중복 데이터베이스 시스템에서 갱신그래프를 이용한 동시성제어

최 희 영[†] · 이 귀 상[††] · 황 부 현[††]

요 약

중앙집중형 데이터베이스는 데이터 관리가 용이하나 통신장애나 사이트 고장시 전체시스템 중지로 인해 신뢰성과 가용성문제가 발생한다. 이러한 문제를 해결하기 위해서 중복데이터베이스가 출현되었다. 그러나 갱신거래가 많이 발생되는 경우에는 중복 데이터에 대해서 갱신이 동일하게 이루어져야 하므로 동기화로 인한 메시지부담과 거래가 지연됨에 따라 동시성이 감소되는 문제가 발생하게 된다. 이 논문은 가용성과 신뢰성을 향상시키기 위한 완전 중복 데이터베이스에서 거래들의 병행성의 정도를 높이기 위한 동시성제어 알고리즘을 제안한다. 중복 데이터베이스에서 시스템 성능을 향상시키기 위해서는 거래가 제출된 사이트에서 마지막 연산까지 수행시키고, 기록 연산들로 구성된 갱신 전용거래를 모든 사이트에서 독립적으로 수행하도록 한다. 각 사이트에서 수행되는 갱신전용거래의 일관성은 모든 사이트에서 갱신그래프를 유지하여 보장한다. 제안하는 동시성 제어 기법은 각 사이트에서 거래들이 동시에 수행할 수 있게 함으로써 거래들의 병행수행정도를 향상시킬 수 있다. 제안하는 동시성 제어 기법의 실제 구현 및 실험을 통한 결과를 분석하여 기존의 방법보다 보다 더 빠른 응답률과 더 적은 철회율를 가져옴을 성능 평가를 통해 우수성을 보여준다.

Concurrency Control Using the Update Graph in Replicated Database Systems

Hee-Young Choi[†] · Guee-Sang Lee[††] · Bu-Hyun Hwang[††]

ABSTRACT

Replicated database system was emerged to resolve the problem of reduction of the availability and the reliability due to the communication failures and site errors generated at centralized database system. But if update transactions are many occurred, the update is equally executed for all replicated data. Therefore, there are many problems the same thing a message overhead generated by synchronization and the reduce of concurrency happened because of delaying the transaction. In this paper, I propose a new concurrency control algorithm for enhancing the degree of parallelism of the transaction in fully replicated database designed to improve the availability and the reliability. To improve the system performance in the replicated database should be performed the last operations in the submitted site of transactions and be independently executed update-only transactions composed of write-only transactions in all sites. I propose concurrency control method to maintain the consistency of the replicated database and reflect the result of update-only transactions in all sites. The superiority of the proposed method has been tested from the respondence and withdrawal rate. The results confirm the superiority of the proposed technique over classical correlation based method.

키워드 : 중복 데이터베이스(Replicated Database), 동시성제어(Concurrency Control), 갱신 그래프(Update Graph)

1. 서 론

분산시스템 환경에서는 여러 사용자가 동일한 데이터에 질의나 갱신을 위하여 동시에 접근하는 경우가 많이 발생한다. 이러한 문제의 처리는 중앙집중 데이터베이스 환경에서의 동시성제어 방법론의 발전을 통하여 상호배제 규칙을 위반하지 않고 동시 다발적 접근을 가능하게 하는 방식으로 이루어졌다.

그러나 분산 네트워크를 경유하여 접근되는 중앙집중 데이터베이스는 통신 장애나 사이트 장애시 전체 시스템 사용을 중지해야 하는 큰 약점을 가지고 있다. 장애에 의하여 데이터베이스를 저장하고 있는 하나의 사이트가 전체 시스템으로부터 단절이 된다면, 그 사이트의 데이터베이스는 접근이 불가능하게 된다. 이러한 이유 때문에 데이터의 중복을 허용하는 중복 데이터베이스 개발의 필요성이 대두되는 것이며, 고장을 다루는 직관적인 방법이다.

데이터베이스 시스템에서 데이터를 중복시킴에 따라 얻

※ 본 연구는 2002년도 두뇌한국21사업 지원에 의하여 수행되었습니다.
† 정 회 원 : 전남대학교컴퓨터 정보학부 시간강사
†† 종신회원 : 전남대학교컴퓨터 정보학부 교수
논문접수 : 2002년 4월 26일, 심사완료 : 2002년 6월 14일

어지는 장점은 다음과 같다. 첫째, 가용성과 신뢰성을 증가한다. 둘째, 성능이 향상된다. 셋째, 병렬처리가 가능하다. 즉 판독만을 요구하는 거래인 경우는 여러 사이트에서 병렬로 처리할 수 있다[5, 6]. 이러한 장점이 있음에도 불구하고 모든 사이트를 동기화시켜야 하는 지나친 비용과 데이터베이스를 접근하는 거래들이 분산된 교착상태에 빠질 수 있다. 따라서 중복 데이터베이스는 갱신 요구가 많은 응용보다는 검색 요구가 많은 응용에 적합하다. 현재 응용에서는 높은 성능과 높은 가용성이 있는 데이터베이스를 요구하므로 이러한 성질을 지원하기 위한 새로운 메커니즘이 출현하게 되었다. 그러나 상업 데이터베이스 회사들은 이러한 요구를 만족하는 약한 일관성을 제공하는 솔루션에 초점을 맞추었다. 약한 일관성을 제공하는 것은 직관적이질 못하고 사용하기가 매우 어려우므로 엄격한 규약 명세와 엄정한 정확성을 뒷받침 해 줄 수 있는 이론적인 배경이 부족하다. 따라서 데이터가 중복된 모든 사이트에서 데이터의 일관성을 유지하면서 효율적으로 동작하는 거래 관리를 위한 갱신 규약을 설계하는 일은 매우 중요하다[1, 3, 12, 18, 19].

중복 데이터베이스 시스템에서 사용되고 있는 거래 관리는 크게 Eager 규약[17, 25]과 Lazy 규약으로 분류할 수 있다. Eager 규약은 갱신연산의 경우 모든 해당사이트의 갱신연산이 하나의 거래에 의해서 이루어져야 하기 때문에 거래의 수행시간이 길어 거래 처리비용이 증가하고 거래의 대기시간이 길어지는 문제가 발생한다. 즉 중복된 모든 사이트에서 거래를 수행하는 동안 모든 중복 데이터베이스에 있는 데이터 아이템을 갱신하기 위해서는 원격 통신이 이루어져야만 하기 때문에 원격통신으로 인한 오버헤드를 줄이고, 일관성 유지를 위한 거래 처리가 필요하다. 특히 데이터가 중복된 모든 사이트에 대한 동기화를 위해서 잠금 기법을 적용한다면 잠금으로 인한 동시성 접근이 어려워 거래가 지연되는 문제가 발생된다[7].

Lazy 규약은 지역사이트에서 거래를 수행하고 완료한 후에 비 동기적(asynchronous)인 방법으로 데이터가 중복된 모든 사이트로 갱신연산을 전파하여 수행시키는 규약이다. 만약 동시에 각 사이트로 접근되는 거래가 있다면 중복되는 데이터 항목에 대해서 데이터의 일관성을 위반할 수 도 있다[4, 13]. 즉 거래가 제출된 사이트에서 거래를 수행하는 동안에는 다른 중복 사이트로 원격 통신을 요구하지 않으므로, 지역적으로 갱신된 연산들에 대해서는 모든 중복 사이트에서 동일하게 수행될 수 있도록 하며, 통신 트래픽(traffic)이 없을 때만 모든 사이트에서 거래가 갱신될 수 있도록 하는 약한 일관성만을 제공한다. 따라서 중복 데이터 베이스에서 Lazy 규약은 성능은 좋지만 너무 무질서하게 사용한다면 직렬성을 파괴할 수도 있기 때문에 강한 일관성을 필요로 하는 애플리케이션에는 적합하지 않다[4, 9]. 따라서 여러 사이트에서 동일한 데이터 항목에 대하여 동시에 접근하는 거래의 전역적 일관성에 대한 보장이 어려우므로 중복 데이터베이스를 가지는 모든 사이트에서 수행되는 거래들의 직렬성을 제공하는 효율적인 검증 절차가 필요하다[1].

이 논문에서 제안하고 있는 거래 관리 기법은 일관성을 중요시하는 Eager 규약을 기반으로 하며, 중복된 모든 사이트에서 병행성을 향상시키고 거래의 정확한 수행을 보장할 수 있는 동시성 제어 기법으로 데이터의 일관성과 거래의 성능을 향상시키기 위한 방법에 초점을 두고 있다.

이 논문에서는 기존의 분산 데이터베이스 환경에서 사용되는 2PL이나 타임스탬프를 이용한 방법이 아닌 새로운 잠금 규약을 이용한 동시성제어 기법을 제안한다.

중복 데이터베이스에서 수행되는 거래는 ROWA기법[9]을 사용해서 각 사이트에 제출된 거래를 그 지역 데이터 베이스에서 독립적으로 수행하도록 하고, 기록연산들은 그 거래의 작업영역에서 수행한 후에 갱신 규약을 사용하여 데이터가 중복된 모든 사이트에서 갱신이 일어나도록 함으로서 거래의 병행 수행정도를 향상되도록 한다. 중복 데이터베이스 간에 전역적 일관성 문제를 해결하기 위하여 데이터를 중복하고 있는 모든 사이트에서 데이터 항목의 충돌을 검사하는 갱신 그래프(Update Graph : UG)를 각 사이트에서 유지하도록 하여 데이터의 일관성을 유지하도록 한다. 또한 제안한 거래 관리기법이 직렬성을 만족하는가에 대해 정확성 기준으로 1-사본 직렬성(one-copy serializability)을 보장할 수 있음을 보인다.

이 논문에서 제안한 동시성 제어기법과 기존의 방법을 비교한 성능분석을 다루고 있다. 성능평가는 CSIM(C SIMulation)을 이용하여, PentiumIII에서 Linux OS에서 수행한다. 성능평가의 기준은 판독전용 거래의 비율에 따라 응답시간과 철회율을 비교한다.

이 논문의 구성은 다음과 같다. 2장에서는 중복 데이터베이스 시스템 환경에서 수행되는 거래를 관리하는 기법과 중복 데이터베이스에서 데이터의 일관성을 유지하기 위한 기존 연구에 대하여 기술하고, Eager 규약에 의해서 기존에 연구된 문제점에 대한 예를 들어 설명한다. 3장에서는 이 논문이 기반으로 하고 있는 시스템 환경을 기술한다. 4장에서는 이 논문이 제안하는 알고리즘을 기술하고, 제안하고 있는 알고리즘이 직렬가능한 수행을 보장함을 증명한다. 5장에서는 기존 DWB방법과 제안한 방법을 평가 및 분석하고, 6장에서는 결론 및 추후 연구 방향에 대해 기술한다.

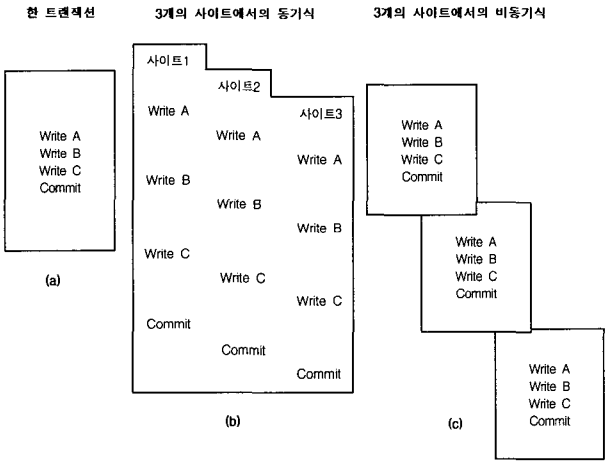
2. 관련 연구 및 문제분석

2.1 관련 연구

이 장에서는 중복 분산 데이터베이스 시스템 환경에서 수행되는 거래 관리 기법과 중복 데이터 베이스에서 데이터의 일관성을 유지하기 위해서 다룬 기존의 연구들에 대해

살펴본다.

