완전중복 데이터베이스에서 갱신 규약을 적용한 거래 관리

최 희 영† · 황 부 현††

요 약

본 논문은 가용성(Availability)과 신뢰성(Reliability)을 향상시키기 위한 완전중복 데이터베이스에서 거래들의 병행성(Parallelism)의 정도를 높이기 위한 동시성제어 알고리즘을 제안하고 있다. 갱신거래는 논리적으로 판독전용 거래와 갱신전용 거래로 이루어 졌다고 할 수 있다. 제안된 알고리즘에서는 판독연산들을 판독전용 거래로 취급하여 거래가 제출된 사이트에 있는 자료항목을 읽게 하고 갱신할 자료항목들을 모아 갱신전용 거래를 만들어 갱신규약을 통하여 모든 사이트에서 원자적으로 갱신이 이루어지도록 하고 있다. 제안된 알고리즘은 각 사이트에서 거래들이 동시에 수행될 수 있게 함으로서 거래들의 병행수행정도를 향상시킬 수 있다. 특히, 거래들 사이의 충돌빈도가 낮다면 보다 높은 병행수행 정도를 얻을 수 있다.

Transaction Management Using Update Protocol in Fully Replicated Databases

Hee-Young Choi† · Bu-Hyun Hwang††

ABSTRACT

We propose a new concurrency control algorithm for enhancing the degree of parallelism of the transactions in fully replicated databases designed to improve the availability and the reliability. The update transactions can be logically decomposed of a read-only transaction and a write-only transaction. In our algorithm, a set of read operations of an update transaction is treated as a read-only transaction and the read-only transaction reads data items in the site to which it is submitted. And a set of write operations of the update transaction is treated as a write-only transaction and it is submitted to all corresponding sites after the update transaction has been completed. By using the proposed update protocol, all write-only transactions can execute at all sites atomically. The proposed algorithm can have transactions execute concurrently at the site to which they are submitted and, after the completion of each transaction, the update protocol is performed for updating their data items and checking their serializability. Therefore, the degree of parallelism of the transactions can be improved. Especially, if the probability of conflict among transactions is low, we can expect the higher degree of their parallelism.

키워드 : 중복 데이터베이스(Replicated Database), 동시성제어(Concurrency Control), 갱신규약(Update Protocol)

1. 서 론

중복 데이터베이스는 데이터를 중복함에 따라 가용성(availability), 신뢰성(reliability), 견고성(fault tolerance)을 향상시키기 위한 메커니즘으로 사용되어 왔다[1, 3, 6, 7, 15]. 그러나, 데이터가 중복된 모든 사이트에서 데이터의 일관성을 유지하면서 효율적으로 동작하는 거래 관리를 위한 분산 규약을 설계하는 것은 어렵다[1, 4, 5, 7, 9, 10].

중복 데이터 베이스시스템에서 사용되고 있는 중복 규약은 크게 eager 규약과 lazy 규약으로 분류할 수 있다. Eager 규약은 갱신연산의 경우 모든 해당사이트의 갱신연산이 하나의 거래 내에 포함되기 때문에 거래의 수행시간이 길어져 거래 처리비용과 오랫동안 거래가 대기하는 문제가 발생한다. 즉 중복된 모든 사이트는 거래를 수행하는 동안 모든 중복 데이터베이스에게 갱신하고자 하는 데이터 아이템을 동시에 갱신시키기 위하여 원격 통신(remote communication)을 요구한다. 따라서 원격 사이트의 통신 오버헤드(overhead)와 일관성을 유지하기 위한 거래 비용이 높다. 만약 데이터가 중복된 모든 사이트에 동기화 시키는 동시성 제어 기법으로 잠금을 적용한다면 오랜 시간 동안 잠금으로 인하여 동시에 접근하는 거래가 대기되는 문제가 발생한다[7, 11].

Lazy 규약은 지역사이트에서 거래를 수행하고 완료한 후에 비 동기적(asynchronous)방법으로 데이터가 중복된 모든 사이트에 갱신연산을 전파하는 규약이다. 만약 동시에 각 사이트에 접근하는 거래가 있다면 중복되는 데이터 항

※ 본 연구는 한국과학재단 2000년도 목적기초연구 지역대학 우수 과학자 지원연구 (R02-2000-00401)비에 의하여 연구되었음.
† 정 회 원 : 전남대학교 전산학과 강사
†† 정 회 원 : 전남대학교 전산학과 교수 · 전남대학교 정보통신연구소
논문접수 : 2001년 6월 8일, 심사완료 : 2001년 10월 22일

목에 대해서 비 일관된 상태를 유지할 가능성이 높다[3, 4, 6]. lAZY 규약은, 거래가 제출된 사이트에서 거래를 수행하는 동안 다른 중복 사이트와의 원격 통신을 요구하지 않으며, 지역적으로 갱신된 연산들은 모든 중복된 사이트에서 동일하게 수행될 수 있도록 통신 트래픽(traffic)이 없을 때 모든 사이트에 전파하므로 약한 일관성을 제공한다. 따라서 중복 데이터 베이스에서 Lazy 규약은 좋은 성능을 유지하지만 너무 무질서하게 사용한다면 비 직렬성의 결과를 가져올 수 있기 때문에 강한 일관성을 필요로 하는 애플리케이션에는 적합하지 않다[3]. 그러므로, Lazy 규약은 지역 사이트에서 수행되는 거래의 일관성을 보장할 수 있지만 여러 사이트에서 동일한 데이터 항목에 대하여 동시에 접근하는 거래의 전역적 일관성을 보장하기 위해서는 중복 데이터를 가지는 모든 사이트에 수행되는 거래들의 직렬성을 검증하는 절차가 필요하다[1, 5, 6].

제안하는 방법은 Eager 규약에서 발생되는 중복된 모든 사이트를 동기화 시킴에 따라 동시성이 저하되는 문제를 해결하기 위하여 각 사이트에 제출된 거래를 그 지역 데이터베이스에서 독립적으로 수행하도록 하고, 기록연산들은 그 거래의 작업영역에서 수행한 후에 갱신 규약을 사용하여 데이터가 중복된 모든 사이트에서 갱신이 일어나도록 하여 거래의 병행 수행정도가 향상되도록 한다. 중복 데이터 베이스간에 전역적 일관성 문제를 해결하기 위하여 데이터를 중복하고 있는 모든 사이트에서 데이터 항목의 충돌을 검사하는 갱신 순서화 그래프(Update Serialization Graph : UG)를 각 사이트에서 유지하도록 하여 데이터의 일관성을 유지하도록 한다. 이렇게 함으로써 거래들 사이에 충돌 빈도가 적은 경우에는 병행수행 정도를 매우 향상시킬 수 있다.

