■ 2000 정보과학 논문경진대회 수상작

클라이언트-서버 환경에서 캐쉬된 공간 데이타의 동시성 제어 및 일관성 유지 기법

(Concurrency Control and Consistency Maintenance of Cached Spatial Data in Client-Server Environment)

신 영 상 † 홍 봉 희 ††
(Youngsang Shin) (Bonghee Hong)

요 약 클라이언트-서버 공간 데이타베이스에서는 대용량의 공간 데이타를 네트웍을 통해 접근하는 과부하를 피하기 위해 캐쉬를 사용한다. 이 논문은 클라이언트에서 지도를 수정하는 변경 트랜잭션들의 동시성 지원과 일관성 제어 문제를 다룬다. 지도를 수정하는 클라이언트 트랜잭션은 화면을 통해 대화식으로 진행되는 긴 트랜잭션이며, 변경 대상인 공간 객체는 공간 관련성에 의한 종속성을 가지는 특징이 있다. 또한, 캐쉬의 동적 중복에 대한 일관성을 제어하기 위해서는 변경 후 서버 뿐만 아니라 다른 클라이언트들로 전파가 필요하며, 이러한 변경 전파로 인한 통신 부하는 캐쉬의 이점을 잃지 않도록 최소화 되어야 한다.

이 논문은 ***CR 잠금***과 ***CX 잠금***을 이용한 캐쉬 영역 잠금법을 제시하여 공간 관련성에 의한 변경 종속성을 해결한다. 또한, ***CS 잠금*** 및 ***COD 잠금***을 제시하여 낙관적인 탐지기반 기법으로 일관성 제어를 지원함으로써 클라이언트들의 메시지 부하 최소화를 지원한다. 그리고, 이러한 확장된 잠금을 적용한 공간 관련성 기반 2PC 프로토콜을 통하여 긴 트랜잭션들 간의 정확한 지도 수정을 보장한다.

Abstract In a client-server spatial database, it is desirable to maintain the cached data in a client side to minimize the communication overhead across a network. This paper deals with the issues of concurrency and consistency of map updates in this environment. A client transaction to update map data is an interactive work and takes a long time to complete it. The map update in a client site may affect the other sites' updates because of dependencies between spatial data stored at different sites. The concurrent updates should be propagated to the other clients as well as the server to keep the consistency of map replicated in a client cache, and also the communication overhead of the update propagation should be minimized not to lose the benefit of caching.

The newly proposed ***cache region locking*** with ***CR lock*** and ***CX lock*** controls the update dependency due to spatial relationships. ***CS lock*** and ***COD lock*** are suggested to use optimistic detection-based approaches for guaranteeing the consistency of cached client data. The cooperative update protocol uses these extended locking primitives and ***Spatial Relationship-based 2PC (SR-based 2PC)***. This paper argues that the concurrent updates of cached client spatial data can be achieved by deciding on collaborative updates or independent updates based on spatial relationships.

1. 서 론

클라이언트-서버 공간 데이타베이스에서는 서버에 있는 지도 데이타가 네트웍을 통해 클라이언트로 전송되어야 하므로 사용자에 대한 응답 시간 지연을 일으킨다. 이의 해소를 위해서는 클라이언트에 캐쉬를 사용할 수 있다. 캐쉬를 사용하는 이유는 클라이언트와 서버 간의 공간 데이타 전송을 줄이고 서버에서 각 클라이언트의 데이타 요구 처리에 대한 과부하를 줄일 수 있는 이점을 가지기 때문이다[3].

† 비 회 원 : 위스콘신대학교 전산학과
ysshin@cs.wisc.edu
†† 종신회원 : 부산대학교 컴퓨터공학과 교수
bhhong@pusan.ac.kr
논문접수 : 2000년 11월 27일
심사완료 : 2001년 5월 8일

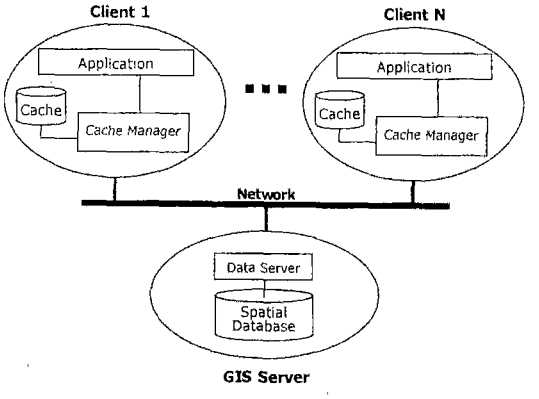


그림 1 캐쉬를 사용하는 클라이언트-서버 공간 데이타베이스 환경

그림 1은 클라이언트에 캐쉬를 사용하는 클라이언트-서버 공간 데이타베이스 응용의 한 예이다. 즉 하나의 공간 데이타베이스 서버에 네트웍을 통해 다수의 클라이언트가 연결되어 있다. 또한 각 클라이언트의 캐쉬에는 실행 시간 중 요구에 따라 서버 공간 데이타베이스의 일부를 중복하여 가지고 있다.

