동일한 데이터로 간주할 수 있다. 이는 메시지 다이제스트 함수가 생성하는 결과 값이 입력 값에 따라 유일하기 때문이며 이를 위해서 메시지 다이제스트 함수는 안정성이 입증된 함수를 사용해야 한다, 즉 상이한 두 데이터가 다이제스트되었을 때 동일한 다이제스트 값이 생성되지 않아야 한다. 이러한 다이제스트 방법은 대량의 데이터가 소량의 데이터로 변경됨으로, 데이터의 내용 변형은 고려하지 않고 데이터의 변화 여부를 판단하는 방법으로 유용하다. 일반적으로는 메시지 다이제스트 방법을 보안 분야에서 전송되는 데이터간의 변조 여부를 판단하기 위해서 사용하지만 본 논문에서는 이 방법을 관계형 데이터베이스의 튜플 단위의 불일치 탐지를 위한 메시지 다이제스트 테이블 생성에 적용하였다.

3.3 다이제스트 테이블

데이터베이스 안의 릴레이션 즉, 테이블의 데이터 변화를 알기 위해서는 튜플과 애트리뷰트를 모두 식별할 수 있어야 한다. 클라이언트와 서버의 각각의 데이터 셀들을 직접 비교하지 않고도 메시지 다이제스트하여 튜플 단위로 클라이언트와 서버의 데이터베이스를 비교하는 것이 가능하다. 이 과정에서 메시지 다이제스트 테이블(MDT)

이 추가적으로 필요하다.

다이제스트 테이블은 데이터베이스안의 튜플들을 하나의 메시지로 보고 메시지 다이제스트를 적용한 결과값들을 저장하기 위한 추가적인 자료구조이다. 이는 이기종 데이터베이스안의 데이터나 이에 기반한 기존 응용프로그램의 수정없이 데이터베이스 상호간의 불일치를 탐지하고 이를 동기화시키기 위해 필요하다. 불일치 탐지는 서버쪽의 MD 테이블과 클라이언트쪽의 MD 테이블사이의 불일치 탐지를 의미한다. 먼저, 메시지 다이제스트 테이블(MDT)은 데이터 릴레이션 개수만큼 생성되는 tMDT(tuple Message Digest Table)과 데이터베이스당 하나만 존재하는 rMDT(relation Message Digest Table)로 분류할 수 있다. 먼저, 튜플 메시지 다이제스트 테이블인 tMDT 테이블의 스키마는 다음과 같다.

tMDT(pk_col, tMDV_col)

이때, tMDT 테이블의 pk_col은 대응되는 릴레이션의 기본키 컬럼값을, tMDV_col은 대응되는 릴레이션안의 튜플의 다이제스트값 tMDV를 저장하기 위해 사용된다. i번째 릴레이션 R의 j번째 튜플에 대한 tMDV(R_{ij})는 R_i에 대한 j번째 튜플의 애트리뷰트 값들(av_1, av_2, . . . , av_n)을 문자열 결합한 뒤, 이 값에 대해 MD4 알고리즘을 적용하여 얻은 128비트 해쉬 값을 32바이트 문자열로 표현한 값이다. 이를 tMDV를 사용하여 정의하면 다음과 같다.

$tMDV(R_{ij}) = hexstr(MD4(av_1 \oplus av_2 \oplus \ldots \oplus av_n))$
R_{ij} : i번째 릴레이션의 j번째 튜플
n: 애트리뷰트의 개수(릴레이션 R_i의 degree)
av_k : k번째 애트리뷰트값(이때, $av_k \in R_{ij}$)
$\oplus$: 문자열 결합 연산자
MD4(m) : 메시지 m을 다이제스트하는 함수
hexstr(h) : 128비트 해쉬값 h을 16진수 표기법에 따른 32바이트로 변환하는 문자열 함수

tMDV들은 마지막 동기화 이후 클라이언트와 서버쪽의 대응되는 릴레이션간의 튜플 단위의 불일치 유무를 탐지하기 위해 사용된다.

한편, 릴레이션 메시지 다이제스트 테이블인 rMDT 테이블의 스키마는 다음과 같다.

rMDT(tbl_name_col, rMDV_col)

이때, rMDT 테이블의 tbl_name_col은 데이터베이스내의 특정 릴레이션의 이름값을, rMDV_ col은 데이터베이스내의 특정 릴레이션안의 모든 튜플들의 다이제스트값 tMDV를 문자열 결합한 뒤 다시 다이제스트한 값을 저장하기 위해 사용된다. i번째 릴레이션 R_i에 대한 rMDV(R_i)는 R_i에 대한 m개의 튜플들의 tMDV값들(tMDV(R_{i1}), tMDV(R_{i2}), . . . , tMDV(R_{im})을 문자열 결합한 뒤, 이 값에 대해 MD4 알고리즘을 적용하여 얻은 128비트 해쉬 값을 32바이트 문자열로 표현한 값이다. 이를 rMDV를 사용하여 정의하면 다음과 같다.