본 논문의 구성은 다음과 같다. 2장에서는 기존연구에 제시된 중복 데이터베이스 시스템에서 거래 관리 방법과 문제점을 제시하고 3장에서는 본 논문에서 사용하는 시스템 구조와 거래 처리 모델을 제시한다. 4장에서는 갱신 규약을 사용한 중복 데이터베이스 관리 알고리즘과 정확성 검증을 기술한다. 5장에서는 향후 연구되어야 할 방향을 제시한다.

2. 관련 연구

분산 데이터베이스 환경에서 중복데이터 베이스의 관리는 전통적으로 동기식 제어와 비동기식 제어 방법이 있다. 동기식 제어기법은 eager 규약을 사용해서 동시성제어를 하고 비동기식 제어기법은 lazy 규약을 사용한다. 중복 데이터베이스 제어기법은 우선적으로 데이터 항목을 갱신하는 사이트의 위치와 갱신결과의 전파방법에 따라 <표 1>과 같이 분류된다.

〈표 1〉 데이터베이스 시스템에서의 중복 제어 기법

Update Location \ Update Propagation	Eager Primary Copy	Lazy Primary Copy
	Eager Update Everywhere	Lazy Update Everywhere

2.1 Eager 규약

Eager 규약은 거래를 완료(commit)하기 전에 중복된 모든 사이트가 동기화되어 거래를 완료하는 방법으로 강한 일관성을 보장하고 견고성의 정도도 좋다. 거래의 정확성 기준은 one-copy 직렬가능성[2]을 사용하고 있다. 그러나 통신 부담(overhead)과 응답시간이 너무 길어 거래가 지연되는 문제가 발생된다. 이 규약은 데이터 항목을 갱신하는데 갱신 연산을 주 사본에서 실행하는 기법과 주 사본이 아닌 어떤 사이트에서도 거래 갱신이 가능한 제어기법으로 갱신연산을 수행하는 위치에 따라 Eager 주 사본 제어방법과 Eager Update Everywhere방법에 의해서 거래를 수행한다.

Eager 주 사본 제어기법은 하나의 주 사본과 여러 개의 부 사본을 두어서 클라이언트가 각 지역사이트에 제출된 거래를 수행하기 위하여 판독연산은 거래가 발생된 사이트에서 수행하고 갱신연산은 주 사본을 가지는 사이트에서만 수행하는 기법으로 충돌하는 연산들의 수행순서가 주 사본을 갖고 있는 사이트에서 결정되고 부 사본 사이트에서는 주 사본 사이트의 결정에 따라 거래의 수행순서가 결정된다. 따라서 주 사본 사이트의 부하가 심하여 전체적인 거래의 성능이 떨어진다. 그러나, 이 기법은 거래의 강한 일관성을 제공한다. 또한, 이 제어기법은 분산 컴퓨팅의 초기 솔루션[13, 14]으로 사용되었으나, 현재는 견고성을 얻기 위한 예비(backup)메커니즘에 사용되고 있다.

Eager Update Everywhere 제어기법은 주 사본 제어기법에서 발생되는 병목현상을 해결하기 위해 주 사본 사이트에서만 거래를 갱신시키지 않고 모든 사이트에서 갱신을 수행할 수 있도록 하는 기법이다. 이 기법은 read-one/write-all 기법을 사용하여 판독은 거래가 발생된 사이트에서 지역적으로 수행하고, 갱신은 모든 사이트에서 동시 수행되도록 하기 위하여 사이트들을 동기화 시켜서 갱신연산을 수행한다. 이 제어기법은 분산된 잠금(distributed locking)방법을 사용하거나 충돌되는 연산의 순서를 원자적으로 방송(atomic broadcast)하는 두 가지 타입이 존재한다.

분산된 잠금 방법은 주 사본 제어기법보다는 거래의 빠른 응답을 제공하고 있으나 클라이언트가 지역 데이터베이스 서버에 연산을 요구하면 모든 사이트를 동기화 시켜서 데이터 항목을 갱신하기 위하여 모든 서버에서 잠금을 승인 받아 거래를 수행하여야 함으로 동시성과 병행성이 저하된다.

원자적 방송(Atomic Broadcast : ABCAST)을 기본으로 한 데이터 중복은 데이터 중복을 구현하기 위하여 그룹 통신 프리미티브(group communication primitive)사용을 제안하였다. 그러나 실제적인 솔루션[4, 13, 16]에서 그룹 통신 프리미티브를 사용하고 있지만 충분한 깊이를 제공하지 못하는 문제가 있다. 이 방법을 사용한 [13]에서는 분산 시스템의 견고성을 증가시키기 위해서 근거리 통신망(Local Area Network)으로 국한된 그룹 통신 프리미티브를 제안하였다. 이 프리미티브는 원자적 방송 프리미티브의 시맨틱

(semantics)을 사용하였으나 메시지를 모든 사이트에 전달하기 전에 조정단계가 필요하고, 이 조정단계가 끝나기 전까지는 메시지를 전달할 수 없으므로 메시지가 지연되는 문제가 발생하게 된다. 이 문제를 해결하기 위해서 [5]에서는 갱신 거래를 중복 사이트에서 즉시 전달하고 수행한 후에 전달 순서대로 수행을 하였는가 즉 그 거래들의 전역 순서가 존재하는가를 검사하는 낙관적인 방법을 제안하였다. 이 방법을 확장하고 발전시킨 방법으로 [12]에서는 원자적 방송(atomic broadcast)의 방송한 순서로 각 사이트에 전달된다는 가정 하에 해당사이트에서 갱신 거래들을 수행시키고 방송순서대로 전달되어 수행되었는가를 검사하는 낙관적인 방법을 제안하였다. 여기에서 제안된 방법은 제출된 사이트에서 판독 연산을 수행하고 기록 연산을 모든 사이트에서 수행하게 하는 방법보다 판독과 기록 연산 모두가 모든 중복 사이트에서 수행되어야 하기 때문에 중복데이터베이스의 특징중의 하나인 병행성이 저하된다. 또한, 각 사이트에서 동시에 수행되는 갱신 거래의 수가 많을 경우에 거래들의 잠정수행순서(tentative order)와 확정수행순서(definitive order)가 달라질 가능성이 많아 철회되는 거래의 수가 많아지게 된다.

2.2 Lazy 규약

Lazy 규약은 갱신연산을 수행할 때 중복 데이터베이스 서버들 사이의 조정단계를 피하기 위하여 한 서버에 있는 데이터를 갱신하고 완료한 후에 갱신결과를 다른 중복 사이트에 전파한다. 이렇게 함으로서 Eager 중복제어 기법에서 발생되는 동기화 오버헤드를 제거할 수 있다. 이 규약은 Eager 규약과 같이 Lazy 주 사본과 Lazy Update Everywhere기법으로 분류할 수 있다.

거래의 수행은 중복을 제어하는 서버에서만 갱신이 일단 완료되면 그 거래의 완료를 허용하며 완료 후 적당한 시기에 그 서버의 책임하에 다른 중복 사이트에 갱신을 전파하는 방법[6]으로 약한 일관성을 보장한다. 이는 각 사이트가 전파된 갱신을 완료하는 시점이 서로 다를 수 있으므로 전역 일관성을 유지하지 못하는 경우가 발생할 수 있기 때문이다.