이러한 환경의 클라이언트에서 지도를 수정하는 트랜잭션은 다음과 같은 특징을 가진다.

- 지도 수정 유형은 대화식 작업을 필요로 하는 긴 트랜잭션이다[1].
- 공간 객체는 서로 위상 관계에 의한 공간 관련성을 갖는다. 즉, 서로 다른 객체이지만 기하의 공간 관련성에 따라 변경이 종속적일 수 있다[1].
- 각 클라이언트의 캐쉬에는 서버의 공간 데이타베이스가 동적으로 중복되어 있다[8].
- 동시성 및 일관성 제어 프로토콜의 메시지 부하 최소화가 우선적으로 고려되어야 한다.

이러한 특징들은 기존의 기법으로 클라이언트 지도 수정 트랜잭션 들의 동시성 지원과 일관성 제어를 어렵게 한다. 비관적 접근 방법(pessimistic approach)에 기반한 중복 제어 기법들은 다음과 같은 문제점들을 가진다. 첫째, 긴 트랜잭션에서의 잠금은 오랜 시간 대기를 초래한다. 둘째, 지도 수정 작업은 잠금 단위가 크다. 즉, 공간 데이타베이스에서는 공간 관련성 때문에 잠금의 대상이 되는 객체들만을 고려하여 수정할 수 없으므로 전체 데이타 집합 혹은 한 레이어 전체에 대하여 잠금을 사용하는 데, 이것은 긴 시간 대기 문제를 더욱 심각하게 한다. 셋째, 기존의 잠금 기법만으로는 공간 관련성에 의한 변경 종속성을 제어할 수 없다. 예를 들어, 반드시 접해 있어야 하는 필지와 도로 등과 같은 공간 객체들은 공간 관련성에 의한 종속성[1]을 지닌다. 서로 다른 객체라 하더라도 종속성이 있는 경우에는 여러 클라이언트에서 독립적으로 수정 후 서버에 변경할 때, 슬리버 폴리곤(sliver polygon)[9]이 발생하거나 경계 일치(edge matching)가 되지 않아 지도의 정확성과 질에 있어 일관성이 깨지는 문제가 발생할 수 있다.

독립적인 수정 후 합병하는 다중버전(multi-version) 제어 기법[10] 등과 같은 낙관적인 접근 방법에 기반한 중복 제어 기법들은 긴 트랜잭션에서 rollback문제를 가진다. 즉, 동일 객체에 대하여 서로 다르게 수정하였을 경우 합병 시에 변경 충돌이 발생하며 이때는 완료된 변경을 취소해야 한다. 또한 공간 관련성에 의한 변경 종속성으로 인해 발생하는 변경 충돌 시에도 마찬가지로 완료된 변경을 취소해야 한다. 그러나 긴 트랜잭션에서 완료된 변경의 취소는 그 비용이 크기 때문에 발생되어서는 안 된다.

이 논문에서는 캐쉬를 사용하는 클라이언트-서버 공간 데이타베이스 환경에서 지도 변경 트랜잭션의 동시성 지원 및 일관성 제어를 위한 새로운 방법을 제시한다. 이 논문에서 제시한 방법은 클라이언트 지도 수정 트랜잭션의 동시성 지원 및 일관성 제어를 위해 기존의 방법을 적용할 때의 문제점을 다음과 같이 해결한다.

첫째, 긴 트랜잭션에서 큰 잠금 단위로 인한 오랜 시간 대기 문제는 캐쉬 영역 잠금법을 도입하여 해결한다. 캐쉬 영역 잠금법에서는 잠금의 영역을 한정하는 CR 잠금을 제시함으로써 잠금의 단위을 줄이며 동시 접근이 가능케 한다.

둘째, 공간 관련성에 의한 변경 종속성을 제어하기 위해서는 캐쉬 영역 잠금법과 이를 기반으로 한 공간 관련성 기반 2PC(이하 SR 기반 2PC)[1]를 도입하여 해결 한다. 캐쉬 영역 잠금법에서는 공간 관련성에 의해 종속인 객체들을 함께 잠금하는 CX 잠금을 제시하여 공간 관련성을 제어한다. 그러나 공간 관련성은 객체의 수정에 따라 변하게 되며 이로 인한 지도 수정의 정확성은 CX 잠금 만으로는 해결 되지 않는 다. 객체의 수정에 의해 새로이 발생될 수 있는 공간 관련성에 의한 종속성은 협동 작업을 통해 사용자가 그 수정의 정확성을 판단 할 수 밖에 없다. 이 논문에서는 SR 기반 2PC를 통하여 협동 작업을 지원한다.