$rMDV(R_i) = hexstr(MD4(tMDV(R_{i1}) \oplus tMDV(R_{i2}) \oplus \ldots \oplus tMDV(R_{im})))$
R_i : i번째 릴레이션
m: 튜플의 개수(릴레이션 R_i의 cardinality)
$\oplus$: 문자열 결합 연산자
MD4(m) : 메시지 m을 다이제스트하는 함수
hexstr(h) : 128비트 해쉬값 h을 16진수 표기법에 따른 32바이트로 변환하는 문자열 함수

이 rMDV들은 마지막 동기화 이후 클라이언트와 서버쪽의 대응되는 릴레이션간의 릴레이션 단위의 불일치 유무를 탐지하기 위해 사용된다. 동기화 이후 두 데이터베이스간의 변경(입력, 수정, 삭제) 빈도가 아주 낮은 경우에 대한 불일치 유무를 판단하는데 효율적이다.

다음 (그림 2)는 tMDV와 rMDV의 예를 보여준다. (그림 2)에서와 같이 STUDENT 릴레이션이 주어졌을 경우, STUDENT 릴레이션에 대응되는 tMDT 테이블이 하나 생성된 후, STUDENT 릴레

이션의 3개의 튜플 각각에 대한 tMDV값을 다음과 같이 구하여 저장한다.

예) STUDENT 릴레이션의 학번이 's001'인 첫 번째 튜플에 대한 튜플 메시지 다이제스트 값

tMDV = hexstr(MD4($av_1 \oplus av_2 \oplus \ldots \oplus av_n$))
= hexstr(MD4("HONG GIL DONG" ⊕ "MATHMATICS" ⊕ "SEOUL"))
= hexstr("11000010011001100000101010-10100110.....10100010000010101011")
= "C45308B335F77EEC893B6DAA9FC-2BAA20CD"

위와 같은 방법으로 (그림 2)에서 처럼 STUDENT 릴레이션의 모든 튜플에 대한 각각의 tMDV 값이 결정되면 이를 사용하여 STUDENT 릴레이션에 대한 rMDV 값을 다음과 같이 구하여 저장한다.

예) STUDENT 릴레이션에 대한 릴레이션 메시지 다이제스트 값

rMDV = hexstr(MD4(tMDV(R_{i1}) ⊕ tMDV(R_{i2}) ⊕ . . . ⊕ tMDV(R_{im})))
= hexstr(MD4("C45308B335F77EEC89-3B6DAA9FC2BAA20CD"⊕
"B048CC945FF8CEB3677CC3323CC822043A2"
⊕"788A72FFE2A8222CC151D8E5768AB4423FF3"))
= hexstr("00010001001100001011101100110011.....10101010010010000110110")
= "1130BB33CC7A72338C3BBD89441-122FD236"

STUDENT

SNO	SNAME	MAJOR	CITY
S001	HONG GIL DONG	MATHMATICS	SEOUL
S002	KIM HOON	ART	SUWON
S003	LEE MIN JAE	COMPUTER SCIENCE	DAEJEON

tMDT

pk_col	tMDV_col
S001	C45308B335F77EEC893B6DAA9FC2BAA20CD
S002	B048CC945FF8CEB3677CC3323CC822043A2
S003	788A72FFE2A8222CC151D8E5768AB4423FF3

rMDT

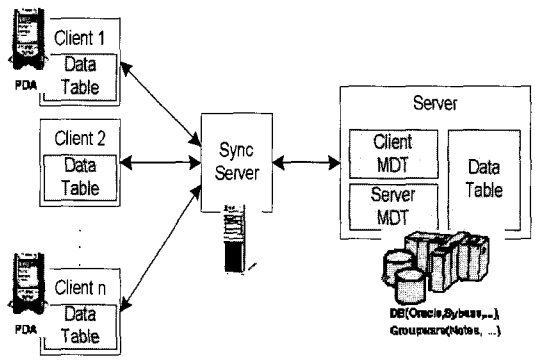
tbl_name_col	rMDV_col
STUDENT	1130BB33CC7A72338C3BBD89441122FD236
....	.....
....	.....