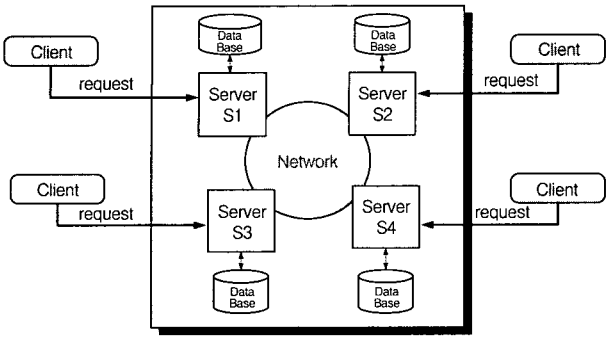
[19]에서는 전역 직렬가능성(global serializability)를 보장하기 위하여 무방향 복사 그래프(undirected copy graph)가 비순환(acyclic) 그래프가 되도록 데이터를 각 사이트에 분산시키는 Lazy 갱신 모델을 제안하였으나, 복사 그래프가 방향성 있는 비순환 그래프인 경우에는 직렬가능성을 보장할 수 없다. 이는 무 방향 복사 그래프가 비순환이 되도록 데이터를 분산시키는 방법이 데이터 분산에 있어서 무방향 복사 그래프가 방향성 있는 경우 보다 많은 제약을 받는다는 것을 의미한다. 이러한 비순환 그래프를 사용하지 않고 전역 직렬가능성을 보장하기 위해서 [7]에서는 Lazy 주 사본 제어기법을 적용하여 판독과 기록연산을 할 때 주 사이트(primary site)로부터 잠금을 얻는데 기록 잠금은 모든 중복 사이트의 복사본에 전파될 때까지 유지되는 기법을 제안하였다. 이 기법은 판독연산을 할 때도 주 사이트를 잠금을 하여야 하므로 잠금 요구를 위한 통신 오버헤드가 높고, 가용성이 상당히 저하된다. [17, 18]에서는 분산시스템내의 모든 거래의 수행에 대한 정보를 포함하고 있는 중복 그래프를 제어 사이트에 유지하여 복사본을 갱신하는 방법을 제안하였다. 이는 제어 사이트의 병목현상을 초래한다. 또한, [8]에서는 새로운 Lazy 주 사본 갱신규약을 제안하였는데, 이 규약은 부분중복 환경에서 주 사본 사이트에서 가지는 데이터를 다른 사이트에서 복사본으로 가지고 있다면 이들 사이에 에지가 연결되는 복사 그래프를 유지한다. 이 논문이 가지는 특징은 복사 그래프가 DAG이면 직렬가능성을 보장한다는 것인데, 복사그래프가 DAG라는 조건은 자료의 분산에 제약을 가하게 되며, 항상 방향성있는 비순환 그래프를 유지해야 한다. 이러한 DAG의 요구를 제거한 규약으로 백 에지 규약을 제안하였다. 백 에지 규약은 Eager 규약과 Lazy 규약을 결합한 혼합형 규약(hybrid protocol)으로 DAG에 사이클이 발생하는 경우에 백 에지를 사용해서 해결하는 방법이다. 그러나 백 에지의 시작점과 끝점의 노드 사이의 경로가 길다면 백 에지가 존재하는 거래의 잠금 소유 시간이 길어져서 다른 거래들의 수행을 대기(block)시켜 전체적인 성능 저하를 가져온다. 또한 직렬가능한 수행을 하지 못하게 될 경우 전역 교착상태가 발생하게 되는데, 이것을 해결하기 위해서는 많은 통신 오버헤드와 비용이 들게 된다.

따라서 본 논문에서는 거래들의 동시성과 병행성을 향상시킬 수 있도록 Eager Update Everywhere 기법에 기반을 둔 낙관적 동시성 제어방법을 사용하여 중복 데이터베이스 환경에서 거래를 관리하는 방법을 제안한다. 제안하는 방법은 제출된 사이트에서 거래의 모든 연산이 수행된 후에 갱신 규약을 수행한다. 각 거래는 작업영역을 가지고 있어서 완료할 때까지의 갱신결과는 작업영역에 보관한다. 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. 갱신 전용 거래는 해당 중복 사이트에서 다른 거래들과 동시에 수행이 된다. 갱신 규약이 완료되어야 거래는 완전히 완료를 하게된다. 이렇게 함으로서 각 사이트에 제출된 거래들은 해당 사이트에서 독립적으로 모든 연산을 수행할 수 있음으로 거래의 병행성을 향상시킬 수 있다. 이 갱신 규약이 가지는 특징은 첫째, Eager 기법에서 발생할 수 있는 문제인 거래 수행중 모든 사이트를 동기화 함에 따라 다른 거래들이 지연되는 문제를 해결할 수 있다. 둘째, 전역적 일관성이 유지될 때만 모든 중복 사이트에서 갱신 전용거래가 수행될 수 있도록 하였다.

3. 시스템 모델

본 논문에서 사용되는 시스템 구조는 (그림 1)과 같다. 각 클라이언트는 그의 지역 데이터 베이스의 서버에게 거

래 처리를 요구할 수 있다. 각 서버에 연결되어 있는 데이터베이스는 데이터를 완전 중복하고 있는 지역 데이터베이스이다. 지역 데이터베이스의 서버는 안정적이고 신뢰할 수 있는 네트워크로 연결되어 있다.



(그림 1) 시스템 구조

중복 데이터베이스 시스템에서 거래는 거래가 제출된 서버에서 수행되는 주프로세서와 거래가 요구하는 중복데이터가 위치한 서버에서 수행되는 부프로세스들로 구성된다[10]. 주프로세스는 조정자(coordinator)의 역할을 하고, 각 복사본을 가지는 사이트에서 수행되는 부프로세스는 참여자(participant)의 역할을 한다. 즉, (그림 1)에서 지역 데이터베이스 서버는 조정자가 되기도 하고 참여자가 되기도 한다.

클라이언트가 각 지역 데이터베이스 서버에 제출한 거래는 판독(read)과 기록(write) 연산으로 구성되어 있다. 기록연산을 포함하고 있는 거래를 갱신 거래(update transaction)라고 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 갱신결과는 작업영역에 보관한다. 이 보관된 갱신결과를 다른 중복 사이트에 전파하기 위하여 갱신연산들로 이루어진 갱신 전용 거래를 생성한다. 갱신 규약은 갱신 전용 거래가 모든 중복사이트에서 일관성이 위배되지 않게 수행할 수 있도록 한다. 갱신 전용 거래는 해당 중복 사이트에서 다른 거래들과 동시에 수행이 된다. 갱신 규약이 완료되어야 거래는 완전히 완료를 하게된다.