분산 데이터베이스 환경에서 중복 데이터베이스의 관리는 전통적으로 동기식제어 기법과 비동기식 제어 기법이 있다. 동기식 제어 기법은 중복 데이터베이스에서 갱신 연산을 모든 중복된 복사본 사이트에서 수행해야 할 때, 거래의 완료 시점을 모든 복사본 사이트에서 동기화 시켜서 갱신 연산을 수행하도록 한 후 완료시키는 제어 기법이다. 즉 한 거래 내에서 중복된 모든 사이트에서 데이터를 갱신시키고 완료시키는 기법으로 Eager 규약을 사용해서 동시성제어를 한다. 비동기식 제어기법은 중복 데이터 베이스에서 거래가 제출된 사이트에서 먼저 거래를 수행하고 완료한 후 통신의 병목현상이나 오버헤드가 없을때 다른 복사본 사이트로 연산들을 전파하여 거래를 수행하는 기법이다. 이 기법에는 거래를 제출한 사이트에서 일단 수행하고 완료한 후 데이터가 중복된 모든 사이트로 전파해서 수행시키는 Lazy 규약이 있다[1, 3, 7, 9]. 이들의 수행과정을 살펴보면 다음 (그림 2.1)과 같다.



(그림 2.1) 중복 데이터베이스에서 거래 수행 구조

(그림 2.1)에서 세 개의 사이트에 데이터 A, B, C가 중복 저장되어 있을 때 (a)와 같은 거래를 수행한다고 하자. 동기식 제어 기법은 각 연산들을 수행할 때 (b)와 같이 세 사이트를 동기화 시켜서 수행하고, 비동기식 기법은 한 사이트 내에서 직렬성을 파괴시키지 않고 수행시킨 다음 다른 복사본 사이트로 전파하여 갱신 연산을 실행시키는 기법으로 (c)와 같다.

동기식 제어 기법을 대표하는 규약인 Eager 규약은 거래를 완료(commit)하기 전에 중복된 모든 사이트에서 동기화시키는 방법으로 강한 일관성을 보장하고 견고성의 정도가 좋다. 1-사본 직렬성[2]의 기준으로 거래의 정확성을 보장하고 있다. 그러나 통신 부담과 응답시간이 너무 길어 거래가 지연되는 문제가 발생된다. 이 규약은 데이터 항목을 갱신하는데 갱신 연산을 주 사본에서 실행하는 기법과 주 사본이 아닌 임의 사이트에서 갱신이 가능한 제어기법이 있다. 갱신연산을 수행하는 위치에 따라 Eager Primary Copy제어방법[7]과 Eager Update Everywhere방법으로 구분한다[16].

Eager Primary Copy 제어기법은 하나의 주 마스터 사본과 여러 개의 부 사본을 두어서 클라이언트가 각 지역사이트로 제출된 거래를 수행하기 위하여 판독연산은 거래가 발생된 사이트에서 수행하고 갱신연산은 주 마스터 사본을 가지는 사이트에서만 수행하는 기법이다. 충돌하는 연산들의 순서는 주사본 사이트에서 결정되고 주 사본의 결정에 따라 부 사본 사이트에서 거래의 순서가 결정된다. 따라서 거래의 동시성이 떨어지고 성능이 좋지 않다. Eager 규약에서 동시성 제어를 하는 방법으로 잠금 기법을 사용한 제어기법은 전역 데드록 문제를 해결해야 하고 병목 현상이 발생되는 문제점을 가지고 있으나 거래의 강한 일관성을 제공한다. 이 모델은 분산 컴퓨팅의 초기 솔루션[7, 15]으로 사용되었으나 현재는 예비(backup)메커니즘에 의해서 견고성을 위해서만 사용되고 있다. 예를 들어 은행업무에서 고객들의 모든 마스터 파일을 중앙 전산실에서 관리했을때 중앙전산실의 데이터베이스 서버가 고장이 나서 모든 은행업무가 마비되어 버린다면 은행은 고객들에게 질적인 서비스를 제공하지 못할 것이다. 따라서 이러한 비상사태를 대비해서 데이터베이스를 완전중복 시켜 놓고 두 시스템이 듀얼로 수행될 수 있도록 한다.

Eager Update Everywhere 제어기법은 주 사본 제어기법에서 발생되는 병목현상을 해결하기 위해 주 마스터 사본에서만 거래를 갱신시키지 않고 거래가 발생된 모든 사이트에서 갱신을 수행할 수 있도록 하는 기법이다. 이 기법은 ROWA 기법을 사용하여 판독은 거래가 발생된 사이트에서 지역적으로 수행하고 갱신은 모든 사이트를 동기화 시켜서 수행한다.

전통적인 Eager 중복 규약은 한번에 하나의 연산을 동기화 함에 따라 동기화 오버헤드가 발생한다. 즉 n개의 사이트에서 수행되는 거래들이 k개가 있고 각 거래들은 m개로 구성되어 있다면 초당 $n*m*k$메시지를 요구한다. 이러한 문제를 피하기 위해서 기록연산을 하나의 기록연산 집합으로 묶어서 하나의 메시지로 나타내었다. 이러한 원시적인 갱신 규약과 거기에서 발생된 문제를 해결하는 규약을 [20]에서 제시하고 성능분석을 하였다. 첫 번째, 원시적인 가장 단순한 규약인 BA(Broadcast All)는 중복 데이터베이스의 벤치마크로서 이용되고 있는데, 이 기법은 거래의 모든 판독과 기록 연산들을 데이터가 중복된 사이트에 방송하여 사이트를 동기화 시켜서 전통적인 엄격한 2PL제어 기법에 의해서 스케쥴하였다. 이는 판독과 기록잠금을 모든 사이트에 동기화시키는 부담과 잠금을 설정하고 해제하기 위한 메시지의

부담이 많을 뿐만 아니라 거래의 지연으로 인한 동시성을 저하시키는 문제를 발생시킨다.

두 번째 규약은 ROWA 기법을 사용한 BW(Broadcast Write)규약이다. ROWA 기법에서 판독연산은 거래가 제출된 사이트에서만 판독 잠금을 얻은 후에 지역적으로 수행하고 기록 연산은 모든 복사본 사이트로 방송해서 기록잠금을 얻은 후에 수행하도록 하는 기법이다. 이 기록잠금은 독점잠금이므로 데이터 x에 대한 잠금을 잠금 관리자에게 요구하면, x의 사본을 포함하는 모든 사이트내의 잠금 관리자에게 x에 대한 잠금 요청을 동기화 시키고 잠금 요청이 승인될 때까지 이 기록연산은 대기하므로 거래가 지연된다. 그리고 이 방법 또한 판독연산과 기록 연산 사이에 전역 교착 상태를 발생시키고 지역거래와 갱신 거래사이에 충돌이 발생되면 어느 한쪽의 거래를 대기시키거나 철회시켜야 한다.

세 번째 규약은 갱신 거래를 완전히 지역적으로 실행하는데 목적을 두고 기록연산을 지연시켜 하나의 기록연산 집합인 메시지를 해당사이트에 방송하여 데이터를 갱신시키는 규약인 DWB(Delayed Write Broadcast)가 있다. 즉 판독연산을 지역적으로 수행시키고 기록연산을 완료시점까지 지연시킴에 따라 거래 수행의 제한을 더 완화시키고 병행성을 증가시키기 위하여 제안된 규약이다. 그러나 클라이언트가 완료를 요구했을 때 기록 잠금을 얻어서 수행하고 그 거래가 완료상태에 도달 할 때까지 잠금을 해제시키지 않기 때문에 동일한 데이터 항목에 대해서는 접근할 수 없으므로, 앞에 기술된 규약보다는 동시성을 증가시킬 수 있으나 근본적인 잠금에 대한 거래 지연은 발생된다.

위의 3가지 갱신 규약들은 지역 동시성 제어를 엄격한 2PL 기법을 사용하기 때문에 잠금의 해제는 거래의 마지막 단계에서 이루어진다. 따라서 2PL에서 발생되는 교착상태의 해결은 전통적인 교착상태 해결 방법을 적용해서 해결할 수 있다. 그러나 잠금으로 인한 지연문제는 기록 잠금을 완료시점까지 지연시켜서 실행함에도 불구하고 데이터가 중복된 환경이고 각 사이트에서 동시에 갱신거래가 발생했을 때 병행성이 떨어지는 문제가 발생하므로 이를 해결할 수 있는 효율적인 동시성 제어기법을 개발하는 연구가 필요하다. 다른 형태의 접근방법은 원자적 방송 프리미티브가 있다. 이 방법을 사용한 [7]에서는 분산 시스템의 견고성을 증가시키기 위해서 근거리 통신망으로 국한된 그룹통신 프리미티브를 제안하였다. 이 프리미티브는 원자적 방송 프리미티브의 시멘틱을 사용하였으나 성능을 향상시키기 위한 구현 전략과 기본적인 병목현상으로 인하여 메시지를 모든 사이트에 전달하기 전에 조정단계가 필요하고, 이 조정단계가 끝나기 전까지는 메시지를 전달할 수 없으므로 메시지가 지연되는 문제가 발생하게 된다. 이를 해결하기 위해서 [8]에서는 갱신거래를 중복 사이트에서 즉시 전달하고, 수행한 후에 전달 순서대로 수행을 하였는가 즉 그 거래들의 전역 순서가 존재하는지의 여부를 검사하는 낙관적인 방법을 제안하였다. 이 방법을 확장하고 발전시킨 방법으로 [6]에서는 원자적 방송의 조정단계가 거래의 수행과 병행하여 이루어지게 함으로서 성능을 향상시킬 수 있는 방법을 제안하였다. 이 방법은 방송한 순서로 각 사이트에 전달된다는 가정 하에 해당사이트에서 갱신 거래들을 수행시키고 방송 순서대로 전달되어 수행되었는가를 검사하는 낙관적인 방법이다. 여기에서 제안된 방법은 제출된 사이트에서 판독연산을 수행하고 기록 연산을 모든 사이트에서 수행하게 하는 방법보다 판독과 기록 연산 모두가 중복 사이트에서 수행되어야 하기 때문에 중복데이터베이스의 특징중의 하나인 병행성이 저하된다. 또한, 각 사이트에서 동시에 수행되는 갱신거래의 수가 많을 경우에 거래들의 잠정수행순서(tentative order)와 확정수행순서(definitive order)가 달라질 가능성이 많아 철회되는 거래의 수가 많아지게 된다.

중복 데이터베이스 시스템 환경에서 원자적 방송을 기반으로 하는 거래 처리에서 가장 큰 문제가 되고 있는 것은 갱신 메시지를 보내는 순서대로 처리해야 하는 것이다. 이 부담 때문에 모든 사이트가 갱신메시지를 보내기 전에 사이트들 사이에 조정을 해야 한다. 이때 하나의 확정 수행순서를 만들기 위해서 주고 받는 메시지의 부담도 클 뿐만 아니라 통신비용도 많이 든다.

비 동기식 제어 기법인 Lazy 규약은 데이터를 중복하고 있는 서버 사이에 어떤 조정을 하기 전에 클라이언트에게 응답을 제공함에 따라 Eager 중복제어 기법에서 발생되는 동기화 오버헤드를 피하기 위하여 제안된 규약이다. 거래의 수행은 중복을 제어하는 서버에서만 수정이 일단 완료되면 그 거래의 완료를 허용하며, 이후 적당한 시기에 서버의 책임하에 수정전파 통신을 수행하는 방법[2]으로 약한 일관성을 보장하고, 각 사이트가 완료하는 시점이 서로 다르므로 전역 일관성을 유지하는 문제가 발생하게 된다. 따라서 이 논문에서는 비 동기식 제어 기법은 고려하지 않는다.

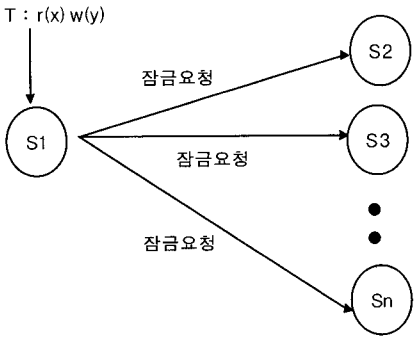
2.2 문제 정의

이 장에서는 중복 데이터베이스에서 Eager를 기반 한 동시성제어 기법에서 발생되는 메시지 부담과 잠금으로 인한 거래 지연문제, 완료되어야 할 거래가 철회되는 문제점들을 기술한다.

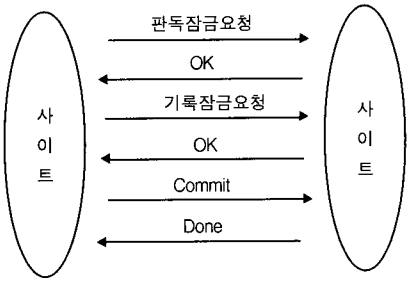
2.2.1 동기화로 인한 메시지 부담과 거래 지연

Eager 규약에서 BA기법은 사이트에 제출된 거래를 모든 사이트에 연산들에 대한 잠금을 요청해서 전역적으로 동기화 시켜서 수행하는 방식이다. 앞에 관련 연구에서 기술된 바와 같이 판독연산 또는 기록 연산을 하기 위해 접근되는 데이터에 대하여 모든 사이트에 부가적인 잠금 요청메시지를

방송한 다음 잠금을 해제시키는 메시지를 방송한다. 이의 관계를 그림으로 표현하면 다음 (그림 2.2), (그림 2.3)과 같다.