셋째, 긴 트랜잭션에서 rollback 문제는 긴 트랜잭션을 분할하고 그 단위의 수정만을 취소 할 수 있도록 제어함으로써 해결한다. 즉, 이전에 commit한 수정 결과를 긴 트랜잭션 전체가 완료되기 전에 rollback하여 수

정을 취소할 수 있는 프로토콜을 제안한다.

마지막으로, 프로토콜의 메시지 부하를 최소화 하기 위해서 낙관적 탐지기반의 캐쉬 일관성 유지 기법을 도입한다. 변경 종속성을 제어하기 위해 협동 작업을 할 경우에는 수정 내용이 협동 작업을 하는 클라이언트들을 대상으로 즉시 전파되어야만 한다. 그러나 변경 즉시 전파는 높은 메시지 과부하를 발생 시키므로 협동 작업을 하지 않는 클라이언트들의 캐쉬는 낙관적 탐지기반 방법으로 일관성을 유지함으로써 메시지 부하를 줄인다.

이 논문의 구성은 다음과 같다. 먼저 2장에서는 관련 연구를 기술한다. 다음으로 3장에서는 공간 데이타의 동적인 중복시의 특성에 대해 설명하고, 4장에서는 공간 데이타 수정의 문제점들을 설명한다. 그리고, 5장에서는 동시성 및 일관성 제어를 위하여 확장된 잠금 기법을 6장에서는 확장된 잠금 기법에 기반한 변경 전파 프로토콜을 설명한다. 그리고, 7장에서는 구현을 기술하며, 8장에서는 결론을 맺는다.

2. 관련 연구

클라이언트-서버 환경에서의 협동 트랜잭션을 위해서는 전통적으로 체크아웃 모델[7]이 사용된다. 체크아웃 모델은 체크인 시 변경 충돌이 발생할 수 있으며 변경 충돌 발생시에는 완료된 변경을 취소하여야 한다. 그러나 지도 수정 트랜잭션에서는 완료된 변경의 취소가 허용되어서는 안 된다. 왜냐하면 지도 수정 트랜잭션은 대화식 긴 트랜잭션이므로 취소 비용이 매우 크기 때문이다.

체크 아웃 모델을 지도 수정 트랜잭션에 적용할 때 변경 충돌이 발생하는 경우는 다음과 같다. 첫째, 동시 수정중인 클라이언트 변경 트랜잭션이 동일한 객체를 각각 수정함으로써 발생하는 변경 충돌이다. 둘째, 공간 데이타 간의 공간 관련성에 의한 종속성[1]으로 인해 발생하는 변경 충돌이다. 공간 데이타는 비공간 데이타와는 달리 공간 관련성을 가지고 있다. 그러므로 체크인 시 각각의 클라이언트에서 변경된 공간 객체가 서로 동일한 객체가 아닐지라도 공간 관련성에 의한 종속성으로 인해 변경 충돌이 발생할 수 있다. 따라서 지도 수정 트랜잭션에 체크아웃 모델을 그대로 적용하기는 어렵다.

잠금 기법을 사용하지 않고 체크아웃 모델을 기본으로 하여 긴 트랜잭션의 변경 충돌을 해결하기 위한 방법 중에는 두 단계 델타 합병 프로토콜[4]이 있다. 이 프로토콜에서는 잠금을 사용하지 않고 각 클라이언트가 변경 즉시 모든 사이트에 변경을 전파한다. 모든 사이트에 대한 변경 전파는 메시지 부하가 너무 높게 되어 사용자의 응답 시간 지연 해소를 위하여 사용하는 캐쉬의 이점을 살리기 어렵게 되는 문제가 있다. 또한 프로그램 객체를 수정 대상으로 함으로써 이 논문의 대상인 공간 객체가 가지는 공간 관련성에 의한 종속성을 고려치 못하고 있다.

전통적인 잠금 기반 방식을 긴 트랜잭션에 적용할 경우 긴 대기 시간을 초래할 수 있다. 또한 공간 데이타베이스에서는 공간 객체들의 공간 관련성으로 인해 긴 시간 대기 문제를 더욱 심각하게 한다. 그러므로 기존의 전통적인 잠금 기법 만으로는 지도 수정 트랜잭션의 동시성 지원 및 일관성 제어가 어렵다.

긴 대기 시간의 문제를 해결하기 위해 확장된 잠금을 사용하는 방법에 관한 연구로는[1]이 있다.[1]에서는 영역 잠금법 및 공간 관련성 바운드 WRITE 잠금법을 사용하여 분산 환경에서 협동 작업을 하는 긴 트랜잭션들의 동시성 및 일관성 제어를 한다.