(그림 2) 메시지 다이제스트 테이블 구조

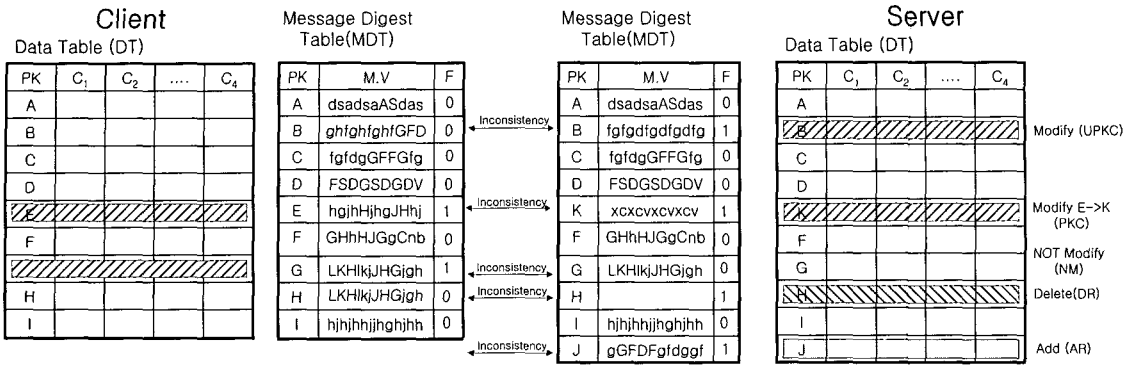
4. 데이터 동기화 알고리즘

모바일 데이터베이스 환경에서의 데이터 동기화는 (그림 3)과 같이 데이터 동기화 서버에 의해 우선, 각 모바일 데이터베이스의 변경된 내용을 데이터베이스 서버에 저장된 클라이언트 MDT 테이블에 반영한 뒤, 클라이언트와 서버 각 MDT 테이블의 변화를 탐지, 이를 일치시키는 과정으로 수행된다. 이를 위해 본 논문에서는 기본 키값을 식별자로 사용하는 A3 알고리즘과 고유한 식별자로 별도로 사용하는 A4 알고리즘을 제시하였다.

MDT 테이블의 구조는 알고리즘에 따라 조금씩 다를 수 있지만 기본적으로는 3.3절처럼 DT(Data Table)테이블의 기본키 컬럼과 다이제스트 값 컬럼으로 이루어진다. 이러한 클라이언트와 서버의 MDT의 다이제스트 값을 서로 비교함으로 데이터가 변경되었는지 여부를 판단할 수 있다. 변경 여부에 따른 데이터 값 연산(add, modify, delete)은 튜플 단위로 이루어진다. 애트리뷰트의 수가 많을



(그림 3) 모바일 데이터베이스의 동기화 개념



(그림 4) A3 알고리즘 적용 테이블 예

수록 알고리즘은 더욱 효과적으로 적용될 수 있다. 또한, 다이제스트 알고리즘은 매우 빠른 알고리즘으로 시스템에 주는 부담이 작다는 점에서 실제 모바일 데이터베이스내의 데이터 테이블의 애트리뷰트 개수와 다이제스트 알고리즘의 속도를 고려할 때 성능 측면에서 이점이 있다. 이때, MDT는 데이터 테이블(DT)과 쌍을 이루며 물리적으로는 분리되어 있을 수도 있다.

제안한 알고리즘은 클라이언트와 서버 각각에 튜플들에 대해 unchange, add, modify, delete 등의 4가지 경우를 처리해야함으로 (표 1)과 같이 하나의 튜플 당 총 16가지의 발생 가능한 경우를 고려해야 한다. 그러나, 이들 중 case 6, 7, 8, 10, 14는 동일한 튜플에 대해서 add 연산과 동시에 발생할 수 없는 비현실적인 경우이다. 따라서, 제안한 동기화 알고리즘에서는 이 경우들은 고려하지 않는다.

4.1 A3 동기화 알고리즘

제안한 A3 동기화 알고리즘은 다음과 같은 점을 가정한다. 첫째, 모든 테이블은 기본키를 가지고 있다. 둘째, 클라이언트와 서버에 새로운 데이터 삽입시 기존의 기본 키(PK)와 동일한 값을 갖지 않는다. 즉, 클라이언트와 서버간의 동기화 과정에서 PK에 의한 무결성 충돌이 일어나지 않는다. 셋째, DT의 PK와 MDT의 PK는 타입과 값이 모두 동일하다.