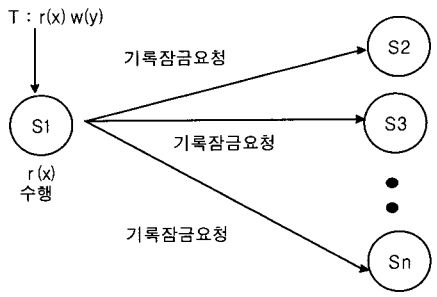
(그림 2.2) BA 기법의 거래수행



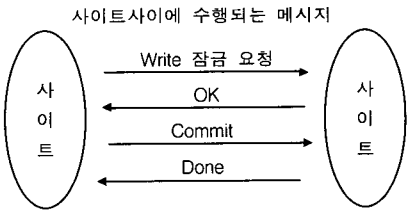
(그림 2.3) 부가적인 메시지 교환

거래 T가 사이트 S_1에 제출되면 T의 연산을 수행하기 위해서 모든 사이트에 잠금을 요청하여 잠금 요청이 승인되면 사이트 S_1에서 판독연산을 수행시키고, 기록 연산은 모든 사이트에서 실행시킨다. (그림 2.3)은 각 사이트들 사이에 주고 받는 메시지 교환을 보여주고 있다. 따라서 BA 기법은 잠금으로 인해 교환되는 메시지 부담과 통신 부담, 잠금으로 인한 거래지연과 교착상태가 발생하게 된다. 이는 중복 규약의 부담을 증가시켜서 동시성을 저하시키는 문제가 발생된다. 교착 상태의 해결은 각 사이트에서 동일한 메커니즘을 적용해서 해결되어야 한다.

BA에서 발생되는 잠금 요구로 인한 부가적인 메시지 교환에 따른 문제를 해결하기 위해서 제안된 BW(Broadcast Write)방법은 모든 연산을 전역적으로 수행시키는 부담을 줄이기 위해서 판독연산을 지역적으로 수행하고, 기록연산은 전역적으로 동기화 시켜서 수행시키는 방법이다. 따라서 판독연산에 대한 잠금 요청 메시지부담은 발생되지 않지만 이 방법 역시 기록연산에 대한 메시지 부담은 그대로 남아 있고 각 사이트에서 제출된 거래를 독립적으로 수행할 수 없기 때문에 거래의 병행성을 향상시킬 수 없다. 또한 모든 사이트의 기록잠금 동기화로 인한 잠금 소유시간이 길어서 동시성이 저하되는 문제를 가진다. 이를 그림으로 나타내면 다음 (그림 2.4), (그림 2.5)와 같다.



(그림 2.4) BW기법의 거래수행



(그림 2.5) 사이트들 사이의 메시지전송

BW 방법은 거래의 갱신연산을 수행하기 위해서 기록 잠금을 요청하는데 사이트 S에서 판독잠금과 충돌이 발생하면 이 판독 잠금을 가지는 거래가 이미 방송을 되었는가를 검사한다. 만약 방송이 되지 않았다면 이 거래를 철회시키고 이미 방송했다면 이 거래가 완료될 때까지 블록된다. 이는 기록연산으로 인하여 잠재적인 교착 상태를 발생시킨다. 거래의 종료는 완료 연산을 모든 사이트에 방송함으로써 종료된다.

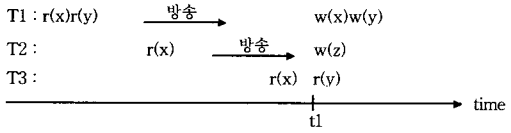
2.2.2 완료되어야 할 거래가 철회되는 문제

DWB[1, 3, 20, 26]은 BA와 BW에서 발생된 잠금 요청에 따른 부가적인 메시지부담, 통신부담, 거래지연 그리고 사이트에서 잠금 동기화로 인한 잠금 소유시간에 따른 동시성 저하의 문제를 해결하기 위해서 제안되었다. 이 기법의 기본적인 아이디어는 기록연산을 거래 끝까지 지연시켜서 모든 갱신 메시지를 방송하는데 있다. 또한 충돌되는 다른 판독연산의 거래를 철회시켜서 교착상태가 발생되지 않도록 하였다. 따라서 지역적으로 거래를 수행시켜서 동시성 증가와 중복 프로토콜의 부담을 줄이기 위한 목적을 가지고 있다. 이 방법의 수행은 클라이언트가 완료를 요구할 때까지 기록연산을 하지 않고 지연시켜서 기록연산들을 하나의 메시지로 묶어서 모든 사이트에 방송하므로 메시지 부담이 감소된다. 그러나 각 사이트에 보내는 연산은 기록잠금, 즉 $W(W_1, W_2, \cdots, W_n)$만을 보내기 때문에 기록연산이 도착된 사이트에서 수행되고 있는 지역거래의 판독연산과 충돌이 발생하면 우선적으로 그 사이트의 지역거래를 철회시켜 버린다. 이 철회된 거래가 사이클을 발생할 수 있는 거래라면 결국 철회되지만 정상적으로 수행이 완료되어야 할 거래라면 철회되지 않아야 할 거래를 철회시켜 버리므로 철회율이 증가됨에 따라서

거래율이 감소된다. 다음 구체적인 예를 통해서 살펴본다.

【예 2.1】 거래의 철회

지역 사이트 S_1은 거래 T_1이 제출되었고, 사이트 S_2에서는 거래 T_2와 T_3이 제출되었다고 하자. 각 사이트에서 수행되는 지역 동시성 제어는 엄격한 2PL 기법에 의해서 수행된다고 가정한다. 각 사이트에 데이터가 완전 중복되어 있을 때 S_1에서 T_1이 판독연산을 수행하고 기록연산들을 묶어서 방송했을 때 사이트2에서는 거래 T_2와 T_3이 판독연산을 수행한다고 하자. 이때 거래 T_1, T_2, T_3이 수행되는 과정에 대해 설명한다.



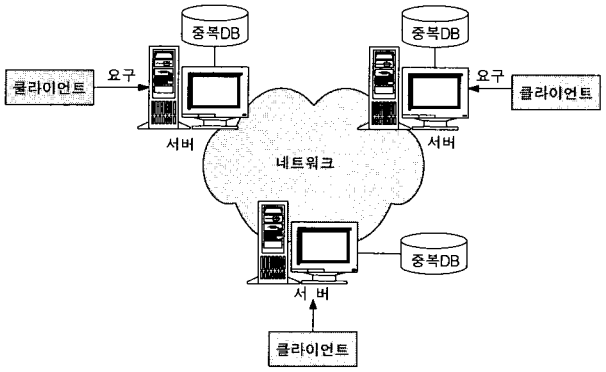
먼저 거래 T_1과 T_2 사이의 관계를 보자. 사이트 S_2에 도착되는 거래 T_1에 대한 w(x)를 수행하기 위하여 기록잠금을 요구했을 때 이미 T_2의 r(x)에 대한 판독잠금이 설정되어 있다면 판독-기록 충돌 잠금이 발생하게 된다. 일반적으로 T_1의 기록잠금은 T_2의 판독잠금이 해제할 때까지 기다렸다가 수행하여야 하는데, 이 기법은 교착상태가 발생하는 것을 미리 방지하기 위해서 T_2를 철회시키고 T_1의 기록연산을 우선적으로 수행하게 된다. 이때 이미 T_2의 기록연산이 방송되었다면 철회메시지를 다른 사이트로 방송해서 철회시키도록 한다. 따라서 T_2의 r(x)연산 다음에 어떤 종류의 연산이 나오는 것에 상관없이 철회시키므로 철회되지 않아야 할 거래가 철회되는 문제가 발생하게 된다.

거래 T_1과 T_3의 관계는 T_1이 사이트 S_1에서 이미 판독연산을 마친 거래이고 T_3은 지역거래, 즉 질의거래이다. T_3이 x에 대해서 기록잠금을 요구했을 때 T_2가 요구한 데이터에 대해서 판독잠금의 설정이 이미 끝났으면 질의거래 T_3은 정상적으로 완료되고 그 동안 T_1을 블록시킨다. 하지만 아직 판독 잠금을 하는 단계라면 이 거래를 철회시키고 T_1의 기록연산을 수행토록 한다. 따라서 DWB는 데이터의 일관성을 만족하지만 중복 프로토콜의 부담을 줄이기 위한 목적으로 데드록 발생을 미리 방지시켰으므로 거래의 철회율이 증가한다. □

지금까지 BA, BW, DWB의 기법에서 발생되는 문제점으로 잠금으로 인해서 거래가 지연되고, 과다한 메시지 부담과 완료되어야할 거래가 철회되는 예를 살펴보았다. 중복 프로토콜에서 시스템 성능을 향상시키기 위해서는 거래가 각 사이트에서 독립적으로 수행되어야 하고, 사이트들 사이의 메시지 부담과 데드록 발생으로 인한 부담을 줄이는 것이다. 이 논문에서는 위의 문제를 해결하고 각 사이트에서 거래를 독립적으로 수행시킴으로써 거래의 병행성을 향상시키는 동시성제어 기법을 제안한다.

3. 중복 데이터베이스 시스템 구조

이 논문이 모델로 하고 있는 중복 데이터베이스 환경에 대하여 살펴본다. 일반적으로 중복 데이터베이스 환경의 모델에서 시스템 구조는 (그림 3.1)과 같다. 중복 데이터베이스 환경은 클라이언트와 중복 데이터베이스에 연결된 지역 서버로 구성되는데 각 지역 서버는 네트워크 망으로 연결되어 있다. 각 클라이언트는 그의 지역 데이터 베이스 서버에게 거래 처리를 요구할 수 있다. 각 서버에 연결되어 있는 데이터베이스는 데이터를 완전 중복하고 있는 지역 데이터베이스로써 클라이언트가 제출한 거래를 관리한다. 클라이언트가 각 지역 데이터 베이스 서버에게 제출한 거래는 판독과 기록 연산으로 구성되어 있다. 지역 데이터베이스의 서버는 안정적이고 신뢰할 수 있는 네트워크에 의해 연결되어 있다.



(그림 3.1) 시스템구조

(그림 3.1)에서 지역 데이터 베이스 서버는 조정자가 되기도 하고 참여자가 되기도 한다. 클라이언트가 각 지역 데이터베이스 서버로 제출한 거래는 판독연산으로 구성된 질의와 판독과 기록 연산들로 구성된 거래로 앞으로 갱신 거래(update transaction)라 한다.

각 지역 데이터베이스 시스템은 그 지역에서 제출된 거래를 관리하고, 갱신 거래일 경우 중복 데이터베이스를 갱신하기 위한 갱신 규약의 조정자로서의 역할과 참여자의 역할을 한다. 각 지역 데이터베이스 시스템은 제출된 거래의 모든 연산이 수행된 후에 갱신 규약을 수행한다. 각 거래는 작업영역을 가지고 있어서 완료할 때까지의 갱신 결과는 작업영역에 보관된다. 이 보관된 갱신결과를 다른 중복사이트에 전파하기 위하여 갱신연산들로 이루어진 갱신 전용 거래를 생성한다. 갱신 규약은 갱신 전용 거래가 모든 중복사이트에서 일관성이 위배되지 않게 수행할 수 있도록 한다. 일관성이 위배되지 않는다는 것은 직렬성을 만족해야 한다. 직렬성은 거래의 수행에 대한 정확성 기준으로 사용되며, 동시성제어 기법에 의해서 달성된다. 하나의 데이터가 여러 서버에 중복되어 있는 환경에서 데이터의 사본들

은 하나의 논리적 데이터로 표현되어야 한다. 이를 위해 1-사본 직렬성을 사용되며 기록 연산들은 중복제어 기법 중 Eager 중복 규약에 의해 데이터를 갱신한다. 1-사본 직렬성은 분산 데이터 베이스에서 논리적으로 연결되어 있고, 단일 사본 데이터베이스 상에 거래의 인터리브한 수행은 순차적인 수행과 상등관계에 있어야 한다. 갱신 전용 거래는 해당 중복 사이트에서 다른 거래들과 동시에 수행된다. 갱신 규약이 완료되어야 거래는 완전히 완료를 하게 된다.