이 논문에서는 동시성 및 일관성 제어를 위하여[1]을 기반으로 한다. 그러나 이 논문과의 차이점은 첫째, 캐쉬를 사용하는 클라이언트-서버 환경에서는[1]의 대상 환경인 분산 환경과는 달리 각 클라이언트가 자치적이 아니고 서버에 종속적이다. 클라이언트는 실행 시간에 추가적인 데이타를 위해서 서버에 요청하며 수정을 위한 잠금 제어도 서버를 통한다. 그러므로[1]에서와 같이 클라이언트에서의 데이타 변경 시 자신의 변경 내용을 다른 모든 사이트로 직접 전파할 때에는 프로토콜의 복잡도를 높이며 부가적인 메시지 부하를 발생시킨다. 왜냐하면 서버와의 필수적인 상호 작용 시 각 클라이언트 캐쉬의 유효성 탐지를 통하여 변경 내용을 전파할 수 있기 때문이다. 이 논문에서는 CS 잠금 및 COD 잠금을 새로이 제시하며 이를 이용한 낙관적 탐지기반 방법을 도입하여 불필요한 메시지 부하를 최소화 한다.

둘째, 캐쉬를 사용하는 클라이언트-서버 환경에서 모든 잠금은 서버의 데이타베이스를 대상으로 설정되며 모든 잠금의 설정은 서버가 제어한다. 그러나[1]에서의 영역 잠금법 및 공간 관련성 바운드 WRITE 잠금법에서 제시된 weak SIX 잠금 및 DSRX 잠금은 각 분산 사이트에 정적으로 중복된 데이타베이스를 대상으로 설정이 되는 분산 잠금이다. 그러므로 각 사이트는 자치적으로 잠금을 관리하며 자신의 잠금 설정 시 호환성 여부를 다른 분산된 사이트들을 통하여 결정하고 잠금 설정이 결정되면 다른 분산된 사이트들로 전파한다. 이 논문에서는 클라이언트-서버 환경에서 사용할 수 있는 CR 잠금 및 CX 잠금을 새로이 제시한다.

3. 데이타의 동적 중복

이 논문의 대상 환경에서는 클라이언트 캐쉬에 동적으로 데이타의 중복이 발생한다. 캐쉬에 중복 저장되는 데이타는 실행시간에 클라이언트에 의해 요청되는 데이타이다. 캐쉬는 크기에 제한이 있으므로 서버 데이타의 부분 집합만을 중복하고 있으며 캐쉬 교체에 의해 중복 내용이 변하게 된다. 이러한 중복을 **동적 중복**이라 정의한다. 동적 중복은 클라이언트 캐쉬 사이, 그리고 클라이언트 캐쉬와 서버 데이타베이스 사이에 발생한다.

데이타의 동적 중복 시에는 다음과 같은 사항이 고려되어야 한다. 우선, 캐쉬의 동적 중복에 대한 일관성 제어 기법이 필요하다. 이를 위하여 각 클라이언트의 동적 중복 정보와 유효성 정보가 관리되어야 한다. 동적 중복 정보는 각 클라이언트가 공간 객체를 서버로부터 전송받아서 캐싱하게 될 때와 캐쉬의 용량 제한으로 공간 객체가 교체될 때 변경된다. 동적 중복의 유효성 정보는 다른 클라이언트 수정 트랜잭션에 의해 공간 객체가 수정되게 될 때 변경된다. 따라서 동적 중복 정보와 유효성 정보를 통해 유효하지 못한 중복 데이타들의 일관성을 유지할 수 있다.

다음으로 캐쉬를 사용하는 이유가 사용자 응답 시간 지연 해소에 있으므로 캐쉬의 사용 목적을 달성하기 위해서는 변경 전파 프로토콜의 메시지 부하가 최소화 되어야 한다. 이를 위해 수정 트랜잭션에 직접 참여하여 협동작업을 하는 클라이언트들과 참여하지 않는 클라이언트들의 캐쉬 일관성 유지 기법을 달리해야 한다. 협동 작업을 통하여 수정에 참여하는 클라이언트들은 전역 공간 관련성을 고려한 수정을 위해 회피기반(avoidance-based) 방법으로 일관성을 제어하는 것이 필요하다. 이는 협동 작업을 하는 클라이언트들은 변경 내용을 확인 후 자신의 수정을 진행해 가야만 하기 때문에 변경 즉시 전파가 필요하기 때문이다. 그러나 변경 즉시 전파는 높은 메시지 부하를 수반하며 협동 작업에 직접 참여하지 않는 클라이언트들에게는 이러한 변경 즉시 전파가 필요치 않다. 그러므로 협동 작업에 직접 참여하지 않는 클라이언트들의 캐쉬 일관성 유지를 위한 프로토콜은 탐지 기반(detection-based) 방법을 사용하여 전체 프로토콜을 복잡하지 않게 설계할 수 있다. 또한 낙관적인(optimistic) 방법을 적용하여 탐지 시기를 최대한 늦춤으로써 메시지 부하를 최소화 할 수 있다[8].