4. 갱신규약을 사용한 중복 데이터베이스 관리 알고리즘

이 장에서는 중복된 모든 데이터베이스에서 일관성 유지를 위한 갱신규약에서 필요한 가정을 하고, 갱신규약에서 요구되는 자료구조를 살펴본다. 그리고, 갱신 규약을 사용한 중복 데이터베이스에서의 거래 관리기법을 제안한다.

4.1 가정 및 자료구조

제안하는 갱신규약을 사용한 중복 데이터베이스 관리 알고리즘은 다음과 같은 가정을 전제로 하고 있다.

【가 정】

1. 데이터 항목은 모든 지역 데이터베이스에 완전 중복되어 있다.
2. 각 거래는 그의 작업영역을 가지고 있고, 갱신연산은 작업영역에서 이루어진다.
3. 거래는 제출된 지역사이트에서 모든 연산이 수행된 후에 갱신규약을 시작한다.
4. 갱신규약이 완료된 후에 거래는 완료된다.
5. 네트워크는 메시지 손실이 없고 보내는 순서대로 전달된다.

가정 1은 시스템의 가용성, 신뢰성, 병행성의 향상을 위하여 제안하는 알고리즘에서 토대가 되는 가정이다. 가정2는 거래의 병행성 향상을 위하여 거래는 제출된 사이트에서 독립적으로 수행이 되며, 갱신결과를 데이터베이스에 반영하기 전에 보관하기 위한 작업영역이 요구되므로 필요하다. 각 서버에서 제출된 거래마다 작업영역을 할당하면 되기 때문에 이 가정은 구현상의 문제를 가지고 있지 않다. 가정 3은 독립적인 수행과 병행성 향상을 위하여 존재한다. 가정 4는 제안하는 알고리즘이 Eager기법을 채택하고 있음을 의미한다.

제안하는 알고리즘을 위해 모든 서버에서 유지하는 자료구조와 의미는 <표 2>와 같다.

〈표 2〉 서버에 유지되는 자료구조

자료구조	의 미
UG (Update Graph)	갱신규약에 의해 동시에 접근하는 데이터 항목에 대하여 충돌이 발생되는 거래를 에지로 연결하여 표현되는 갱신그래프
Node(Tr_id, State)	• Tr_id는 거래 번호 • State는 거래 상태를 나타내는 마크
$RS(T_i)/WS(T_i)$	거래 T_i의 판독 집합/거래 T_i의 기록집합
$ts(T_i)$	거래 T_i의 타임스탬프
After	읽은 데이터항목을 갱신한 갱신전용 거래의 소스정보
Table	각 데이터 항목마다 갱신한 거래에 대한 소스정보를 유지
Tr_list	제출된 거래 리스트

각 사이트에서 수행되는 거래는 독립적인 작업영역을 가지고 있고, 이 작업영역에 갱신연산을 기록한다. 또한 각 사이트에서 동시에 수행되는 거래의 직렬성을 보장하기 위하여 각 거래는 판독항목들의 집합 *RS*, 갱신한 항목들의 집합 *WS*, *After*의 정보를 가지고 있고, 각 데이터 항목은 그의 갱신전용거래의 소스(source)에 관한 정보를 가지고 있다. 소스는 갱신 그래프에 있는 노드 중 입력에지(in-edge)가 없는 노드를 소스노드라 한다. 각 거래의 *After*는 읽은 데이터항목을 후에 갱신한 갱신전용 거래의 소스정보를 가지

며, Table은 각 데이터 항목을 갱신할 때 갱신전용거래의 소스정보를 유지한다. Tr-list는 각 사이트에서 제출된 거래 리스트를 나타낸다.

4.2 제안하는 중복 데이터베이스 관리 알고리즘

제안하는 중복 데이터베이스 관리 알고리즘은 각 사이트에 제출된 거래를 그의 지역 사이트에서 독립적으로 거래의 마지막 연산까지 수행하는데 거래리스트에 등록하고 거래를 수행하면서 판독한 데이터 항목에 대한 판독잠금을 설정한다. 이 판독 잠금은 기존의 2PL에서 사용되는 잠금의 방법과는 달리 다른 잠금과 공유가 가능하다. 즉 이미 판독 잠금이 걸린 데이터에 대해서도 기록 잠금을 승인할 수 있다. 그리고, 판독한 데이터 항목들의 집합 *RS*와 작업영역의 갱신 데이터 항목들의 집합 *WS*를 구한다. 이때 기록연산은 독자적인 거래의 작업영역에다 기록하고, 거래의 마지막 연산을 수행한 후에 작업영역에 있는 갱신하는 자료들로 구성된 갱신전용 거래를 수행하기 위한 갱신규약을 시작한다. 갱신 규약은 다른 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 갱신 중에 현재 수행중인 거래가 설정한 판독 잠금이 있다면 그 수행중인 거래의 *After*에 갱신전용거래의 소스번호를 추가한다. 소스번호를 추가한 이유는 갱신전용거래가 UG로부터 제거되더라도 수행중인 거래에 갱신전용거래와의 관계를 기억하기 위하여 필요하다. 그리고 갱신전용거래가 데이터베이스에 그의 *WS*가 가지는 갱신 값을 기록하고자 할 때 모든 데이터 항목에 대해서 기록잠금을 설정하고 기록한 즉시 기록잠금을 해제시킨다. 이러한 잠금의 사용은 지역 거래가 판독한 데이터를 갱신전용거래에 의해서 갱신되는 것에 의해 일관성이 위배될 수 있는 경우가 발생된다. 그러나 지역거래에 의해서 판독하고 아직 갱신규약에 진입하지 않는 거래들이 데이터 항목을 기록할 때 데이터를 판독한 거래의 *After*에 자신의 소스번호를 추가하고 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. 다음은 제안하는 알고리즘에서 중복된 데이터 관리를 위한 갱신 규약에 따른 조정자 알고리즘과 참여자 알고리즘에 관해서 기술한다.

4.2.1 갱신 규약의 조정자 알고리즘

제안하는 방법에서는 거래가 발생된 사이트는 조정자가 되고, 복사본을 가지고 갱신 규약에 참여하여 기록연산을 행하는 모든 지역 사이트를 참여자라 한다. 조정자 사이트는 제출된 거래의 동시성을 제어하며 모든 사이트에서 일관성 있게 데이터 항목이 갱신 될 수 있도록 관리한다. 먼저 기록 연산들로 구성된 갱신 전용 거래를 모든 사이트에서 수행하기 위하여 모든 참여자에게 갱신 메시지인 갱신전용거래를 전파하는 것이고, 또 다른 하나는 모든 참여자 사이트에서 오는 응답을 모아 완료와 철회를 결정하는 것이다. 조정자 사이트에 제출된 거래는 그의 지역사이트에서 판독 연산을 하고 기록 연산들을 수행토록 하는데, 이 기록 연산은 지역적으로 독자적인 공간에 데이터 항목에 대한 갱신 값을 유지한다. 이 갱신 된 값은 아직 데이터베이스에 반영되기 전이므로 다른 거래들은 판독할 수 없다.