4. 중복 데이터베이스 시스템에서 갱신그래프를 이용한 동시성제어

이 장에서는 완전 중복 데이터베이스 시스템에서 거래들의 병행성을 향상시키면서 데이터베이스의 일관성을 유지하기 위한 거래관리 기법을 제안한다. 제안한 동시성제어 기법은 CCM-RD(A Concurr-ency Control Method for Replicated Databases)라고 칭한다.

4.1 CCM-RD에서의 잠금규약

제안하는 CCM-RD에서는 각 사이트에 제출된 거래를 그의 지역 사이트에서 독립적으로 거래의 마지막 연산까지 수행한다. 거래는 거래리스트에 등록되고, 거래가 데이터 항목을 판독하는 경우에 그 데이터 항목에 대한 판독 잠금을 설정한다. 이 판독 잠금은 기존의 2PL에서 사용되는 잠금의 방법과는 달리 기록 잠금과 양립 가능하다. 즉, 이미 판독 잠금이 걸린 데이터에 대해서도 기록 잠금을 승인할 수 있다. CCM-RD의 잠금 규약에서 잠금 형태에 대한 양립성 표는 <표 4.1>과 같다.

〈표 4.1〉 CCM_RD방법의 잠금양립성 표

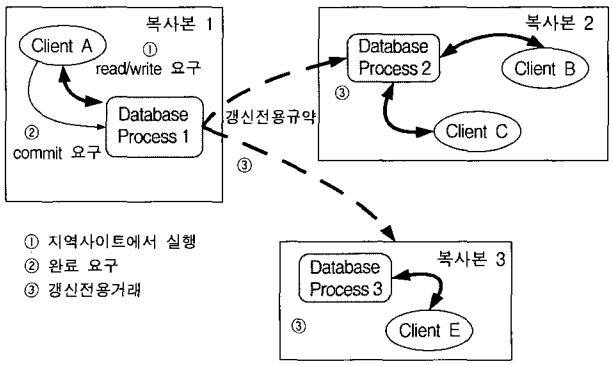
Hold \ Request	Rl	Wl
Rl	y	y
Wl	n	n

- 범 례 -
Rl : Read_lock
Wl : Write_lock,
y : 허용
n : 허용하지 않음

기존 방법에서는 판독 잠금을 유지된 상태에서 다른 거래가 기록 잠금을 요청하면 서로 배타적인 잠금 관계이나 CCM-RD에서는 기록 잠금을 허용한다. 이는 판독-기록 충돌로 인해 거래의 지연을 방지하기 위해서 인데, 기록을 허용한다는 것은 갱신전용거래가 우선적으로 수행해야 하고 기록연산을 하는 갱신전용거래가 철회되면 안되기 때문이다. 왜냐하면 갱신규약을 진입하기전의 모든 거래들을 지역거래라 했을 때 지역거래를 구성하는 연산으로 인해 갱신전용거래를 철회시키면 지금까지 수행했던 연산을 무시하고 새로 동일한 연산과정을 반복해서 행해져야 하기 때문에 전체적인 시스템 성능을 저하시키기 때문이다. 잠금의 해제는 조정자 사이트에서 갱신 규약에 의해 수행해야 하는 부 거래가 참여자 사이트에서 전부 실행이 끝날 때 잠금을 해제시킨다. 지역사이트에 제출된 트랜잭션을 전체 거래라 하고, 모든 사이트로 제출된 갱신전용거래를 부 거래라 했을 때, 판독 잠금은 부 거래들의 수행이 모두 마쳤을 때 거래가 제출된 사이트에서 해제시키고, 기록 잠금은 각 사이트에서 거래가 완료되면 해제시킨다.

4.2 갱신 규약

각 사이트로 제출된 갱신거래는 해당 사이트에서 모든 연산을 수행하고 갱신 전용거래를 생성한 후 갱신규약에 의하여 갱신 전용거래를 모든 사이트로 전파함으로서 갱신이 이루어지는데, 이 과정을 그림으로 나타내면 (그림 4.1)과 같다. (그림 4.1)은 사이트 3개로 구성된 완전 중복 데이터베이스 시스템이다. 클라이언트 A가 복사본1을 가지고 있는 사이트로 거래를 제출하였다고 하면, 지역적으로 판독연산은 판독 잠금을 얻어서 수행하고, 기록연산은 독립적인 공간에 가상기록을 수행한다. 클라이언트가 완료를 요구한다면 다른 모든 복사본 사이트로 갱신 전용 규약에 의해서 갱신요구 메시지인 UR(Tr-id, RS, WS, TS, After)을 전파하게 되고, 각 사이트에서는 직렬성 검사를 하여 직렬성 검사를 통과한 거래만 지역 스케쥴러에게 제출해서 처리된다. 여기에서 복사본1을 갖고 있는 사이트는 조정자가 되고 복사본1을 포함한 나머지 사이트들은 참여자가 되어서 갱신 규약을 수행한다.



(그림 4.1) 갱신전용거래 수행과정

각 조정자 사이트에서는 데이터를 판독할 때 판독한 데이터 항목들의 집합 *RS*와 작업영역의 갱신데이터 항목들의 집합 *WS*를 구한다. 이때 기록연산은 독자적인 거래의 작업영역에다 가상기록(Virtual Write)하고, 거래의 마지막 연산을 수행한 후에 작업영역에 있는 갱신된 자료를 반영하는 갱신전용 거래를 수행하기 위한 갱신규약을 시작한다. 갱신 규약은 중복 데이터베이스의 일관성을 파괴하지 않고 갱신전용거래의 결과를 모든 사이트에 동일하게 반영하기 위하여 필요하다. 즉 갱신 규약은 조정자 사이트와 참여자 사이트들 사이에서 서로의 갱신 메시지를 전송하여 전역 직렬성 유지를 보장하기 위한 규약이다.

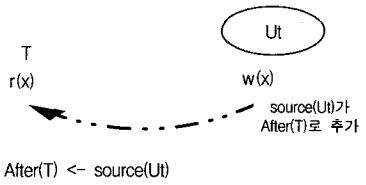
이 규약은 각 지역 사이트에서 원격 통신 없이 거래가 독립적으로 마지막 연산까지 수행하게 함으로서 병행성과 처리율을 향상시킬 수 있다. 만약 갱신 규약 결과의 결정이 완료이면 그 거래를 완료시키고 철회이면 거래를 철회시킨다. 갱신전용거래가 데이터 항목을 갱신중에 현재 수행중인 거래가 설정한 판독 잠금이 있다면 그 수행중인 거래의 *After*에 갱신전용거래의 소스번호를 추가시키고, 지역거래가 판독하고자 하는 데이터 항목을 갱신전용거래가 갱신하였다면 *Source*(*y*)를 *Before*에 추가시킨다.

After(T_i)는 갱신규약에 진입하기 전, 즉 지역거래 T_i가 판독한 데이터항목을 갱신전용거래 Ut들에 의해서 갱신되는 모든 소스번호의 집합이고, *Before*(T_i)는 완료되지 않는 갱신전용거래가 기록한 데이터를 지역거래 T_i가 판독하는 데이터항목이 가지는 소스번호의 집합이다. 그리고 *Source*(*x*)는 완료되지 않은 거래가 데이터 항목 x를 갱신한 갱신전용거래의 소스번호의 집합을 *Source*(*x*)라 하고, *Source*(*Ut*)는 갱신그래프 UG에서 갱신전용거래 Ut를 선행하는 노드들이 가지는 소스 번호의 집합을 나타낸다. UG 그래프는 임의 이력 H에 대한 방향성 에지를 가지며, UG 그래프 UG(H)에 있는 $T_i \rightarrow T_j (i \neq j)$는 T_i의 연산이 충돌 관계에 있는 Ti의 연산에 선행한는 직렬화 그래프를 의미한다.

*After*에 소스번호를 추가시키는 이유는 갱신전용거래가 UG로부터 제거되더라도 수행중인 거래와 UG내에 있는 갱신전용거래와의 관계를 기억하기 위하여 필요하다. 또한 판독-기록 충돌일 때 기록연산을 허용한다는 의미는 지역 거래가 판독한 데이터를 갱신전용거래에 의해서 갱신되는 것에 의해 일관성이 위배될 수 있는 경우가 발생할 수 있다. 그러나 지역거래에 의해서 판독한 데이터 항목을 갱신전용거래가 기록할 때 지역거래 *After*에 자신의 소스번호를 추가시키고 데이터베이스에 기록하므로, 지역거래가 갱신규약에 의해 수행되었을 때 UG 노드의 소스정보가 *After*의 소스정보를 포함하고 있으면 사이클이 발생되었다는 것을 알 수 있다. 그러므로 직렬성은 파괴되지 않으며 모든 사이트에서 동일한 데이터 값으로 갱신하게 되어 일관성을 유지할 수 있다. 소스의 생성에 대해서는 참여자 알고리즘 절에서 자세히 기술한다. *After*가 소스 정보를 유지하는 【예는 4.1】과 같다.

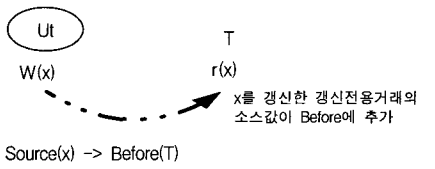
【예 4.1】 어떻게 *After*가 소스정보를 유지하는가 ?

각 거래는 *After*값과 *Before*값을 유지하는데 *After*값은 갱신규약실행전의 거래가 읽은 데이터 항목을 갱신하는 갱신전용거래의 소스정보를 말하고, *Before*값은 지역거래가 판독하고자 하는 값을 이미 갱신전용거래에 의해서 기록연산을 수행했을 때 그 갱신전용거래의 소스정보를 말한다. 따라서 각 갱신된 데이터 항목은 자신을 갱신한 갱신전용거래의 소스 값을 유지한다. 소스 값이 *After*에 추가되는 경우는 갱신규약전의 지역거래 T가 읽은 데이터 x를 갱신전용거래 Ut가 기록할 때 Ut의 소스 값을 T의 *After*로 추가시키는 경우로서 (그림 4.3)과 같다. *Before*에 소스 값을 추가시키는 경우는 아직 완료되지 않은 갱신전용거래가 갱신한 값을 지역거래가 판독하는 경우이다. 이는 판독연산보다 기록연산이 선행해서 수행되었기 때문에 이의 선행관계를 유지하기 위해서 *Before*에 갱신전용거래 소스 값을 추가시킨다. 이는 각 사이트에서 직렬성을 유지하기 위해서 유지되는 UG내의 갱신전용거래와의 관계를 기억하기 위해서 필요하며 사이클을 검색하지 않고도 사이클이 발생되었다는 것을 인지하기 위해서 필요하다.



(그림 4.2) *After*에 *Source* 번호 추가

(그림 4.2)은 지역 거래 T가 판독한 데이터 항목 x를 갱신전용거래 Ut가 x를 갱신하고자 할 때 판독-기록 충돌이 발생하지만 이 논문에서는 이의 충돌관계를 허락함으로서 이들 사이의 관계를 유지하기 위해서 T의 *After*에 Ut의 소스정보를 추가시키는 것을 나타낸다.



(그림 4.3) *Before*에 *Source* 번호 추가

(그림 4.3)은 갱신전용거래 Ut가 갱신한 x를 지역거래 T가 판독하고자 한다면 x가 가지는 소스 값을 T의 *Before*에 추가시키는 것을 보여준다. □

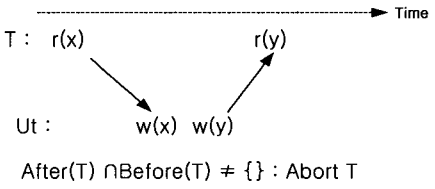
위에서 기술한 갱신전용거래를 각 지역 사이트에서 수행되는데, 클라이언트가 지역사이트로 제출한 거래의 종류는 두 가지가 있다. 하나는 판독연산들로만 구성된 질의를 들 수 있고, 두 번째는 기록연산을 하나이상 포함된 갱신 거래이다. 질의는 거래가 제출된 사이트에서 수행하고 기록연산이 없으므로 다른 복사본 사이트로 전파하지 않고 지역적으로 완료시킨다. 이때 아직 갱신 규약에 진입되지 않은 지역거래인 판독연산과 갱신 규약에 진입된 갱신전용거래사이에 직렬성이 위배되는 문제가 발생하게 되는데 이에 대한 예는 【예 4.2】와 같다.