그런데, 비공간 데이타와는 달리 공간 데이타의 동적 중복인 경우에는 각 클라이언트가 수정하고자 하는 수정 대상 객체 뿐 아니라, 각각의 수정 대상 객체와 공간 관련성에 의해 종속적인 객체들도 반드시 함께 클라이언트의 캐쉬에 중복되어져야만 하는 특징이 있다. 왜냐하면 지도 수정 트랜잭션에서는 사용자가 직접 화면을 통해서 수정 대상 객체를 수정하므로, 공간 관련성을 고려한 정확한 수정을 위해서는 수정 대상 객체 뿐만 아니라 변경 종속인 객체들도 함께 화면에 출력되어져야만 하기 때문이다.

4. 전역 공간 관련성에 의한 데이타 종속성

모든 두 공간 객체는 공간 관련성을 가진다. Egenhofer [6]는 공간 관련성을 Disjoint, Meets, Equals, Inside1, Inside2, Covers1, Covers2, Overlaps의 8가지로 분류하였다. 이러한 공간 관련성은 기본적으로 하나의 데이타 집합 내에서 정의된다. 이 논문에서는 한 사이트의 데이타 집합 내에서 존재하는 공간 관련성을 **지역 공간 관련성(local spatial relationship)**이라 정의한다. 캐쉬를 사용하는 클라이언트-서버 환경에서는 분산 환경에서와 같이 서버의 데이타가 여러 클라이언트 캐쉬에 중복되어 있기 때문에 서로 다른 클라이언트에 존재하는 두 객체라 하더라도 위치 투명성 (location transparency)의 관점에서 서버의 데이타 집합 내에 존재하던 공간 관련성이 여전히 존재한다. 이 논문에서는 서로 다른 클라이언트에 저장된 두 공간 객체간의 이진 위상 관계(binary topological relation)를 **전역 공간 관련성 (global spatial relationship)**이라 정의한다.

그림 2에서는 'Inside' 전역 공간 관련성을 예로 보이고 있다. 클라이언트1은 거주지(residential_district)만을 클라이언트2는 가옥(house)만을 자신의 캐쉬에 가지고 있지만 서버 데이타베이스에서의 거주지 객체와 가옥 객체들간의 공간 관련성은 그대로 유지되어 전역 공간 관련성을 가진다.

그림 3은 공간 객체 SO_A와 SO_B의 수정을 서로 다른 트랜잭션 T_A와 T_B가 각각 수정하는 3가지 시나리오를

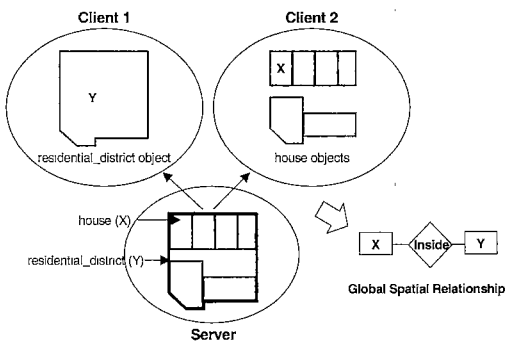


그림 2 Inside 전역 공간 관련성의 예

보인다. 여기서 T_A는 SO_A에 잠금을 걸고, T_B는 SO_B에 잠금을 걸고 수정 작업을 수행한다고 가정한다. 그리고 그림 3에서 SO_A와 SO_B는 경계 부분이 접해있는 'Meet' 공간 관련성을 가진다. 이때 그림 3 (a)와 (b)에서 T_A와 T_B를 순차적으로 수행한다면 공간 객체 SO_A와 SO_B간의 공간 관련성은 그대로 지켜지며 그 결과는 정확함을 알 수 있다.

그런데, SO_A와 SO_B는 서로 다른 객체이기 때문에 T_A와 T_B 간에 잠금 충돌은 존재하지 않는다. 따라서, 두 트랜잭션이 각각 SO_A와 SO_B에 잠금을 걸고 병렬 수정을 한다면 그림 3 (c)에서 처럼 두 객체의 경계 부분이 접하지 않는 문제가 발생될 수 있다. 이러한 문제가 발생하는 이유는 SO_A와 SO_B가 서로 수정 결과를 모르는 상태에서 개별적으로 수정되었으므로, 변경 후에도 서로의 경계선이 접해야 한다는 제약 조건이 위반될 수 있기 때문이다. 즉, SO_A와 SO_B는 서로 경계가 접해있어야 하는 공간 관련성을 지니며, 이 관련성이 데이타 종속성으로 작용했기 때문이다. 따라서, 전통적인 객체 단위의 잠금 기법을 공간 데이타에 그대로 적용할 때 문제가 발생한다. 그러므로 기존의 공간 데이타베이스들은 지도 수정 시 전체 데이타 집합 혹은 한 레이어 전체를 잠금하고 있다.