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

갱신요구를 한 갱신전용거래의 조정자는 해당거래의 모든 참여자 사이트에 이 거래에 대한 갱신전용거래를 제출하고, 각 참여자 사이트로부터 갱신요구에 응할 수 있는지(*YES*) 없는지(*NO*)의 응답을 받는다. 조정자 사이트는 모든 참여자 사이트로부터 *YES*를 응답 받으면 갱신가능상태(Update) 메시지를 모든 참여자 사이트로 보낸다. 그리고, 갱신완료 메시지를 모든 참여자 사이트로부터 기다리며, 모든 참여자로부터 갱신완료 메시지를 받은 후 해당 거래를 완료할 수 있다. 이에 대한 알고리즘은 (알고리즘 1)과 같다.

판독전용거래는 판독연산만을 포함하고 있기 때문에 지역적으로 이루어진다. 지역적으로 이루어 진 판독전용거래가 데이터 x에 대해서 판독할 때 이미 갱신 전용거래에 의해서 갱신되었다면 그 데이터가 유지한 소스번호를 After에 가져오고, 판독전용거래가 다른 데이터 y를 판독할 때 그 데이터가 가지는 소스번호와 교집합이 발생하면 이 판독 전용거래는 사이클이 발생되므로 지역거래를 철회시킨다. 따라서 지역거래와 갱신전용거래 사이에 사이클이 발행되지 않도록 스케쥴한다.

```
/* 갱신 규약의 조정자 */
/* 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
        commit for the transaction T and release all lock of T
```

(알고리즘 1) 갱신 규약의 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. 여기에서 UG는 지역 사이트에서 마지막 연산까지 수행한 거래의 갱신연산들로만 구성된 갱신전용거래를 나타내는 노드로 구성된다. 노드는 거래의 번호와 상태를 나타내는 필드를 가진다. UG의 상태가 가질 수 있는 값은 *'Vote'와 'Update', 'Submit', 'Commit'*으로 표기한다. *Vote*는 UG그래프 검사를 마치고 조정자에게 *YES/NO*를 투표한 거래를 나타내고, *Update*는 조정자가 완료를 결정해서 모든 사이트로 갱신전용거래를 보냈다는 의미이고, *Submit*는 UG내의 노드중 입력에지가 없는 노드로서 각 지역 데이터베이스로 거래를 제출한다는 의미이다. 또한 *Commit*는 제출된 거래가 완료되었다는 것을 나타낸다. UG의 노드가 입력에지를 가지지 않는다는 것은 충돌되는 거래가 존재하지 않는다는 것이고, 이 노드를 소스(source)노드라 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) $WS(Tug) \cap RS(T) = \{x\}$: 생성된 에지 T -> Tug (Tug 투표한 거래)

(조건 2) $RS(Tug) \cap WS(T) = \{x\}$: 생성된 에지 Tug -> T (Tug 투표한 거래)

(조건 3) $((WS(Tug) \cap WS(T) = \{x\})$ and $((WS(Tug) \cap RS(T) = \{\})$ and $(RS(Tug) \cap WS(T) = \{\})))$

(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

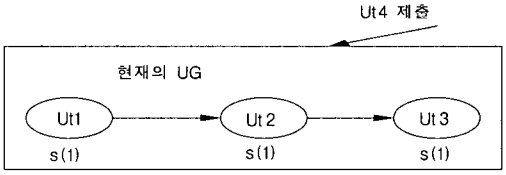
조건 1에서 거래 T는 갱신전용거래 Tug가 데이터항목을 갱신 전에 그 데이터 항목들을 읽었기 때문에 T가 직렬화 순서에서 Tug를 선행하여야 한다는 의미이다. 조건 2 역시 거래 Tug는 T가 데이터항목을 갱신하기 전에 그 데이터 항목에 대한 판독연산을 수행하였다는 것을 의미한다. 이는 각 사이트에서 독립적으로 수행하고 나서 갱신전용거래를 수행하기 때문이다. UG에서 T1 → T2의 관계는 직렬화 순서에서 T1이 T2를 선행한다는 것을 의미한다. UG에서 입력에지를 갖지 않는 갱신가능 상태의 거래는 지역 데이터베이스에 제출되어 수행될 수 있다. 이 노드는 소스가 되며 소스번호를 가지고 있다. 소스번호는 UG에 갱신전용거래를 추가할 때 소스 번호가 부여되고, 같은 소스를 가진 노드는 동일한 소스번호를 가진다. 소스번호의 부여는 UG에 추가하고자 하는 갱신전용거래의 노드 에지가 입력 에지일때 바로 전 노드의 소스번호를 자신의 소스번호로 생성하고, 출력 에지(out-edge)이면 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 생성한다. 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. 여기에서는 주기적으로 필요 없는 소스번호를 갱신시킨다고 가정한다. UG에서 완료된 노드 삭제는 갱신 규약 거래가 지역 스케쥴러로 제출할 때 새로운 타임스탬프를 부여받아서 각 사이트에서 전파한 갱신규약의 타임스탬프와 비교하여 완료된 거래의 타임스탬프가 더 작은 노드만을 삭제시킨다. 소스번호의 전파이유는 다음 예제 2로 설명한다.

【예제 1】 소스번호 생성 예

사이트 S에서 UG의 현재 상태가 (그림 2)의 (a)와 같을 때 갱신전용거래 Ut4가 도착하였다고 하자.

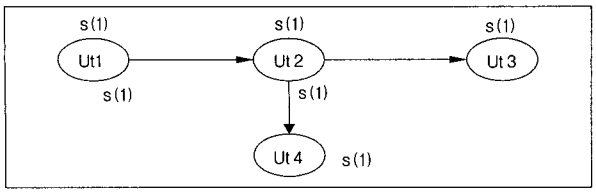
① 생성된 에지가 입력에지인 경우

T1 : r(x)w(a)	*RS* = {x}	*WS* = {a}
T2 : r(z)r(y)w(x)	*RS* = {y,z}	*WS* = {x}
T3 : r(b)w(y)	*RS* = {a}	*WS* = {y}
T4 : r(c)w(z)	*RS* = {c}	*WS* = {z}



범례 -
Ut : 갱신규약에 의해서 기록 연산들로만 구성된 갱신 전용 거래
s : 소스정보를 나타냄

(a) Ut4를 UG에 추가하기 전 상태



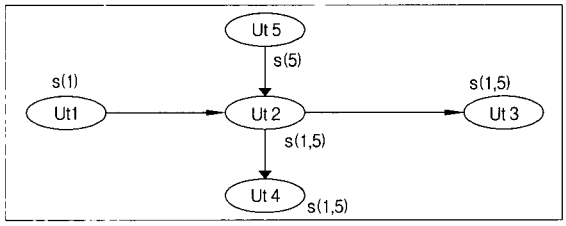
(b) Ut 4를 추가했을 때 소스번호

(그림 2) T4의 소스번호를 부여하기 전과 후의 상태

(그림 2)의 (a)는 UG에 Ut4를 추가하기 전이고 (b)는 Ut4를 각 사이트에 전파했을 때 데이터 z는 T2 연산 z와 충돌관계이므로 Ut2 →Ut4의 에지가 생성이 되는데, 생성된 에지는 입력 에지를 가지므로 인접된 노드의 소스번호를 자신의 소스번호에 추가한다.