【예 4.2】 지역거래와 갱신전용거래 사이에 직렬성 위배

한 사이트에서 지역거래 T와 갱신전용거래 Ut가 다음과 같이 수행될 때 직렬화 그래프 SG에 사이클이 발생하게 된

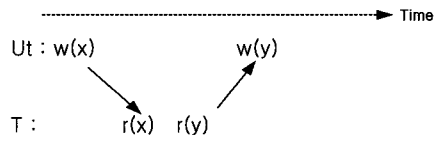
다. CCM-RD에서는 수행중인 지역거래 T 와 갱신전용거래 Ut 사이에 직렬성이 위배되는 문제를 해결하여야 하며, Ut가 철회되지 못하도록 하여야 한다. 왜냐하면 Ut는 이미 갱신규약의 수행결과 각 사이트의 데이터베이스에 반영되어야 하므로, T를 철회하여야 한다. T는 질의 거래이거나 갱신규약을 수행하기 전의 갱신거래이다.

① T → Ut → T의 사이클이 발생



After(T) ∩Before(T) ≠ {} : Abort T

② Ut → T → Ut의 사이클이 발생



지역 동시성제어 기법으로서 2PL 기법을 사용하는 기존 방법에서 이러한 사이클이 발생되면 ①의 경우는 T를 철회시키고, ②의 경우는 Ut를 철회시킨다. 그러나 CCM-RD에서는 잠금으로 인한 거래의 지연을 방지하기 위하여 기존 2PL 기법을 적용하지 않는다. 그러므로, 이와 같은 사이클이 발생되지 않도록 T를 스케쥴하여야 한다.

CCM-RD에서는 갱신전용 거래 Ut가 갱신규약 전 단계에 있는 거래 T에서 읽은 데이터 항목을 갱신하려고 한다면 *After(T)*에 *Source(Ut)*를 추가한다. *Source(Ut)*는 갱신전용거래 Ut의 소스정보이다. 이는 두 거래사이에 T->Ut의 관계가 있음을 표현한다. 이 후에 T가 *Source(Ut)*가 *After(T)*에 포함된 데이터 항목 *y*를 읽으려고 *Source(y)*에 Ut의 소스를 추가하고 *After(T)*와 *Before(T)*에 공통이 되는 소스가 있기 때문에 T를 철회함으로서 ①의 경우와 같은 사이클을 방지할 수 있다. 이는 그래프를 검사하지 않고도 ①의 경우와 같은 종류의 사이클이 발생하는 지를 검사할 수 있다. CCM-RD에서는 이미 설정된 판독 잠금에 대하여 기록 잠금이 허용되나 기록 잠금이 설정된 상태라면 다른 잠금은 허용되지 않으며, 기록 잠금의 해제는 갱신전용 거래의 완료 후에 이루어지므로 ②와 같은 사이클발생을 방지할 수 있다. 즉 Ut가 갱신하려고 하는 데이터 항목 x를 기록 잠금을 얻은 후에 갱신하기 때문에 갱신중인 갱신전용거래의 결과를 읽을 수 없다. 그러므로 ②의 경우는 발생되지 않는다. □

지역거래와 갱신전용거래를 관리하는 알고리즘은 (알고리즘 4.1)과 같다.

```
/* 지역거래와 갱신거래 사이의 수행 알고리즘 */
/* T : 갱신규약 수행전의 거래 또는 질의 거래
   Ut : 갱신전용거래  Source( x ) : x를 갱신한 갱신전용거래 소스 */
When a read-only transaction T or Update Transaction Ut request
a lock for a data item x
case requested-lock-type of
read_lock :
  IF(there is no conflict according to given lock compatibility)
    THEN
      Before(T) ← Source(x)
       IF (Before(T) ∩ After(T) == ∅)
       THEN  set a read_lock for x
          insert T into lock table to represent that T is reading x
       ELSE T is aborted
    ELSE the T is wait
write_lock :
  IF(there is no conflict according to given lock compatibility)
  THEN
    set a write_lock for x
    insert Ut into lock table to represent that Ut is writing x
    FOR any T' in the list of transactions having a read lock
     IF( x in RS(T'))
     THEN  add Source(Ut) to After(T')
     ENDIF
    ELSE the Ut is wait
```

(알고리즘 4.1) 판독전용거래와 갱신전용거래의 수행 알고리즘

다음은 제안하는 중복 데이터베이스 거래관리 알고리즘에서 사용하는 갱신 규약에서 조정자 알고리즘과 참여자 알고리즘을 기술한다.

4.2.1 갱신 규약의 조정자 알고리즘

제안하는 CCM-RD에서는 거래가 발생된 사이트는 조정자가 되고 복사본을 가지고 있는 모든 지역 사이트를 참여자라 한다. 조정자 사이트는 제출된 거래의 동시성을 제어하고 모든 사이트에서 일관성 있게 데이터 항목이 갱신 될 수 있도록 크게 두 가지 역활을 수행한다. 하나는 갱신 전용 거래를 모든 사이트에서 수행하기 위해 모든 참여자에게 갱신요구 메시지인 UR(Update Request)를 보내는 것이고, 또 다른 하나는 모든 참여자 사이트에서 오는 응답(YES 또는 NO)을 모아 완료와 철회를 결정하는 것이다. 조정자 사이트에 제출된 거래는 그의 지역사이트에서 판독연산을 하고 가상 기록을 수행토록 하는데, 이 가상 기록은 지역적으로 독자적인 공간에 기록하는 것을 의미한다. 이 가상 기록의 값은 아직 데이터베이스에 반영되기 전이므로 다른 거래들은 판독할 수 없다.

판독연산은 지역적으로 수행하고 기록 연산은 전역적으로 수행한다는 것은 다른 사이트에서도 동시에 서로 다른 거래들이 동일한 데이터 항목에 대해 판독, 기록 연산을 수행할 수 있기 때문에 판독연산과 기록연산 사이에 충돌이 발생하여 사이트들 사이에 데이터의 불 일치성을 가져올 수 있다. 따라서 제안하는 방법은 직렬성을 보장하기 위해 모든 사이트에 갱신그래프인 UG를 유지하도록 한다.

갱신요구를 한 갱신전용거래의 조정자는 해당거래의 모든 참여자 사이트로 이 거래에 대한 갱신요구를 제출하고, 각 참여자 사이트로부터 갱신요구에 응할 수 있는지(*YES*)

없는지(*NO*)의 응답을 받는다. 모든 참여자 사이트로부터 *YES*를 응답 받으면 갱신가능상태(Update) 메시지를 모든 참여자 사이트로 보낸다. 그리고, 갱신완료 메시지를 모든 참여자 사이트로부터 기다린다. 조정자는 모든 참여자로부터 갱신완료 메시지를 받은 후 해당 거래를 완료시킨다. *NO*이면 그 거래를 철회시키라는 메시지를 모든 참여자에게 보낸다. 이에 대한 알고리즘은 (알고리즘 4.2)와 같다.

```
/* 갱신 규약에 따른 조정자 알고리즘 */
/* Ut : 거래 T에 대한 write 연산으로 구성된 갱신전용거래 */
/* 갱신전용거래 message = (Tr_id, RS(T), WS(T), TS(T), After) */
/* 1. 지역적으로 수행된 거래를 모든 참여자사이트로 전파하는
      단계 */
   Propagate update message for Ut to the all site
/* 2. 참여자들로부터 투표결과도착 */
   When receive voting results from all participator sites
        IF (all result of voting == 'YES')
        THEN send the 'Update' message to all participator sites
        ELSE send the 'Abort' message to all participator sites
/* 3. 모든 참여자로부터 갱신이 완료되었다는 메시지 도착 */
   When receive update complete message from all participator
   sites
        send 'commit' message to all participator sites
```

(알고리즘 4.2) 갱신 규약에 따른 조정자 알고리즘

4.2.2 갱신 규약의 참여자 알고리즘

조정자 사이트에서 갱신 요구 메시지를 받아 처리하는 복사본을 가지고 있는 사이트를 참여자 사이트라 한다. 참여자 사이트는 먼저 조정자 사이트에서 보내는 갱신전용거래가 직렬성을 파괴하는지를 검사하여 조정자에게 그 결과를 투표하는 일을 수행한다. 직렬성 검사는 각 거래의 RS와 WS를 검사하여 충돌이 발생하는가에 따라서 UG내에 에지를 추가시킨다. UG에 대한 정의는 (정의 4.12)에 기술된바 있다. UG는 지역 사이트에서 마지막 연산까지 수행한 거래의 갱신연산들로만 구성된 갱신전용거래를 나타내는 노드로 구성된 그래프로서 직렬가능 한 순서를 보장하는 그래프이다. UG의 노드는 거래의 번호와 상태를 나타내는 필드를 가진다. UG의 상태가 가질 수 있는 값은 '*Vote*'와 '*Update*', '*Submit*', '*Commit*'으로 표기한다. Vote는 UG그래프 검사를 마치고 조정자에게 *YES/NO*를 투표한 거래를 나타내고, *Update*는 조정자가 완료를 결정할 경우 모든 참여자 사이트에서 갱신연산을 지역 스케줄러에게 제출하여 기록하라는 의미이다. *Submit*는 *Update*와 *Vote*로 마크된 입력에지가 하나도 없는 노드로서 갱신전용거래를 각 지역 스케줄러로 제출하였다는 의미이다. 또한 *Commit*는 제출된 거래가 완료되었다는 것을 의미한다. UG 노드 상태 값이 *Update*와 *Vote*로 마크된 노드를 입력에지로 가지지 않는다는 것은 충돌되는 거래가 존재하지 않는다는 것이고, 이 노드를 소스노드라 한다. 즉 소스노드는 입력에지의 노드 상태 값이 '*Commit*' 으로만 구성된 노드를 소스노드라 한다.

각 사이트에서 UG의 접근은 동기화가 되며, UG에 생성되는 노드는 다음의 조건이 만족하게 되면 UG 내의 임의의 노드 Tug와 갱신규약을 요구한 거래 T의 두 노드 사이에 에지가 생성된다. *WS*(Tug)는 갱신전용거래 Tug가 갱신할 자료항목들의 집합이고, *RS*(T)는 거래 T가 판독한 자료항목들의 집합이다. Tug와 T사이에 에지가 생성될 조건 및 생성된 에지는 다음과 같다.

(조건 1) $WS(Tug) \cap RS(T) \neq \emptyset$: 생성된 에지 T -> Tug (Tug 투표한 거래)

(조건 2) $RS(Tug) \cap WS(T) \neq \emptyset$: 생성된 에지 Tug -> T (Tug 투표한 거래)

(조건 3) $((WS(Tug) \cap WS(T) \neq \emptyset)$ and $((WS(Tug) \cap RS(T) = \emptyset)$ and $(RS(Tug) \cap WS(T) = \emptyset)))$

(조건 4) 조건 3 and (ts(Tug) < ts(T)) : 생성된 에지 Tug -> T

(조건 5) 조건 3 and (ts(Tug) > ts(T)) and (Tug는 제출되지 않음) : 생성된 에지 T -> Tug

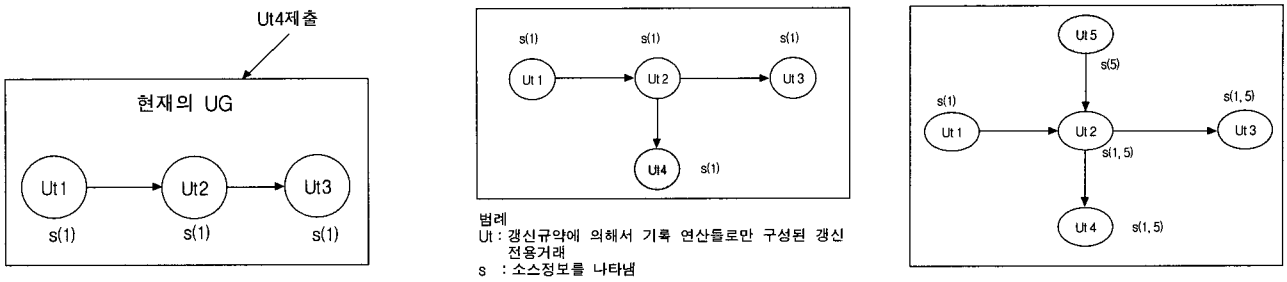
(조건 6) 조건 3 and (ts(Tug) > ts(T)) and (Tug는 제출됨) : T 는 철회됨

(조건 1)에서 거래 T는 갱신전용거래 Tug가 데이터항목을 갱신 전에 그 데이터 항목들을 읽었기 때문에 T가 직렬화 순서에서 Tug를 선행하여야 한다는 의미이다. (조건 2) 역시 거래 Tug는 T가 데이터항목을 갱신하기 전에 그 데이터 항목에 대한 판독연산을 수행하였다는 것을 의미한다. 이는 각 사이트에서 독립적으로 수행하고 나서 갱신전용거래를 수행하기 때문이다. UG에서 $T_1 \rightarrow T_2$의 관계는 직렬화 순서에서 T_1이 T_2를 선행한다는 것을 의미하고 T_1이 소스 노드의 거래이면 지역 데이터베이스에 제출되어 수행될 수 있다. 이 소스노드는 소스번호를 가지고 있다. 소스번호는 UG에 갱신전용거래를 추가할 때 소스 번호가 부여되고, 같은 경로를 가진 노드는 동일한 소스번호를 가진다. 소스번호의 부여는 UG에 추가하고자 하는 갱신전용거래의 노드 에지가 입력 에지일때 바로 전 노드의 소스번호를 자신의 소스번호로 생성하고, 출력 에지이면 자신의 거래 번호로 새로운 소스번호를 생성하여 인접된 같은 소스번호를 가진 경로의 단말노드까지 소스번호를 전파한다. 만약 출력에지가 소스노드에 연결되면 소스번호를 생성하지 않고 소스노드의 번호를 자신의 소스번호로 한다. 그에 대한 예는 【예 4.3】 과 같고, 소스번호의 전파이유는 다음 【예 4.4】로 설명한다.