A3 알고리즘을 적용하기 위한 테이블 예는 (그림 4)와 같다. (그림 4)에서 MDT는 앞의 tMDT를 뜻하며, rMDT의 경우, 테이블 전체에 대한 변경 여부를 탐지하기 위해 사용하므로, 이 경우에는 이미 rMDT 비교를 통해 DT가 변경되었음을 가정하였다. MDT 테이블에는 편의상 PK, M.V 컬럼명을 사용하였고 추가로 1 비트의 플래그 컬럼을 두어 데이터의 변경 유무를 표시(0: unchange,

(표 1) 발생가능 연산들의 조합

C	client	server	C	client	server	C	client	server	C	client	server
1	unch	Unch	5	Unch	Add	9	Unch	Modify	13	Unch	Delete
2	Add	Unch	6	Add	Add	10	Add	Modify	14	Add	Delete
3	Modify	Unch	7	Modify	Add	11	Modify	Modify	15	Modify	Delete
4	Delete	Unch	8	Delete	Add	12	Delete	Modify	16	Delete	Delete

1: change)하였다. 각 MDT 테이블의 임의의 열(row)의 메시지 다이제스트 값이 동일하면 두 열의 데이터는 동일하다. (그림 4)에서 클라이언트와 서버쪽 데이터베이스에는 각각 MDT 테이블이 존재하며 초기에는 MDT 서로 간에 동기화가 이루어진 상태였지만 시간이 경과함에 따라 클라이언트 쪽에 새로운 튜플(PK가 E, G인 튜플)이 추가 되거나, 서버 쪽에 기존 튜플들 중에서 수정(PK가 B, E인 튜플)되거나, 삭제(PK가 H인 튜플) 혹은 새로운 데이터(PK가 J인 튜플)가 추가됨으로써 MDT간의 불일치가 발생했음을 보여준다.

제안한 A3 동기화 알고리즘은 클라이언트와 서버의 데이터가 모두 변경된 경우에는 정책 설정에 의해서 동기화 방향을 결정할 수 있다. 또한, 서버 데이터베이스와 클라이언트 데이터베이스에 대한 DT와 MDT사이의 동기화는 실질적인 동기화 작업의 사전 준비 과정에 해당함으로 동기화 작업과 분리되어 실행시킬 수 있어 동기화 운영의 융통성을 제공하고 동기화 시간을 단축시킬 수 있다.

즉, 클라이언트 쪽은 동기화 작업 이전에 미리 MDT와 DT의 동기화 작업을 실행하여 동기화의 부담을 감소시킬 수 있다. 서버 쪽은 관리자에 의해서 서버가 여유 있는 시간을 이용하여 MDT와 DT의 동기화 작업을 사전에 실시하여 동기화의 부담을 감소시켜야 한다. A3 알고리즘의 상당 부분이 MDT간에서 이루어지기 때문에 DT의 빈번한 접근에 따른 기존 서비스 속도 저하를 줄일 수 있다.

(그림 5)는 A3 알고리즘이 (그림 4)와 같은 클라이언트, 서버쪽 MDT간의 불일치 즉, 충돌이 발생했을 경우, 각 MDT에 튜플이 새로 추가되거나 기존 튜플이 수정, 삭제되는 경우들에 대해 순차적으로 테스트하고 각 튜플들을 고유하게 식별하는 기본 키값을 사용하여 동기화시켜가는 과정들을 보여준다. 이때, 동일한 튜플에 대한 수정이 클라이언트와 서버 양쪽에서 모두 발생한 경우에는 일반적으로 서버쪽을 중심으로 동기화를 수행한다.