② 생성된 에지가 출력에지인 경우

①의 (b)상태에서 T5 = r(x)w(b)이 지역 사이트에 제출되었다고 하자. T5가 갱신 규약에 의해서 Ut4 → Ut5의 에지가 만들어진다면 UG의 노드에 (그림 3)과 같은 소스번호를 가진다.



(그림 3) Ut5를 추가했을때 소스번호

(그림 3)에서 거래의 충돌로 인해 Ut5에서 Ut2로 가는 출력에지가 만들어진다면 Ut5는 소스노드이므로 소스번호를 자신의 거래 번호로 생성한 다음 같은 경로를 가지는 노드에 소스번호를 전파하면 Ut2와 Ut4는 Ut5의 소스번호에 추가되어 s(1, 5)라는 소스번호를 가진다. □

【예제 2】 소스번호의 전파이유

UG는 Ut1 → Ut2라고 가정하고 한 지역 사이트에서 거래 T_i가 수행 중에 Ut1이 T_i가 판독한 데이터 항목을 갱신한다고 하자. 그리고 T_i의 수행이 끝난 후에 T_i의 갱신전용 거래 Ut_i가 각 사이트에서 수행된다고 하자. 이때 T_i → Ut1 → Ut2 → Ut_i의 수행순서가 존재하게 된다. 이는 Ut_i는 T_i이므로 직렬성이 파괴된다. 그러나, Ut_i의 갱신규약 수행 때까지 Ut1이 UG에 존재하면 UG에서 사이클이 생성되어 Ut_i의 투표 결과는 *NO*가 되어 T_i는 철회가 되므로 직렬성을 보장하게 된다. 즉, Ut1의 소스번호는 Ut2의 소스번호가 된다. T_i의 *After*에는 소스번호 1이 있고, Ut_i를 UG에 첨가할 때 인접노드 Ut2의 소스번호가 1이므로 T_i 에서 Ut_i까지 충돌경로가 존재한다는 것을 의미한다. 갱신규약을 수행중에 *After*에 UG에서 Ut_i의 인접노드의 소스번호가 존재하면 *NO*를 투표하고 T_i를 철회함으로서 직렬성을 보장할 수 있다. 만약 Ut1이 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 마지막 노드는 삭제시키지 않고 남겨두며, 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 포함되어 있지 않으면 UG에 사이클이 존재하는가를 검사한다. 사이클이 존재하지 않으면 *Yes*를 투표하여 조정자 사이트로부터 응답을 기다린다. 이때 투표한 거래를 나타내는 노드를 '*Vote*'라 마크한다. 참여자 사이트는 조정자 사이트로부터 '*Update*' 메시지를 받으면 UG에서 해당 거래를 '*Update*'로 마크한다. 해당 거래의 노드가 입력 에지를 갖지 않으면 그 거래는 소스가 되므로 지역 스케쥴러에게 제출하여 실행시킨다. 따라서 UG는 갱신규약을 요구한 거래가 각 사이트에서 갱신을 할 수 있음을 검사하기 위하여 사용되며, 갱신규약의 참여자는 갱신규약을 요구한 거래가 전역 직렬성을 파괴하지 않고 수행될 수 있는가의 결과(YES/NO)를 투표한다. 그러나 지역사이트에서 아직 갱신 규약에 의해 전파되지 않는 거래가 기록 잠금을 요구한 데이터 항목의 리스트에 포함되어 있으면 갱신 전용거래의 소스정보를 지역거래의 *After*에 삽입하고 기록잠금을 얻어 수행한 후 잠금을 해제한다. 실행이 완료되면 갱신완료 메시지를 조정자 사이트로 보내고 UG에서 그 거래 노드를 제거한다. 따라서 잠금으로 인한 거래 지연이 발생하지 않고 동시성을 향상시킬 수 있다. 이를 위한 참여자 알고리즘은 (알고리즘 2)와 같다. 참여자사이트는 갱신 규약에 의해 전파된 갱신 전용거래에 대하여 직렬가능한 순서가 있는지를 검사하는 알고리즘은 (알고리즘 3)과 같다.

```
/* 갱신규약에 따른 참여자 알고리즘 */
/* 갱신전용거래 = (Tr_id, RS(T), WS(T), TS(T), After)
   Ut : 거래 T에 대한 갱신 전용 거래                */

 When receive a 'write-only transaction' from coordinator site
/* 1. 직렬성 검사 알고리즘을 이용하여 갱신전용거래 Ut의 직
      렬성검사 */
 call check_serializability_procedure
 if (the execution sequence of serializability exists)
 then send the 'YES' massage to the coordinator and marked
 the state as 'Vote'
 else send the 'NO' massage to the coordinator. and marked 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'
      if (Ut is the source node and state=='Update')
      then submit Ut to the local scheduler
           marked the state in source node as 'Submit'.
      else wait.
    else abort T
/* 3. 완료 */
 When receive commit message from coordinator site
 commit for the transaction Ut
```

(알고리즘 2) 갱신규약의 참여자 알고리즘

```
/* 직렬성검사 알고리즘 */
 check_serializability_procedure
 Tug : UG에서의 임의의 노드
 Ut : 거래 T에 대한 갱신 규약에 의해 모든 사이트에 전달된 갱신
      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) ≠ {}) and (WS(Tug) ∩ RS(T) = { })
                             and (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"

/* 소스번호 부여하는 단계 */
   if(in-edge of Ut)
   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

/* 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
```

(알고리즘 3) 직렬성 검사 알고리즘

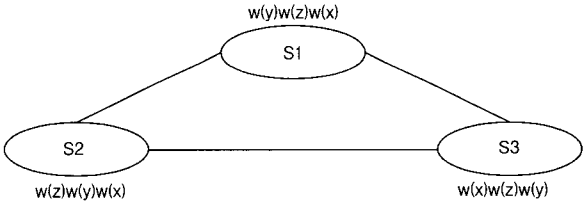
다음 예제는 3개의 사이트로 이루어진 완전 중복 데이터베이스이다. 그리고, 3개의 사이트에서 동시에 갱신규약 요구를 하였다고 가정한다.