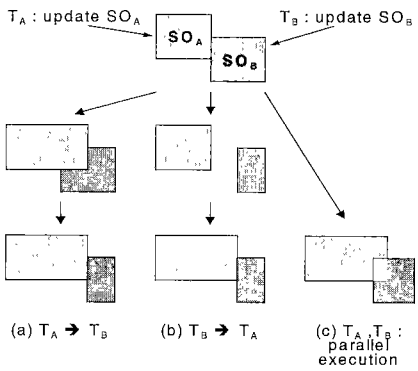


그림 3 공간 관련성을 가진 객체들의 수정

기하를 가지는 공간 객체는 인접한 객체와의 공간 관련성을 지니므로 잠금 대상이 상이할 경우에도 병렬 수정될 수 없다. 공간 관련성에 의한 병렬 수정 제약 조건을 공간 관련성에 의한 종속성[1]이라 한다. 공간 관련성에 의한 종속성은 'Disjoint' 공간 관련성을 제외한 모든 공간 관련성을 갖는 두 객체간에 존재한다. 전역 공간 관련성을 가진 공간 객체들 간에도 이러한 병렬 수정 제약 조건은 여전히 존재하며 이를 **전역 공간 관련성에 의한 종속성**이라 하고 다음과 같이 정의한다.

정의 1 서로 다른 두 클라이언트 C_i, C_j에서 수행되는 수정 트랜잭션을 각각 T_i, T_j라 하고, SO_i, SO_j를 각각 T_i와 T_j가 수정하는 공간 객체라고 할 때, 두 공간 객체 SO_i와 SO_j간에 Disjoint를 제외한 전역 공간 관련성이 존재할 경우 두 객체는 전역 공간 관련성에 의해 종속적이며 이러한 전역 공간 관련성에 의한 병렬 수정 제약 조건을 **전역 공간 관련성에 의한 종속성**이라 한다.

5. 확장된 잠금 기법

이 장에서는 클라이언트 변경 트랜잭션들의 동시성 및 일관성 제어를 위해 확장된 잠금을 제시한다. 제시된 잠금들은 서버에서 설정되며 관리된다.

5.1 CS 잠금과 COD 잠금

이 절에서는 탐지기반 방법에 의하여 각 클라이언트 캐쉬의 일관성을 유지하기 위한 잠금 모드를 설명한다. 탐지기반 방법을 위해서는 서버에 클라이언트들의 동적 중복 정보 및 유효성 정보를 유지해야 한다. 이는 각 클라이언트가 서버에서 캐쉬의 유효성을 탐지함으로써 일관성을 유지하기 위함이다.

캐쉬의 동적 중복 정보를 서버에서 유지하기 위해 CS 잠금을 도입하며 다음과 같이 정의한다.

정의 2 클라이언트가 캐쉬을 위해 서버로부터 공간 객체들을 요구할 때 공간 객체를 대상으로 설정되고 클라이언트의 캐쉬에서 교체될 때 해제되는 잠금 모드를 CS(Cache-Shared) 잠금이라 정의한다.

CS 잠금을 통하여 서버는 각 클라이언트가 어느 공간 객체들을 캐쉬에 중복 저장하고 있는 지에 대한 동적 중복 정보를 유지할 수 있다. CS 잠금이 Shared 잠금과 다른 점은 동시성 지원을 위해 다른 클라이언트들의 수정을 위한 잠금 요청을 허용한다는 것이다. 따라서 하나의 클라이언트가 데이타 일부를 자신의 캐쉬에 중복 저장한 경우 (CS 잠금)에도 다른 클라이언트들이 이 데이타를 수정할 수 있다.

CS 잠금된 공간 객체에 대해 다른 클라이언트에서의 수정을 허용하였기 때문에, 각 클라이언트에 중복 저장된 데이타의 일관성 유지 문제가 발생한다. 각 클라이언트 캐쉬의 일관성을 유지하기 위해서는 현재 각 클라이언트의 캐쉬에 있는 동적 중복된 데이타의 유효성을 탐지할 수 있어야 한다. 이를 위해 COD(Cache-Out-of-Date)잠금을 도입하며 다음과 같이 정의한다.

정의 3 CS 잠금 후 클라이언트 캐쉬에 동적 중복

저장된 객체가 이후 다른 클라이언트 트랜잭션의 수정에 의해 무효화 되었을 때, 이 객체의 CS 잠금 모드는 무효성 정보를 가진 잠금 모드로 변환되어 져야 한다. 이때 CS 잠금이 강등(*demote*)되어 지는 잠금 모드를 COD(Cache-Out-of-Date) 잠금이라 정의한다.

서버는 COD(Cache-Out-of-Date) 잠금 모드를 통하여 각 클라이언트 캐쉬의 유효성 정보를 유지 할 수 있다. 그러므로 각 클라이언트는 이 정보를 통하여 클라이언트 캐쉬의 유효성을 탐지할 수 있다.