4.2 A4 동기화 알고리즘

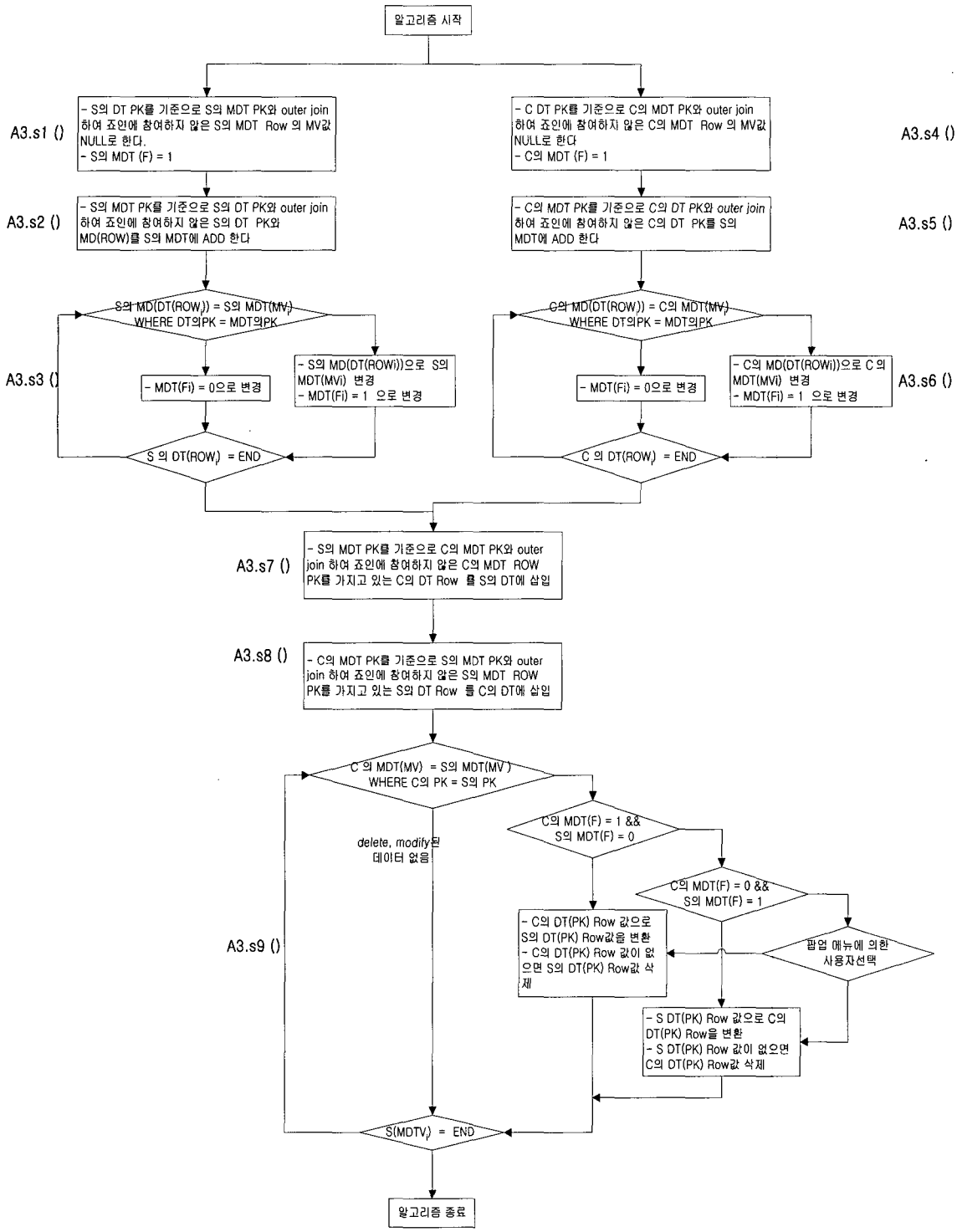
제안한 A4 동기화 알고리즘은 다음과 같은 점을 가정한다. 첫째, 모든 테이블은 기본키를 가지고 있다. 둘째, DT의 PK와 MDT의 PK는 타입과 값이 모두 동일하다. 셋째, MDT에 플래그 컬럼을 두어 데이터의 변동 여부를 표시(0: unchange, 1: change)한다. 넷째, MDT에는 DT의 PK값 외에 MDT의 PK인 자체 순차 번호를 가진다. 따라서, 순차 번호를 이용하여 두 MDT를 비교할 수 있다.

A4 알고리즘을 적용하기 위한 테이블 예는 (그림 6)과 같고 (그림 7)은 A4 알고리즘의 개략적인 내용을 보여준다. (그림 6)에서는 클라이언트와 서버쪽에서 PK를 포함한 모든 애트리뷰트의 수정이 발생할 수 있음을 가정하였고 이에 따라 MDT 테이블의 각 튜플들을 PK가 아닌 MDTV 일련번호를 통해서 식별하도록 확장되었다. 따라서, A4 알고리즘은 A3 알고리즘이 갖고 있는 기본 키값 변경 금지에 대한 제한점을 더 이상 갖지 않는다.

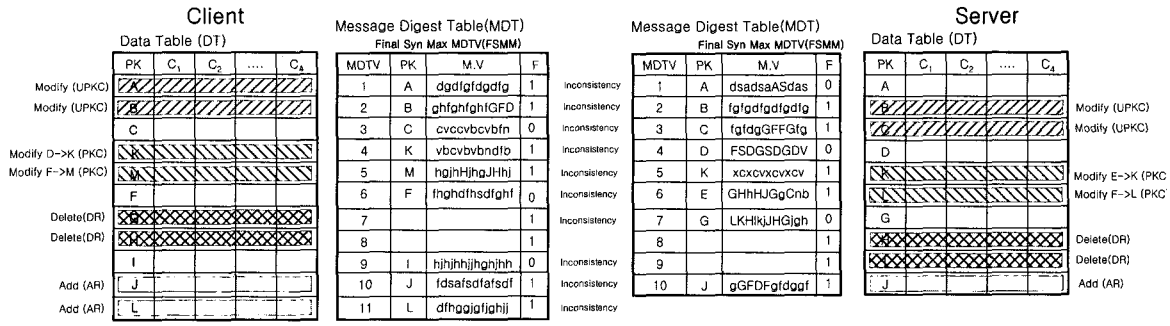
A4 동기화 알고리즘은 클라이언트와 서버에 새로운 데이터를 삽입할 때 동일한 PK값을 사용할 수 있다. 클라이언트와 서버간의 동기화 과정에서 PK에 의한 무결성 충돌을 처리하며 충돌이 발생하면 충돌 여부를 통보하고 다른 값을 입력하도록 요청한다. 한편, 클라이언트와 서버의 데이터가 모두 변경된 경우에는 정책 설정에 의해서 동기화 방향을 결정할 수 있다.