【예제 2】 완전 중복 데이터베이스에서 거래관리

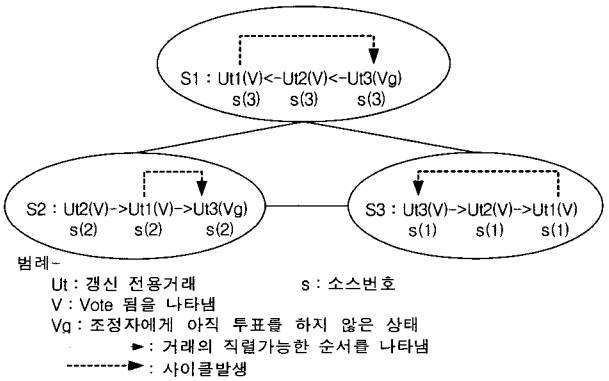
데이터 항목 x, y, z가 사이트 1, 2, 3에 완전 중복되어 있다. 거래 T1은 사이트1, 거래 T2는 사이트2, 그리고 거래 T3은 사이트3에 제출된다고 가정할 때 각 거래와 그가 판독하는 자료항목들의 집합과 갱신하는 자료항목의 집합이 다음과 같다.

사이트 1 : T1 : r1(x)w1(y) $RS(T1) = \{x\}$, $WS(T1) = \{y\}$
T1의 갱신전용거래 Ut1 : w1(y)
사이트 2 : T2 : r2(y)w2(z) $RS(T2) = \{y\}$, $WS(T2) = \{z\}$
T2의 갱신전용거래 Ut2 : w2(z)
사이트 3 : T3 : r3(z)w3(x) $RS(T3) = \{z\}$, $WS(T3) = \{x\}$
T3의 갱신전용거래 Ut3 : w3(x)

세 사이트에서 거래의 수행을 마치고 동시에 갱신규약을 수행한다고 하자. 시점 t1에서 사이트 s1에 Ut1, 사이트 s2에 Ut2, 사이트 s3에 Ut3가 도착하였다고 하자. 이 때 각 사이트의 UG는 하나의 노드로만 구성되어 있기 때문에 사이클이 존재하지 않으므로 각 사이트 모두 *YES*를 투표할 수 있다. 시점 t2에서 사이트 s1에 Ut2, 사이트 s2에 Ut1, 사이트 s3에 Ut2가 도착하였다고 하자. 역시 이 때에도 각 사이트의 UG는 사이클을 포함하고 있지 않으므로 각 사이트 모두 갱신전용 거래에 대하여 *YES*를 투표한다. 시점 t3에서 사이트 s1에 Ut3, 사이트 s2에 Ut3, 사이트 s3에 Ut1이 도착하였다고 하자. 이 때 각 사이트에 마지막 갱신전용 거래가 도달되어 처리된다. 가장 먼저 사이클을 검출한 사이트에서 사이클을 유발한 갱신전용거래에 대하여 타임스탬프가 가장 크다면 *NO*를 투표하고 UG에서 해당 노드를 제거한다. 그렇지 않으면 *YES*를 투표한다. 한 사이트에서 사이클이 발생하게 되면 모든 사이트에서 동일한 사이클이 발생하게 된다. 지역 사이트에서 갱신 전용거래는 (그림 4)와 같고, (그림 5)는 각 사이트에서 UG에서 발생되는 사이클은 동일한 사이클이 존재한다는 것을 보여주고 있다.



(그림 4) 지역 사이트에서 갱신전용거래



(그림 5) 지역 사이트에서 유지되는 UG

시점 t3에서 사이트 s1과 s2는 T3의 갱신전용 거래를 받아 사이클을 탐지하고 사이클에서 Ut3의 타임스탬프가 가

장 크므로 *NO*를 투표하고, 완료규약 수행결과 각 사이트의 UG에서 T3이 제거가 되어 사이클이 없어지기 때문에 T1과 T2는 정상적으로 완료규약을 끝내고 완료할 수 있다.

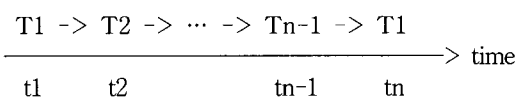
한 사이트의 UG에서 사이클이 생성되면 다른 모든 사이트에서도 동일한 사이클이 생성되게 된다. 이는 완전중복 데이터베이스이고 갱신이 모든 사이트에서 이루어져야 하기 때문이다. 그러므로, 사이클이 생성되면 사이클 내에서 큰 타임스탬프를 갖는 거래에 대하여 *NO*로 투표하고 각 UG에서 그 거래를 제거함으로서 사이클을 없앤다. 사이클 내의 큰 타임스탬프를 갖는 거래가 이미 *YES*를 투표하였어도 문제가 발생하지 않는다. 이는 다른 사이트에서 가장 큰 타임스탬프를 갖는 거래의 노드가 UG에 추가될때 그 거래에 대하여 *NO*를 투표하고 각 사이트의 UG에서 그 거래가 제거되어 사이클을 없앨 수 있다.

거래들 사이에 충돌이 적을 때 사이클이 생성될 가능성이 적기 때문에 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.

【정리 1】 ***갱신규약을 사용한 중복데이터베이스 관리 알고리즘은 거래들이 직렬 가능한 수행을 보장한다.***

(증명) 각 지역 사이트에서는 수행중인 거래 T와 갱신전용 거래 Ut사이의 충돌이 발생할때 T가 판독한 항목을 Ut가 갱신하려고 하면 T의 *After*에 Ut의 소스가 추가된다. 각 지역에서 갱신전용 거래들은 UG를 구성하는데 UG에서 소스이며 *Update*로 마크되어있을 때만 지역 스케쥴러에 의하여 스케쥴되어 수행이 된다. 약간의 시간 차이는 있을 수 있지만 각 사이트의 UG는 동일한 그래프이다. 이는 통신지연에 의하여 발생한다.

증명을 위해 거래들 사이에 직렬성이 보장되지 않는다고 하자. 이때 다음과 같은 완료된 거래들 사이의 직렬화 그래프 SG에 사이클이 발생하게 된다.



위와 같은 사이클이 형성이 되었다는 것은 T2, T3, ···, Tn-1은 갱신규약을 완료하고 데이터 베이스에 갱신하고자 하는 값을 갱신중인 거래이고 T1거래는 지역 사이트에서 수행중인 활성거래이거나 갱신규약을 완료하고 US에 포함된 갱신중인 거래일 수 있다. 이들 경우를 살펴보면 다음과 같다.

(a) T1이 지역에서 수행중인 거래일 경우

시점 t1에서 T1의 *After*는 T2의 소스를 포함하게 된다. 제안된 알고리즘에 의하면 T2 -> ··· -> Tn-1과 같은 경로가 존재하므로 T2의 소스와 Tn-1의 소스는 동일하게 된다. 그러므로, 시점 tn에서 T1이 Tn-1이 갱신한 데이터 항목을 판독하려고 할 때 지역 사이트에서 T1의 *After*에 T1이 판독하려고 하는 항목의 소스가 포함되어 있기 때문에 T1은 지역 사이트에서 철회되었어야 한다. 이는 제안된 알고리즘의 스케쥴링 정책에 모순이 된다. 그러므로, 위와 같은 사이클은 존재하지 않는다.