【예 4.3】 소스번호 생성

사이트 S에서 UG가 다음 (그림 4.5)의 (a)와 같을 때 거래 T_4에 대한 Ut_4가 각 사이트에 전파되었다고 하자. 이 때 Ut_1, Ut_2, Ut_3은 UG의 노드로써 충돌관계에 의해서 같은 소스번호경로를 가지고 있다고 하자.

T_1 : r(x)w(a) $RS = \{x\}$ $WS = \{a\}$
T_2 : r(z)r(y)w(x) $RS = \{y, z\}$ $WS = \{x\}$
T_3 : r(b)w(y) $RS = \{b\}$ $WS = \{y\}$
T_4 : r(c)w(z) $RS = \{c\}$ $WS = \{z\}$



(a) Ut_4를 추가 전 상태 (b) Ut_4를 추가 후 소스번호 (C) Ut_5를 추가 후 소스번호

(그림 4.4) T_4와 T_5의 소스번호를 부여하기전과 후의 상태

(그림 4.4) (a)는 UG에 Ut_4를 추가하기 전이고 (b)는 Ut_4를 각 사이트에 전파했을 때 데이터 z는 T2 연산 z와 충돌관계이므로 $Ut_2 \rightarrow Ut_4$의 에지가 생성이 되는데, 생성된 에지는 입력 에지를 가지므로 인접된 노드의 소스번호를 자신의 소스번호에 추가한다. ①의 (b)상태에서 T5가 지역 트랜잭션에 제출되었다고 할때 T5 = r(x)w(b)하자. T5가 갱신규약에 의해서 $Ut_4 \rightarrow Ut_5$의 에지가 만들어진다면 UG의 노드에 (C)와 같은 소스번호를 가진다. (C)에서 거래의 충돌로 인해 Ut_5에서 Ut_2로 가는 출력에지가 만들어진다면 Ut_5는 소스노드이므로 소스번호를 자신의 거래 번호로 생성한 다음 같은 경로를 가지는 노드에 소스번호를 전파하면 Ut_2와 Ut_4, Ut_3은 Ut_5의 소스번호가 추가되어 s(1, 5)라는 소스번호를 가진다. □

【예 4.4】 소스번호의 전파이유

UG는 $Ut_1 \rightarrow Ut_2$라고 가정하고 한 지역 사이트에서 거래 T_i가 수행 중에 Ut_1이 T_i가 판독한 데이터 항목을 갱신한다고 하자. 그리고 T_i의 수행이 끝난 후에 T_i의 갱신전용거래 Ut_i가 각 사이트에서 수행된다고 하자. 이 때 $T_i \rightarrow Ut_1 \rightarrow Ut_2 \rightarrow Ut_i$의 수행순서가 존재하게 된다. 이는 Ut_i는 T_i이므로 직렬성이 파괴된다. 그러나, Ut_i의 갱신규약 수행 때까지 Ut_1이 UG에 존재하면 UG에서 사이클이 생성되어 Ut_i의 투표 결과는 *NO*가 되어 T_i는 철회가 된다. 그러므로, 직렬성을 보장하게 된다. 즉, Ut_1의 소스번호는 Ut_2의 소스번호가 된다. T_i의 *After*에는 소스번호 1이 있고, Ut_i를 UG에 첨가할 때 인접노드 Ut_2의 소스번호가 1이므로 T_i 에서 Ut_i까지 충돌경로가 존재한다는 것을 의미한다. 갱신규약을 수행 중에 *After*에 UG에서 Ut_i의 인접노드의 소스번호가 존재하면 *NO*를 투표하고 T_i를 철회함으로서 직렬성을 보장할 수 있다. 만약 Ut_1이 UG에서 제거된다면 갱신규약 수행 중에 사이클을 검출할 수 없으므로 직렬가능하지 않은 수행의 결과를 낳는다. 이러한 문제를 해결하기 위하여 UG 그래프의 마지막 노드의 갱신 전용 거래가 지역 사이트에서 갱신을 하고 완료했다 하더라도 삭제시키지 않고, 그 노드의 마크 필드에는 '*Commit*'으로 마크시킨다. 따라서 이러한 비 직렬성의 문제를 해결할 수 있다. □

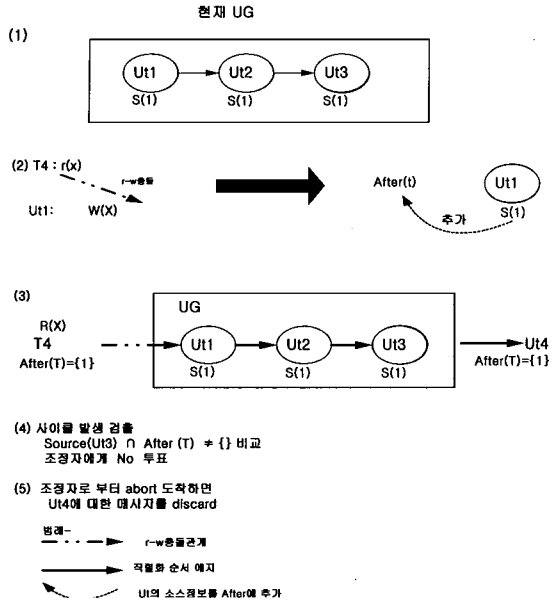
위에서 기술된 것처럼 소스번호의 전파는 갱신전용거래와 지역거래들 사이에 충돌관계를 검사하여 직렬성을 파괴하지 않고 수행하기 위해서 필요하다. 이때 직렬성 검사는 UG의 거래가 충돌연산이 발생하면 UG 그래프에 에지를 추가하여 소스를 삽입하고 소스번호가 *After*에 포함되어 있으면 조정자 사이트에 *NO*를 투표한다. 그러나 *After*에 포함되어 있지 않으면 UG에 사이클이 존재하는가를 검사한다. 사이클이 존재하지 않으면 *Yes*를 투표하여 조정자 사이트로부터 응답을 기다린다. 이때 투표한 거래를 나타내는 노드의 상태를 '*Vote*'라 마크한다. 참여자 사이트는 조정자 사이트로부터 '*Update*' 메시지를 받으면 UG에서 해당 거래를 '*Update*'로 마크하고 이 노드의 입력에지 노드가 *Commit* 또는 *submit*인가를 확인하여 *Commit* 또는 *submit*가 아니면 그 거래는 지역 스케줄러에게 제출되어 실행시킨다. 따라서 UG는 갱신규약을 요구한 거래가 각 사이트에서 갱신할 수 있음을 검사하기 위해여 사용된다. 다음은 *After*와 *Source* 값을 이용한 사이클 검출 예는 【예 4.5】와 같다.

【예 4.5】 *After*와 *Source* 값을 이용한 사이클 검출

현재 갱신전용 거래 Ut_1, Ut_2, Ut_3이 다음과 같이 (그림 5.7)의 (1)과 같을 때 지역 거래 T_4가 제출되었다고 하자.

Ut_1 : w(x) RS = {y} WS = {x} Ut_2 : w(y) RS = {z} WS = {y}
Ut_3 : w(z) RS = {a} WS = {z} T_4 : r(x)w(a) RS = {x} WS = {a}

(그림 4.5)의 (1)는 현재 UG가 가지는 노드이다. UG가 Ut_1을 지역 스케줄러에게 제출하여 기록잠금을 요구했을 때 이미 x에 대해 판독연산을 가지고 있다면 기록연산을 수행하되 Ut_1의 소스 번호를 T의 *After*에 추가시키고 기록한다. 이러한 관계는 판독-기록충돌이 발생했다는 것을 알 수 있고 지역거래와 갱신전용거래사이의 충돌관계를 나타내고 있다. 거래 T_4가 갱신전용거래로서 Ut_4로 각 지역 사이트에 제출되면 UG에 있는 거래들이 가지는 RS과 WS을 비교해서 에지를 형성하면 $Ut_3 \rightarrow Ut_4$로 가는 에지가 만들어진다. 이때 *Source*(Ut_3)의 값이 *After*(T)의 값을 포함하고 있으므로 (3)과 같은 사이클이 발생되었다는 것을 알 수 있다. 따라서 사이클를 검사하지 않고도 사이클 발생 유무를 확인할 수 있다. □



(그림 4.5) *After*와 *Source*를 이용한 거래 수행단계

UG에서 갱신전용거래가 지역 스케쥴러에 의해서 수행을 마친 후 UG그래프에서 갱신전용거래 노드를 제거해 주어야 한다. 노드의 제거는 지역거래와의 상태 값을 유지하면서 지역거래의 판독연산과 직렬성 유지에 관계없는 노드만을 제거해야 한다. 이를 위해 UG에서 지역 스케쥴러로 제출하여 기록한 시점에 대한 타임스탬프 $TS_{write}(Ut)$ 값을 유지하여 하나의 갱신전용거래가 완료하는 시점에서 UG의 노드의 상태 값이 *Commit*이고 $TS_{write}(Ut) < TS(T)$인 갱신전용거래만을 UG에서 삭제시킨다. 알고리즘은 (알고리즘 4.3)과 같다.

```
/*  Node_Deletion_Procedure  */
/* T : 갱신규약진입전 지역거래,            Ut : 갱신전용거래
   TS(T) : 각 사이트에서 현재 활동 거래의 타임스탬프
   TSwrite(Ut) : Ut가 소스노드일 때 지역 스케쥴러에게 제출한 타
                 임스탬프
   Utnode : commit된 갱신전용거래 노드들          */
  When Utnode from UG is deleted
  IF(TS(T) > TSwrite(Ut))
  THEN delete  Utnode in UG
  ENDIF
```

(알고리즘 4.3) 노드삭제 알고리즘

그러나 거래가 완료된 후 UG에 있는 노드 삭제시 동일한 소스번호를 가지는 노드는 계속 삭제된 소스번호를 유지하고 있기 때문에 시간이 지남에 따라 필요 없는 소스번호를 유지할 수 있다. 이러한 문제는 한 노드에 입력 에지가 여러개 존재하는 경우에 발생하게 된다. 따라서 적당한 시점에서 소스번호를 갱신시켜 주어야 한다. 소스번호의 갱신은 주기적으로 갱신한다고 가정한다.