4.3 알고리즘 평가

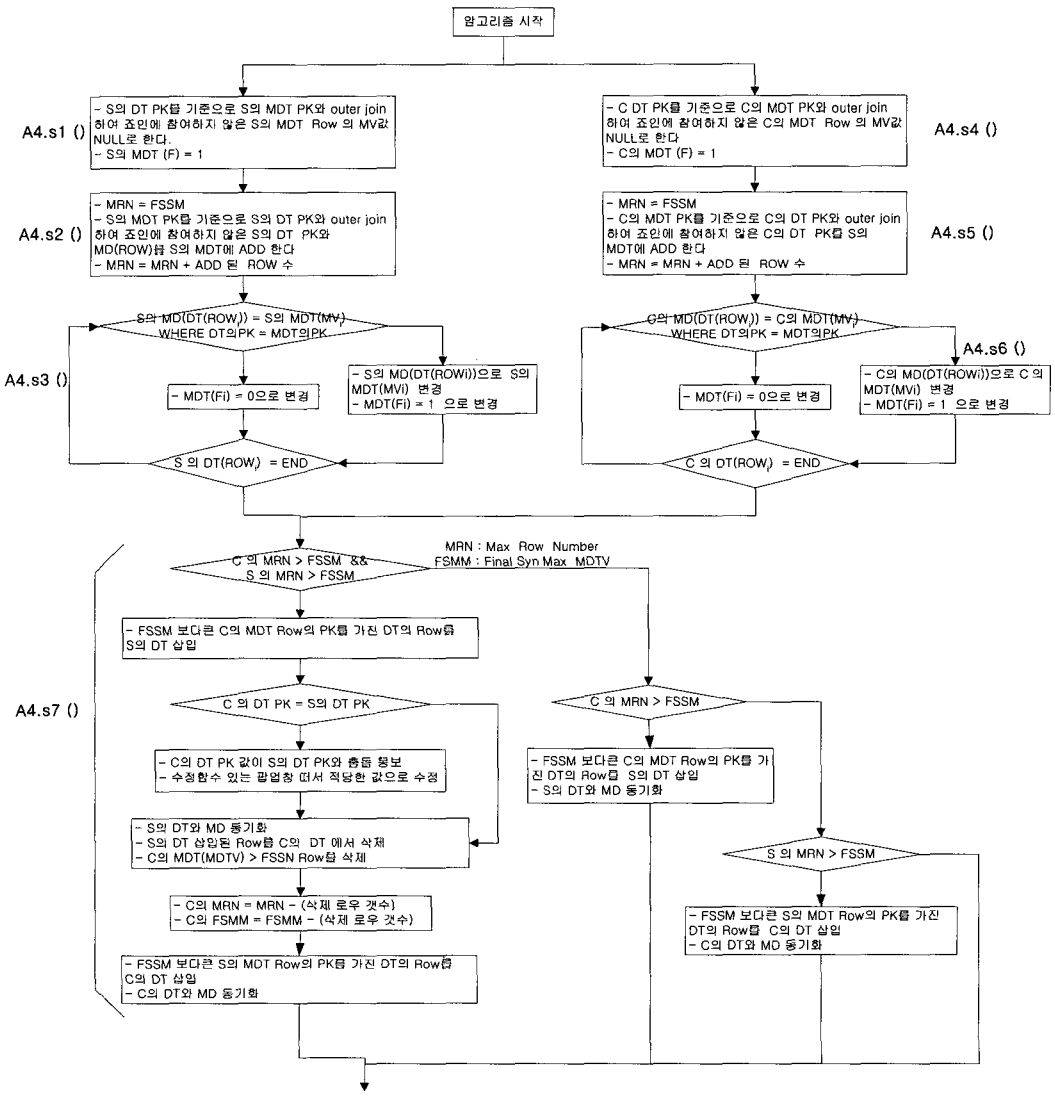
제안한 A3 알고리즘을 가상의 모바일 환경하의 클라이언트, 서버 데이터베이스를 대상으로 다음과 같은 조건에서 평가하였다. 테스트 조건은 5,000건의 튜플들에 대해서 테스트를 실시하였고 모든 테이블은 동기화 알고리즘이 실행되는 컴퓨터와 동일한 컴퓨터에 설치된 DB에 저장된다고 가정하였다. 테스트 데이터는 자체적으로 랜덤하