5.2 동시성 제어

이 장에서는 동시성 제어를 위하여 클라이언트의 화면에 출력된 지도에서 특정 영역 단위로 지도 수정 트랜잭션이 데이타 수정 범위를 정하고, 이 영역 내에 포함된 객체들을 대상으로 수정하기 위한 캐쉬 영역 잠금법(Cache-Region Locking)을 설명한다.

5.2.1 캐쉬 영역 잠금법

각 클라이언트들의 캐쉬에 동적 중복된 공간 데이타에는 전역 공간 관련성에 의한 종속성이 존재하기 때문에 잠금한 객체만의 잠금 호환성을 따져 동시성을 제어할 수 없다. 이 논문에서는 전역 공간 관련성에 의한 종속성을 기반으로 동시성 제어를 하기 위해서 기존의 WRITE 잠금을 확장한다. 확장된 잠금은 WRITE 잠금을 걸고자 하는 객체 뿐만 아니라, 그 객체와 전역 공간 관련성에 의하여 종속적인 관계에 있는 다른 객체들도 함께 잠금을 설정하는 잠금으로서 다음과 같이 정의한다.

정의 4 한 공간 객체에 대하여 수정 하고자 할 때, 그 객체 또는 그 객체와 전역 공간 관련성에 의한 종속 관계에 있는 객체들이 수정 중이라면 대기해야 하는 쓰기 잠금 모드를 CX(Cache-eXclusive) 잠금이라 정의한다.

기존의 WRITE 잠금은 전역 공간 관련성을 고려하지 않고 있기 때문에 공간 데이타 수정은 전체 데이타 집합 또는 한 레이어 전체에 대한 잠금을 한다. 그러므로 긴 트랜잭션에서 여러 클라이언트에 의한 동시성을 지원하기 어렵다.

이 논문에서 제시하는 캐쉬 영역 잠금법(Cache-Region Locking)은 사용자가 수정하고자 하는 특정 영역 내의 공간 객체 집합에 대하여 READ 잠금을 미리 요청하고 그 공간 객체들에 대해서만 CX 잠금 요청을 허용하는 잠금 방법이다. 이 방법에서는 각 사용자가 수정 트랜잭션 내에서 수정할 영역을 지정한 뒤 수정을 하기 때문에 수정 트랜잭션을 수행하는 클라이언트들 간의 동시성을 높여준다. 즉, 수정하려는 영역이 겹치지 않을 경우에는 전역 공간 관련성에 의한 종속성을 지닌 공간 객체들을 동시 수정할 가능성이 없으므로 병렬 수정을 보장해 줄 수 있다. 또한 수정하려는 영역이 겹칠 경우, 협동 작업을 통하여 동시성을 보장해 준다. 이 논문에서는 협동 작업을 통해 동시성을 보장하기 위하여 CR (Cache-Region) 잠금을 제안하며 다음과 같이 정의한다.

정의 5 한 map의 전체 데이타 집합(Data set)을 D, 한 map의 전체 영역을 R, 긴 트랜잭션 T_i를 실행하는 사용자가 정의한 R의 부분 영역(sub region)을 R_i, 그리고, R_i영역에 'Inside'되는 (완전히 포함되는) D의 부분 객체 집합 (sub object set)을 D_{Ri}라 하자. 이때, D_{Ri}에 대하여 READ 잠금을 설정하고 그 중 일부 객체들에 대해 CX 잠금의 요청을 의도하는 잠금을 CR 잠금(Cache-Region Lock)이라 정의 한다. 또한 이때, CR 잠금의 영역 R_i를 CRG_i (CR locked reGion)라 한다.

CR 잠금은 다중 단위 (Multiple Granularity) 잠금법 [11]의 SIX 잠금과 유사하지만 잠금한 객체들에 대하여 다른 트랜잭션의 CS 잠금을 허용한다는 점에서 차이가 있다. CS 잠금을 허용하는 이유는 한 클라이언트의 지도 수정을 위해 SIX 잠금을 걸 경우 다른 클라이언트들이 오랫동안 공간 데이타를 참조할 수 없기 때문이다. CS 잠금을 허용함으로써 발생할 수 있는 dirty read 문제는 캐쉬의 일관성 유지 기법으로 해결될 수 있다. CR 잠금은 동시성을 높이기 위해 다른 트랜잭션의 CR 잠금과 CX 잠금 요청도 허용한다. 다른 트랜잭션의 CR 잠금과 CX 잠금을 허용함에 의해 발생할 수 있는 dirty read와 lost update 문제는 협동 작업 시 변경 전파 프로토콜에 의해 해결될 수 있다.