제안하는 방법은 2PL에서 발생되는 잠금으로 인한 거래 지연이 발생되지 않으므로 동시성을 향상시킬 수 있다. 이를 위한 참여자 알고리즘은 (알고리즘 4.4)와 같고 갱신 규약에 의해 전파된 갱신 전용거래에 대하여 직렬가능한 순서가 있는지를 검사하는 알고리즘은 (알고리즘 4.5)와 같다.

```
/* 갱신규약에 따른 참여자 알고리즘 */
/*  갱신전용거래 =(Tr_id, RS(T), WS(T), TS(T), After)
    Ut : 거래 T에 대한 갱신 전용 거래
    source node : 노드의 상태값이 commit or submit를 가지지 않
                  는 경우*/
When receive a 'write_only transaction' from coordinator site
 /*  1. 직렬성 검사 알고리즘을 이용하여 갱신전용거래 Ut의
        직렬성검사 */
call check_serializability_procedure
IF (the execution sequence of serializability  exists)
THEN send the 'YES' massage to the coordinator and mark
the state as 'Vote'
ELSE send the 'NO' massage to the coordinator and mark the
state as 'Vote'
/*  2. 투표한 결과 메시지 도착(Update or Abort)  */
When receive a 'Update' or 'Abort' message from coordinator
site
     IF (Update)
     THEN corresponding transaction_node in UG marks 'Update'
          wait until Ut is the source node
          submit Ut to the local scheduler
          mark the state in source node as 'Submit'.
     ELSE abort T
/* 3. 완료  */
When receive commit message from coordinator site
commit for the transaction Ut
Node_Deletion Procedure
```

(알고리즘 4.4) 갱신규약에 따른 참여자 알고리즘

```
/* 직렬성검사 알고리즘 */
 check_serializability_procedure
 Tug : UG에서의 임의의 노드
 Ut : 거래 T에 대한 갱신 규약에 의해 모든 사이트에 전달된 갱
      신 전용 거래
  T  : 조정자 사이트로 제출된 지역 거래   */
/*  Ut를 UG에 삽입하는 작업  */
   For any Tug in UG
   IF (WS(Tug) ∩ RS(T) ≠ ∅)
   THEN IF(state of Tug has been Submit or Commit)
          THEN add an edge "Ut → Tug" to UG
          ELSE return No
    IF (RS(Tug) ∩ WS(T) ≠ ∅) THEN add an edge "Tug
       -> Ut" to UG
    IF((WS(Tug) ∩ WS(T) ≠ ∅) && (WS(Tug) ∩ RS(T) = ∅)
                             && (RS(Tug) ∩ WS(T) = ∅))
    THEN IF (ts(Tug) < ts(T)
           THEN add an edge "Tug -> Ut" to UG
         ELSE IF (state of Tug has been Submit or Commit)
                THEN return No
              ELSE add an edge "T → Tug"
/*  사이클 검사하는 단계  */
     IF (After of Ut ∈ source number of Tug)
     THEN return No              /* 사이클 발생  */
     ELSE IF (exist the cycle)
          THEN return No
            ELSE mark state of Ut as Vote
                 return Yes
/*  소스번호 부여하는 단계  */
     IF (Ut is in-edge)
     THEN add the source number of Tug to the source
number of Ut
     ELSE IF (Tug is source)
          THEN add the source number of Tug to the source
number of Ut
          ELSE create source number as Tr-id
               propagate source number of Ut to path of Tug
```

(알고리즘 4.5) 직렬성 검사 알고리즘

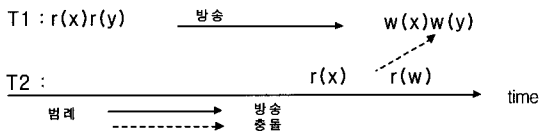
다음 【예 4.6】 은 앞의 문제 정의에서 기술된 DWB에서

발생되는 문제를 OCC-RD에서 발생되지 않음을 보이고 있다.

【예 4.6】 DWB에서 발생되는 문제 해결

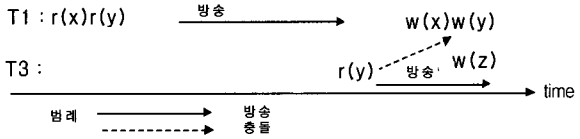
갱신거래 T_1, T_3과 질의거래 T_2가 있다고 하자. 이들 거래가 사이트 S에서 동시에 수행한다고 한다. 먼저 질의와 갱신거래사이에 충돌이 발생되는 경우와 갱신거래들 사이에 충돌이 발생되는 경우를 나누어서 살펴본다.

① 질의와 갱신거래인 경우



질의거래 T_2가 판독한 데이터 x를 갱신거래 T_1이 기록할려고 할때 DWB에서는 이 질의거래를 철회시키고 T_1을 수행하도록 하는데 OCC-RD에서는 T_2와 상관없이 T_1은 기록연산을 수행하고 완료시킨다. 단 T_1의 갱신전용거래가 Ut_1이라면 Ut_1의 *Source*(Ut_1)을 *After*(T_2)에 추가시키면 된다.

② 갱신거래와 갱신전용거래인 경우



갱신거래 T_3이 y를 판독한 후 T_1의 갱신전용거래가 y에 대해 기록잠금을 요청하면 판독-기록충돌이 발생한다. 기존의 DWB에서는 거래 T_3을 철회시키고 T_1에게 기록잠금을 승인하지만 CCM-RD에서는 거래 T_3을 철회시키지 않고 T_1이 기록 연산을 수행하도록 하고, T_3의 *After*에 T_1의 소스 값을 추가시킨다. 다음에 이 *After* 값을 이용하여 사이클을 검사한다. 따라서 DWB에서 발생되는 완료되어야 할 거래를 철회시키지 않고 정상적으로 수행시키고 기록-잠금 충돌에 의해서 잠금의 거래가 지연되는 문제를 해결하였다. □

거래들 사이에 충돌이 적을 때 사이클이 생성될 가능성이 적게 되고 거래들은 각 사이트에서 독립적으로 수행되고 완료될 가능성이 높으므로 보다 좋은 성능향상을 보일 수 있다. 거래의 수행시간과 그의 갱신전용거래의 갱신규약 수행시간을 비교할 때 갱신규약 수행시간이 상대적으로 상당히 짧다. 그러므로 병행수행의 효과를 얻을 수 있다.

5. 구현 및 평가

본 장에서는 제안하는 중복 데이터베이스의 동시성 제어 기법인 CCM-RD기법과 성능평가 대상으로는 DWB 동시성 제어 기법을 비교함으로서 성능평가 결과를 기술하고자 한다. 성능평가 기준으로는 각 트랜잭션들의 평균응답시간과 철회율을 사용하였고, 거래의 도착 간격에 따른 거래의 응답시간을 사용하였다. 거래들의 평균 수행시간은 트랜잭션들의 평균 응답시간이기 때문에 중복 데이터 베이스 환경에서 성능을 비교할 수 있다.

모의 실험은 Linux 운영체제로 Pentium-Ⅲ에서 수행하였으며, 모의실험도구로는 MCC (Microelectronics and Computer Technology Coorperation)에서 개발한 프로세스 지향의 모의실험언어인 CSIM을 사용하였다. CSIM의 라이브러리는 C나 C++ 언어를 사용하는 프로그램머들이 쉽게 프로세스 지향의 모의실험 모형을 표현할 수 있도록 되어 있는 각종 함수들이다.

5.1 모의 실험 매개변수

모의 실험에서 사용된 매개변수들은 거래 파라메터와 시스템 파라메터로 분류해 볼 수 있다. 거래 파라메터는 중복 데이터베이스 환경의 모의실험에서 사용하는 각 매개변수의 값들은 <표 5.1>에 나타내었다.

각 중복 데이터베이스는 40개의 자료 항목을 보유한다고 가정한다. 이 매개변수의 값은 모의 실험을 Pentium PC에서 수행하였기 때문에 PC 용량에 적절하도록 선정하였다. 각 거래의 길이에 대한 매개변수 값은 지역 데이터베이스의 15%에 해당하는 자료 항목에 대해 읽기/쓰기연산을 수행하도록 한다. 각 지역 데이터베이스의 데이터 항목을 15% 접근한다는 것은 너무 긴 거래로서 현실적으로 적용하기는 부적절하다. 또한 접근하는 데이터 항목의 비율이 클수록 충돌이 발생할 수 있는 확률은 증가한다. 각 클라이어트가 중복 데이터베이스의 트랜잭션 관리자에게 거래를 제출하는 것은 지수분포에 의해 랜덤하게 발생시키며, 매개변수 값 중에서 시간과 관계되는 값들은 논리적인 시간단위를 나타낸다. 거래 파라메터에 대한 의미와 값은 <표 5.1>과 같다. 시스템 파라메터는 중복 데이터베이스 시스템 환경 및 특성을 규정하는 것으로 <표 5.2>와 같다.

〈표 5.1〉 거래 파라메터

파라메터	의 미	값
Tr_client_submit	하나의 클라이언트에서 제출되는 거래 수	100
Tr_arrival_interval	클라이언트에서 거래가 발생되는 평균간격(간격은 지수분포에 따라서 랜덤하게 제출)	Value
Tr_length	거래의 길이(MAX, MIN)	데이터베이스 크기의 비율(%)
ROPercent	판독전용거래 비율	Value
Write_in_UT	갱신거래에서 기록연산비율	Value

〈표 5.2〉 시스템 파라메터

파라메터	의 미	값
Number_of_server	서버 수	Value
Number_of_client	클라이언트 수	Value
Dataset_size	데이터 집합 크기	Value
DiskTime	디스크 접근시간	10ms
Lock_CpuTime	잠금을 거는데 CPU 시간	0.006 ms
Sch_CpuTime	스케쥴 CPU 시간	10ms
Msg_cpu_time_	한 개의 메시지를 처리하는 CPU 시간	0.5ms
Msg_trans_delay	한 사이트에서 메시지 전송하는데 걸리는 시간	0.7ms

5.2 응답시간에 따른 분석

모의 실험에서 사용된 중요한 성능평가 기준은 각 거래들의 평균 응답시간이다. 거래들의 응답시간은 지역 거래들이 모든 지역 데이터베이스로 제출된 시간으로부터 그 수행이 완료될 때까지의 시간이다. 어떤 거래가 수행 중에 철회되었다면 그 거래의 응답시간은 그 거래가 재 시도되어 수행이 완료될 때까지 소요된 시간이다. 거래의 응답시간을 분석함으로써 사용자에게 얼마나 빠른 결과를 제공할 수 있는지를 판단할 수 있다. 성능평가의 기준으로 거래가 데이터베이스의 거래 관리자에게 제출하는 분포는 지수분포에 따르며 랜덤하게 발생시켰다.

제안한 CCM-RD와 비교대상이 되는 DWB와 판독전용거래 비율에 따라서 응답시간을 상호 비교하여 (그림 5.1) ~ (그림 5.3)에 나타내었다. 아래의 그래프에서와 같이 CCM-RD가 DWB보다 더 빠른 응답시간을 제공함을 알 수 있다.

또 다른 응답시간에 따른 분석은 서버에 도착하는 거래의 간격 비율에 따라서 분석할 수 있다. 이는 도착 간격 비율이 짧을수록 동시에 각 사이트에서 수행되는 거래가 많이 발생하게 되고, 갱신거래의 비율이 높을수록 충돌하는 확률이 높다는 것을 의미한다. 이 모의실험에서는 판독전용 거래의 비율은 75%로 하였고 •갱신거래내의 기록연산의 비율은 50%로 하여 모의실험을 하였다. DWB와 CCM-RD의 응답시간을 비교하여 분석한다. 이 두 방법을 사이트가 3개, 6개, 9개일 때 도착간격의 값을 변경해서 비교했을 때 사이트 수가 많을수록 서버에 도착하는 거래의 간격이 짧을수록 응답시간도 커지는 것을 알 수 있다. 모의 실험을 통해서 CCM-RD가 DWB 보다 더 적은 응답시간을 가짐을 보여주며, 이들 관계를 그래프로 나타내면 (그림 5.4) ~ (그림 5.6)과 같다. 이는 CCM-RD가 DWB보다 하나의 거래를 수행하는 시간이 적게 소요됨에 따라 더 짧은 응답시간을 제공함을 알 수 있다.

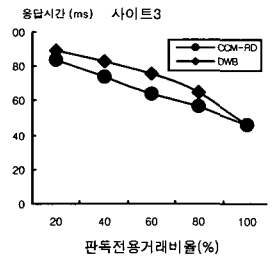
5.3 철회율에 따른 분석

트랜잭션의 성능을 평가하는 또 하나의 기준은 트랜잭션의 철회율이다. 이것은 트랜잭션의 처리량을 평가하는 기준이 된다. 철회율이 높으면 처리량이 적다는 것을 의미하므로 철회율의 발생이 적을수록 시스템 성능을 향상시킬 수 있다. 판독전용 거래 비율이 거래의 철회율에 어떠한 영향을 미치는 지는 사이트의 수에 따라서 다르며 제안한 CCM-RD와 비교대상이 되는 DWB와 판독전용거래 비율에 따라서 철회율을 상호 비교하여 (그림 5.7) ~ (그림 5.9)에 나타내었다.

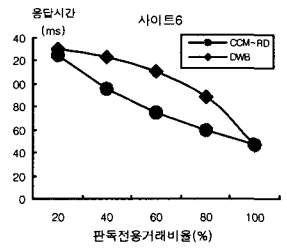
CCM-RD에서의 철회율은 판독전용거래 비율이 많을수록 그래프의 곡선이 완만하게 변화되고 있음을 알 수 있다. 판독전용거래 비율이 100% 일때는 사이트 수에 상관없이 철회율은 CCM-RD와 DWB 모두 0%이다. 이는 모든 거래가 지역 사이트에서만 수행되기 때문이다. 그리고 판독전용거래 비율이 적을수록 갱신 거래 비율이 높으므로 모든 사이트로 갱신된 데이터 값을 동일하게 반영시켜야 하므로 각 사이트로 제출되는 지역거래와의 충돌은 많아지며 이에 따른 거래의 철회율도 증가된다. 또한 판독전용거래 비율이 20%일때 사이트 6과 사이트 9에서는 거의 같은 철회율을 제공하고 있다. 따라서 판독전용거래 비율이 높을수록 CCM-RD와 DWB는 거의 동일한 철회율 값을 가진다는 것을 모의실험을 통해 알 수 있었다.