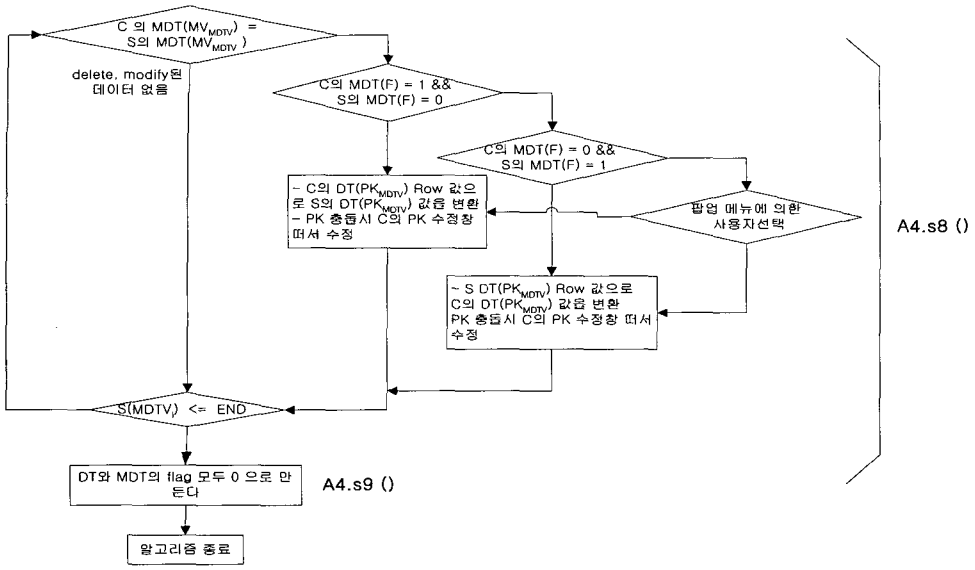


(그림 5) A3 동기화 알고리즘



(그림 6) A4 알고리즘 적용 테이블 예



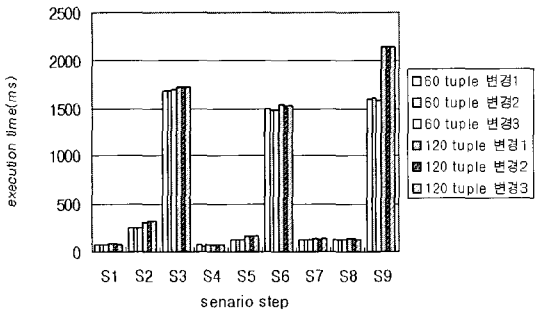


(그림 7) A4 동기화 알고리즘

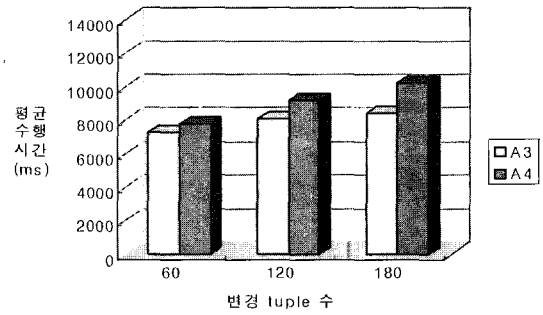
게 불특정 데이터를 생성하여 변화된 튜플의 수를 조절해 가면서 3회씩 평가하였다. 먼저, 알고리즘을 구현하고 그 평가 결과를 통해 변경된 튜플들에 대한 변경 탐지와 그에 따른 데이터 동기화가 완전하게 수행됨을 검증하였다.

제시한 A3 알고리즘에 대해 60개, 120개씩 튜플을 변경할 경우의 실행 시간측면에서 테스트한 결과는 (그림 8)과 같다. (그림 8)은 60개 튜플을 변경한 경우와 120개 튜플을 변경한 경우에 대한 A3 알고리즘 실행 시간을 평가한 결과이다. 이때, 서버와 클라이언트 쪽에서는 각각 add, modify, delete가 각기 10 혹은 20건이 발생했다고 가정하였다. 그 결과, 60개의 튜플을 변경한 경우와 120개의 튜플을 변경한 경우 모두 거의 유사한 형태의 실험 결과를 보여준다. 즉, 변경된 튜플의 개수에 알고리즘 실행시간이 크게 영향 받지 않음을 보여준다. (그림 8)에서 메시지 다이제스트 테이블에 대한 참조와 비교가 직접적으로 이루어지는 S3, S6, S9 단계가 상대적으로 많은 실행시간을 소요함을 알 수 있다.