(b) T1이 갱신완료를 한 거래인 경우

위의 사이클이 형성이 되었다는 것은 시점 tn에서 T1은 갱신완료를 하였다는 의미이다. UG에서 T2가 제거되지 않은 경우는 T1이 갱신규약을 수행할 때 UG에서 사이클이 형성이 되어 T1의 갱신전용 거래는 *NO*를 투표받고 철회가 된다. 그리고, T2의 갱신전용거래가 UG에서 제거된 후에 갱신규약을 수행한 경우 T2 -> ··· -> Tn-1과 같은 경로가 존재하므로 T2의 소스와 Tn-1의 소스는 동일하게 된다. 그러므로, 시점 tn에서 T1의 갱신전용거래의 인접된 입력노드인 Tn-1의 소스가 T1의 *After*에 포함되어 있기 때문에 T1의 갱신규약 중 *NO*를 투표하고 철회되게 된다. 그러므로, 위와 같은 사이클은 존재할 수가 없다.

따라서 본 논문에서 제안한 알고리즘은 (a)와 (b)의 결과로 인해 직렬성이 보장됨을 알 수 있다.

5. 결론 및 향후 연구방향

본 논문에서는 가용성과 신뢰성을 향상시키기 위하여 데이터가 중복된 분산시스템 환경에서 거래의 병행수행의 정도를 높이기 위한 동시성 제어 규약을 제안하였다. 제안된 동시성 제어 기법은 각 사이트에서 동시에 수행되는 거래들의 병행성의 정도를 높이기 위하여 각 사이트에 제출되는 거래는 독립적으로 그 지역 사이트에서 마지막 연산까지 수행을 마친 후에 그의 작업영역에 있는 갱신결과를 전파하기 위한 갱신전용거래를 생성한 후에 갱신규약을 수행한다. 거래가 일관되게 갱신한다는 것은 1-사본 직렬가능성이 보장되어야 한다. 즉 모든 사이트가 서로 일관된 값을 유지하면서 동일한 순서에 의해 갱신 거래를 수행한다는 것이다. 이를 위해 모든 사이트에서 동시에 수행되는 갱신전용거래가 직렬가능 순서로 수행되어야 한다. 각 사이트에서 동일한 직렬가능 순서로 거래들이 수행되도록 하기 위하여 갱신규약을 사용하여 갱신전용거래들이 모든 사이트에서 동일한 직렬가능 순서로 수행되도록 하였다. 그리고, 각 사이트에서 수행중인 거래와 갱신중인 거래들 사이의 직렬가능성을 보장하기 위하여 소스라는 개념을 사용하였다. 동시에 완료규약을 수행한 거래들 사이의 사이클 형성을 방지하기 위하여 그들의 타임스탬프 값을 적용하여 가장 큰 타임스탬프를 갖는 거래를 철회하도록 하였다. 중복 데이터베이스에서 갱신규약을 사용한 거래관리 알고리즘은 갱신거래들 사이에 충돌발생이 적으면 병행수행의 정도가 높아져서 좋은 성능을 보인다. 즉, 각 사이트에서 제출된 거래가 동시에 읽고 필요한 연산을 수행한 후 갱신이 필요한 경우에 갱신규약을 사

용하여 모든 사이트에 갱신을 한다. 해당사이트에서 갱신하는데 걸리는 시간은 읽고 처리하는 시간에 비하여 상당히 짧다. 그러므로, 거래가 동시에 수행되는 병행수행의 효과를 얻을 수 있다. 제안된 알고리즘은 중복 데이터베이스를 사용함으로서 가용성과 신뢰도를 향상시키고, 해당 사이트에서 모든 연산을 수행함으로 병행성의 정도를 높일 수 있다.

향후 연구되어야 할 방향은 본 논문에서 제안한 갱신규약 알고리즘에의 통신비용을 줄이는 문제와 거래들의 충돌 정도에 따른 정확한 성능평가가 이루어져야 한다. 또한, 모든 사이트에서 동시에 수행되는 갱신전용거래들의 직렬가능 순서를 보다 효율적으로 보장하는 방법에 관한 연구가 더 이루어져야 할 것이다.

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] F. Pedone, A. Schiper, "Optimistic Atomic Broadcast," In Proceedings of the 16th International Symposium on Distributed Computing, September, 1998.

[6] M. Wiesmann, F. Pedone, A. Schiper, B. Kemme, and G. Alonso, "Understanding Replication in Databases and Distributed Systems," In Proceedings of 20th International Conference 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] P. A. Bernstein, N Goodman, "An Algorithm for Concurrency Control and Recovery in Replicated Distributed Database," ACM Trans. on Database Systems, Vol.9, pp.596-615, 1984.

[10] M. Oszu and P. Valduriez, Principles of Distributed database management, Prentice-Hall, 1991.

[11] J. Gray, P. Homan, H. Korth, and R. Obermark. "A Strawman Analysis of the Probability of Wait and Deadlock," Technical Report RJ2131, IBM San Jose Research Laboratory, 1981.

[12] 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.

[13] 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.

[14] P. Alsberg, and J. Day, "A Principle for Resilient Sharing of Distributed Resource," In Proceeding of the International Conference on Software Engineering, Oct. 1976.

[15] M. Stonebraker, "Concurrency Control and Consistency of Multiple Copies of Data in Distributed INGRES," IEEE Transactions on Software Engineering, SE-5 : 188-194, May. 1979.

[16] A. Schiper and M. Raynal, "From Group Communication to Transaction in Distributed System," Communications of the ACM, 39(4) : 84-87, Apr. 1996.

[17] 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.

[18] 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.

[19] P. Chundi, D. J. Rosenkratz, and S. S. Ravi, "Deferred Updates and Data Placement in Distributed Databases," In Proceedings of the Twelveth International Conference on Data Engineering, New Orleans, Louisianal, 1996.

최 희 영

e-mail : hychoi@sunny.chonnam.ac.kr

1987년 목포대학교 전산통계학과(학사)

1989년 전남대학교 대학원 전산통계학과 (이학석사)

1999년 전남대학교 대학원 전산통계학과 (박사과정 수료)

1997년~현재 전남대학교 전산학과 시간강사

관심분야 : 중복 데이터베이스, 전자상거래, 실시간 시스템, 분산시스템

황 부 현

e-mail : bhhwang@chonnam.chonnam.ac.kr

1978년 숭실대학교 전산학과 (학사)

1980년 한국과학기술원 전산학과(공학석사)

1994년 한국과학기술원 전산학과(공학박사)

1980년~현재 전남대학교 전산학과 교수

관심분야 : 분산시스템, 분산 데이타베이스 보안, 객체지향 시스템, 전자상거래