6. 결 론

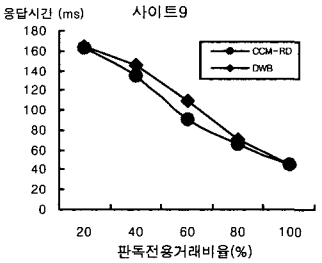
분산 데이터베이스 시스템을 기반으로 한 중복 데이터베



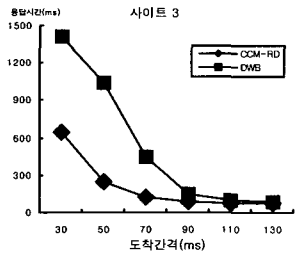
(그림 5.1) 사이트 3에서 판독전용비율에 따른 응답수



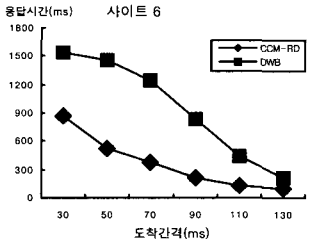
(그림 5.2) 사이트 6에서 판독전용비율에 따른 응답수



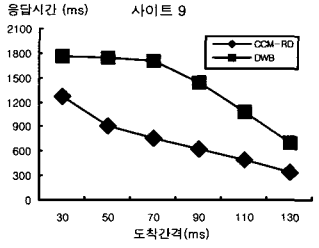
(그림 5.3) 사이트 9에서 판독전용비율에 따른 응답수



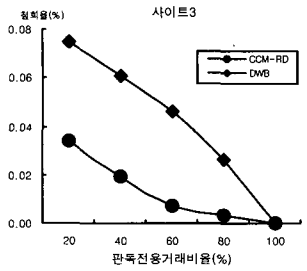
(그림 5.4) 사이트 3일 때 도착간격에 따른 응답시간



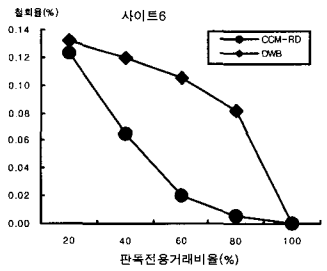
(그림 5.5) 사이트 6일 때 도착간격에 따른 응답시간



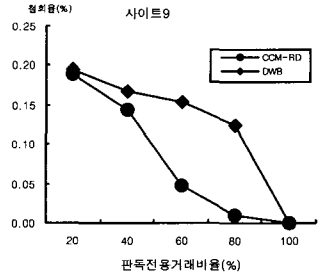
(그림 5.6) 사이트 9일 때 도착간격에 따른 응답시간



(그림 5.7) 사이트 3에서 판독전용비율에 따른 철회율



(그림 5.8) 사이트 6에서 판독전용비율에 따른 철회율



(그림 5.9) 사이트 9에서 판독전용비율에 따른 철회율

이스 시스템 환경은 가용성과 신뢰성을 증가시키기 위한 정보 처리의 중요한 분야로 등장하고 있다. 만약 중복 데이터베이스 시스템에서 수행되는 거래가 판독전용 거래이면 응답시간이 짧아지고, 동시에 처리되는 거래의 처리량도 향상시킬 수 있지만 갱신거래라면 중복되는 모든 사이트에서 일관성을 유지하기 위하여 각 사이트들 사이에 교환되는 메시지 부담과 분산 교착상태의 문제가 발생하게 된다.

이 논문은 Eager 규약에서 발생되는 모든 사이트의 동기화로 인한 메시지 부담과 분산 교착상태가 발생되지 않고 중복 데이터베이스에서 거래의 병행수행을 높이기 위한 동시성 제어 기법인 CCM-RD를 제안하였다. 기존에 연구된 내용과 제안된 CCM-RD가 가지는 특징을 요약하면 다음과 같다.

첫째, 모든 사이트에서 갱신이 가능한 Update Everywhere 방법으로 주사본 관리 기법과는 달리 사이트의 지역 자치성을 최대화시키고, 지역 사이트에서 자치적으로 동일한 데이터에 대한 갱신이 이루어지도록 하였다. 이를 위해서 CCM-RD에서는 각 사이트에 제출되는 거래가 독립적으로 그 지역 사이트에서 마지막 연산까지 수행을 마치고 그의 작업영역에 있는 갱신결과를 갱신규약에 의해서 모든 사이트에서 동일하게 수행되도록 하였다.

둘째, 이 논문에서 사용되는 잠금 규약은 지역거래(갱신규약에 진입되기 전 거래)와 갱신전용거래사이의 충돌과 같은 판독-기록 충돌이 발생될 경우, 잠금으로 인한 지연을 방지하기 위해서 판독-기록 충돌을 허용하였다.

셋째, 판독-기록 충돌을 허용하기 위해서 사용되는 *After*와 *Source*의 자료 값을 이용해서 사이클을 검출하지 않고도 사이클 발생을 인지하였다.

넷째, 전역 직렬성을 보장하기 위해서 판독-기록 충돌시 판독연산을 먼저 수행하는 거래가 선행되도록 하고, 기록-기록 충돌이 발생하면 타임스탬프 값을 이용해 타임스탬프 값이 적은 거래가 먼저 수행되도록 하였다. 이러한 거래의 순서는 각 사이트에서 동일한 순서를 보장하는 UG 그래프를 이용해서 전역적 직렬성을 보장하였다.

다섯째, 기존의 판독-기록 충돌로 인한 갱신 거래는 철회되는 문제를 *After*의 정보를 유지함으로써 미래에 일어날 사이클을 해결하였다.

중복 데이터베이스에서 갱신규약을 사용한 거래관리 알고리즘은 갱신거래들 사이에 충돌발생이 적으면 병행수행의 정도가 높아져서 좋은 성능을 보였다. 또한 읽기 연산의 수가 쓰기 연산의 수보다 훨씬 많은 응용에서는 읽기 연산의 국지성을 증가시키고 거래가 동시에 수행되는 병행수행의 효과가 있음을 확인하였다. 따라서 CCM-RD는 중복 데이터베이스 시스템 환경에서 가용성과 신뢰도를 향상시키고, 해당 사이트에서 모든 연산을 수행함으로 병행성의 정도를 높일 수 있다.

제안한 CCM-RD는 완전 중복 데이터베이스를 구축하는 모든 시스템에서 보다 효율적인 거래관리를 제공해줄 뿐만 아니라 보다 높은 시스템 성능을 보장하므로 일관성을 중요시하는 업무와 비상시 기존의 데이터베이스를 대신해서 완전 중복된 데이터베이스를 활용함에 따라 보다 높은 신뢰성을 제공할 수 있다.

참 고 문 헌

[1] D. Agrawal, G. Alonso, A. E. Abbadi, "Exploiting Atomic Broadcast in Replicated Databases," In Proceedings of EuroPar(EuroPar'97), Passau(Germany), 1997.

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

[3] B. Kemme, G. Alonso, "A Suite of Database Replication Protocols Based on Group Communication Primitive," In Proceedings of the 18th International Conference on Distributed Computing Systems (ICDCS), Amsterdam, The Netherlands, May, 1998.

[4] B. Kemme, F. Pedone, "Processing Transactions over Optimistic Atomic Broadcast Protocols," In Proceedings of the International Conference on Distributed Computing Systems, Austin Texas, June, 1999.

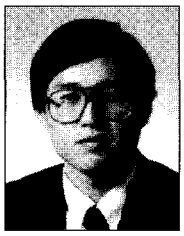
[5] A. Kumar, A. Segev, "Cost and Availability tradeoff in Replicated Concurrency Control," ACM Transactions on Database Systems, pp.102-131, March, 1993.

[6] M. Wiesmann, F. Pedone, A. Schiper, B. Kemme, and G. Alonso, "Understanding Replication in Databases and Distributed Systems," In Proceedings of 20th International Con-

ference on Distributed Computing Systems, pp.264-274, 2000.
[7] J. Gray, P. Helland, D. Shasha, "The Dangers of Replication and a Solution," In Proc. of the ACM SIGMOD, pp.568-574, 1996.
[8] Y. Breitbart, R. Komondoor, R. Rastogi, S. Seshadri, A. Silberschatz, "Update Propagation Protocols for Replication Databases," In Proc. of the ACM SIGMOD, pp.97-108, 1999.
[9] B. Kemme, F. Pedone, G. Alonso, A. Schiper, "Using Optimistic Atomic Broadcast in Transaction Processing Systems," Technical Report No.325, Department of Computer Science, ETH Zurich, Mar., 1999.
[10] B. Kemme, and G. Alonso. "A Suite of Database Replication Protocols Based on Group Communication Primitives," In proceedings of the 18th International Conference on Distributed Computing System (ICDCS), Amstergam, The Netherlands, May, 1998.
[11] M. Stonebraker, "Concurrency Control and Consistency of Multiple Copies of Data in Distributed INGRES," IEEE Transactions on Software Engineering, pp.188- 194, May, 1979.
[12] Todd Anderson, Y. Breitbart, Henry F. Korth, Avishai Wool, "Replication, Consistency, and Practicality : Are These Mutually Exclusive?," In Procs. of ACM SIGMOD International Conf. on Management of Data, Seattle, WA, Vol.27, No.2, pp484-495, 1998.
[13] Y. Breitbart and Henry F. Korth. "Replication and Consistency : Being Lazy Helps Sometimes," In Proceedings of the ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems, Tucson, Arizona, 1997.
[14] J. Holliday, D. Agrawal, A. e. Abbadi, "The Performance of Database Replication using Atomic Broadcast Group Communication," Techinical Report TRCS 99-11, The University of California at Santa Barbara, 1999.
[15] M. Wiesman, F. Pedone, A. Schiper. "A Systematic Classification of Replicated Database Protocols Based on Atomic Broadcast," In Proceedings of the 3rd Europeean Research Seminar of Advances in Distributed Systems, Madeira, April, 1999.
[16] M. Wiesmann, F. Pedone, A. Schiper, B. Kemme, and G. Alonso, "Database Replication Techniques : A Three Parameter Classification," In Proceedings of 19th IEEE Symposium on Reliable Distributed Systems, Nurnberg, Germany, October, 2000.
[17] B. Kemme, G. Alonso, "A New Approach to Developing and Implementing Eager Database Replication Protocols," ACM Transaction On Database Systems, September, 2000.
[18] Oracle Corporation, 500, Oracle Parkway, Redwoord City, CA 94065, Oracle8i Advances Replication, November, 1998.
[19] 최희영, 황부현, "중복데이터베이스 시스템에서 낙관적인 원자적 방송을 이용한 동시성제어", 정보처리학회논문지 D, Vol. 8-D, No.5, pp.543-552, October, 2001.

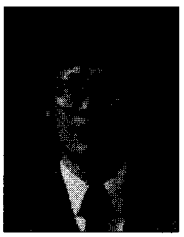
최 희 영

e-mail : hychoi@sunny.chonnam.ac.kr
1987년 목포대학교 전산통계학과(학사)
1989년 전남대학교 대학원 전산통계학과 (이학석사)
2002년 전남대학교 대학원 전산통계학과 (이학박사)
1997년~현재 전남대학교 컴퓨터정보학부 시간강사
관심분야 : 분산시스템, 전자상거래, 중복데이터베이스, 실시간 처리시스템



이 귀 상

e-mail : gslee@chonnam.ac.kr
1980년 서울대 공대 전기공학과 졸업(학사)
1982년 서울대 대학원 전자계산기공학과 (석사)
1982년 금성통신 연구소 근무
1991년 Pennsyivania 주립대학 전산학과 박사
1984년~현재 전남대 컴퓨터정보학부 교수
관심분야 : 멀티미디어통신, 영상처리 및 복원, 화상통신, 논리합성, VLSI/CAD



황 부 현

e-mail : bhhwang@chonnam.chonnam.ac.kr
1978년 숭실대학교 전산학과(학사)
1980년 한국과학기술원 전산학과(공학석사)
1994년 한국과학기술원 전산학과(공학박사)
1980 ~ 현재 전남대학교 컴퓨터정보학부 교수
관심분야 : 분산시스템, 분산데이터베이스 보안, 이동통신, 전자상거래, XML