(그림 9)는 60개, 120개, 180개 튜플들을 변경했을 경우의 A3, A4 알고리즘의 각 실행시간을 상대적으로 평가하여 보여준다. 실험 결과에서 보는 바와 같이 제시한 A3, A4 알고리즘 모두 변경된 튜플의 개수에 따른 큰 차이를 보이지 않고



(그림 8) A3 각 단계별 실행시간



(그림 9) A3, A4 실행시간 비교

변경된 튜플 개수에 따라 선형증가함으로써 모바일 환경에서 데이터베이스 간에 다량의 데이터가 변경되더라도 적용 가능함을 보여준다. 결과적으로 알고리즘 평가를 통해 제시한 메시지 다이제스트를 사용한 동기화 기법이 각 단계별로 완전하게 동작함을 검증하였고 다량의 데이터가 변경되는 상황에도 실제로 적용할 수 있는 동기화 기법임을 확인하였다.

5. 결 론

모바일 데이터베이스 환경에서는 다양한 이기종 데이터베이스가 사용될 수 있다. 그러나, 현재 상업용 데이터베이스 시스템의 경우에는 DBMS마다 별도의 동기화 방식을 이용하기 때문에 제한된 환경에서는 우수하지만 이기종 데이터베이스간의 동기화에는 많은 어려움이 발생하고 있다. 본 논문에서는 이기종 모바일 데이터 동기화를 위해 추가적인 보완이나 확장없이 동기화를 수행할 수 있는 DBMS 독립적인 범용 데이터 동기화 기법을 제안하였다.

제안한 데이터 동기화 기법은 튜플 단위의 메시지 다이제스트 결과를 충돌을 탐지하기 위한 정보로 사용함으로써 기존 동기화 방법에 비하여 DBMS 독립적이면서 비교적 적은 동기화 정보를 사용한다는 점에서 의의가 있다. 제안한 데이터 동기화 기법은 다양한 모바일 환경에서 중복된 데이터간의 일치를 요구하는 많은 응용 프로그램을 위해서 사용될 수 있다.

참 고 문 헌

[1] Michael A. Olson, "Selecting and Implementing an Embedded Database System," IEEE Computer, Vol. 33. No. 9, pp. 27-44, 2000. 9.

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

[3] S. S. Chawathe and H. Garcia-Molina. Meaningful Change Detection in Structured Data. In Proc. Of SIGMOD '97, pp.26-37, 1997.

[4] K. Zhang, J. T. L. Wang, and D. Shasha, On the editing distance between undirected acyclic graphs and related-problems. In Proc. of the 6th Annual Symposium on Combinatorial Pattern Matching, pp.395-407, 1995.

[5] S. J. Lim and Y. K. Ng, An Automated Change-Detection Algorithm for HTML Documents Based on Semantic Hierarchies, In Int'l Conf. on Data Engineering, pp.303-312, 2001.

[6] G. Cobena, S. Abiteboul and A. Marian, Detecting Changes in XML Documents, In Int'l Conf. on Data Engineering, pp.41-52, 2002.

[7] W. Labio and H. G. Monila, Efficient Snapshot Differential Algorithms for Data Warehousing, In Proc. Of the 20th VLDB Conf., pp. 63-74, 1996.

[8] Oracle, Oracle 9i Lite Administration and Development Guide 5.0.2, http://www.oracle.com/technology/documentation/oracle9i_arch_901.html, 2004.

[9] IBM, DB2 Sync Server Administration Guide 7.2.1, www.decus.de/slides/sy2002/16_04/1L02.pdf, 2004.

[10] Sybase, Synchronization Technologies for Mobile and Embedded Computing, www.ianywhere.com/downloads/whitepapers/mobilink_sql.pdf, 2004.

[11] Rivest, R. , The MD4 Message Digest Algorithm, RFC 1320, MIT and RSA Data Security, Inc., http://www.rfc-archive.org/getrfc.php?rfc=1320

◐ 저 자 소 개 ◑

박 성 진(Seong-Jin Park)
1991년 고려대학교 전산학과 졸업(학사)
1993년 고려대학교 대학원 전산과학과 졸업(석사)
1998년 고려대학교 대학원 전산과학과 졸업(박사)
2000~현재 한신대학교 컴퓨터정보소프트웨어학부 교수
관심분야 : 데이터베이스, 데이터웨어하우징, 데이터 마이닝
E-mail : sjpark@hs.ac.kr