그림 4는 CR 잠금의 예를 보인 그림이다. 클라이언트에서는 서버 지도 데이타베이스의 부분 집합만을 캐쉬에 중복 저장한다. 그림 4에서 클라이언트 캐쉬에는 서버의 전체 데이타 집합 중 필지(Property) 레이어의 객체들만을 중복 저장하고 있다. 사용자가 공간 객체의 수

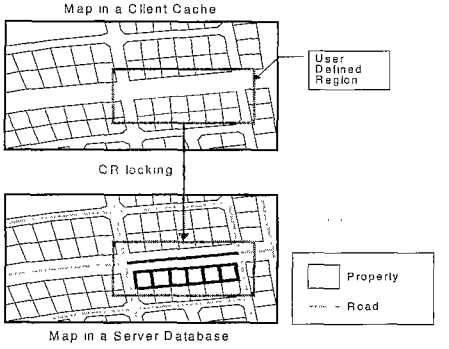


그림 4 CR 잠금의 예

정을 위해서 그림 4와 같이 사용자 정의 영역을 설정하고 서버에 CR 잠금을 설정하면 서버 데이타베이스에서 CR 잠금 영역 내에 완전히 포함되는 모든 공간 객체들이 CR 잠금의 대상 객체가 된다. 그러므로 클라이언트에서 사용자가 정의한 CR 잠금 영역으로 서버에 CR 잠금을 요청하면 서버에서는 클라이언트 캐쉬에 중복 저장되어 있지 않은 도로(Road) 객체도 CR 잠금의 대상으로 잠금한다. 단 이때 수정하려는 공간 객체와 전역 공간 관련성에 의한 종속 관계에 있는 공간 객체들도 수정 시 함께 고려되어야 한다. 왜냐하면 공간 객체가 가지는 전역 공간 관련성은 기하의 변경에 따라서 변하기 때문에 기하의 변경 후 새로운 종속 관계가 발생할 수 있기 때문이다[1]. 따라서 CR 잠금이 설정된 공간 객체들 중 클라이언트의 캐쉬에 중복되어 있지 않는 객체들은 클라이언트로 전송되어 져야 한다.

이 논문에서는 CR 잠금과 관련하여 다음 사항들을 전제로 가정한다. 첫째, 각 클라이언트의 지도 수정 트랜잭션은 종료 시까지 CR 잠금이 설정된 객체 집합 내의 객체들에 대해서만 CX 잠금을 설정할 수 있으며, CX 잠금을 설정한 객체의 기하는 CR 잠금 영역 내에서만 수정할 수 있다는 것을 전제로 한다. 둘째, 일반적으로 공간 데이타베이스 응용에서는 지도를 확대 및 축소하여 다양한 스케일로 화면 상에 출력할 수 있다. 그러나 이 논문에서는 지도 수정 작업이 현재 출력되는 화면 내의 모든 공간 객체들이 캐쉬에 저장되어 질 수 있는 스케일에서만 허락되어 진다고 가정한다. 셋째, CR 잠금 영역 내의 모든 공간 객체들은 트랜잭션이 끝날 때까지 클라이언트의 캐쉬에서 교체되지 않는 다고 가정한다.

정의 4와 정의 5를 이용한 캐쉬 영역 잠금법(Cache-Region Locking)의 정의는 다음과 같다.

정의 6 사용자가 공간 객체를 수정하기 위해서는 사용자가 지정하는 특정 영역 내의 공간 객체 집합에 대하여 CR 잠금을 설정해야만, 그 중 일부 객체들에 대한 CX 잠금 요청을 허용하는 잠금법을 캐쉬 영역 잠금법(Cache-Region Locking)이라 정의한다.

5.2.2 캐쉬 영역 잠금법에 의한 동시성 제어 기법

이 장에서는 CR 잠금 및 CX 잠금에 의한 동시성 제어 기법에 대해서 자세히 설명한다. 동시성 제어 기법은 서로 다른 수정 트랜잭션에 의한 수정 영역이 겹치지 않는 경우와 겹치는 경우에 따라서 병렬 수정과 협동 작업으로 나눌 수 있다. 먼저 동시성 제어 기법과 관련된 용어를 설명한다. 트랜잭션 T_A의 CR 잠금과 트랜잭션 T_B의 CR 잠금이 서버에 설정되었을 때 각각의 CR 잠금 영역 CRG_A와 CRG_B는 공간 객체의 공간 관련성과 같이 8가지 관계에 있을 수 있다. 이중 'Disjoint'를 제외한 경우 상호 중첩이 존재하는 데 이것을 NDJ_{AB} (NonDisJoint area of T_A and T_B)라 하고 다음과 같이 정의한다.

정의 7 트랜잭션 T_A의 CRG_A와 트랜잭션 T_B의 CRG_B는 'Disjoint'를 제외한 경우 상호 중첩을 가지는 데 이러한 상호 중첩 영역을 NDJ_{AB}(NonDisJoint area of T_A and T_B)라 하고, 각 공간 관계에 따라 다음과 같이 정의한다.

① 'Overlaps' 될 경우
$NDJ_{AB} = CRG_A$

② CRG_B 'Inside'와 'Cover' 될 경우
$NDJ_{AB} = CRG_A$

③ CRG_B 'Meets'과 'Equal' 될 경우
$NDJ_{AB} = CRG_A \quad CRG_B$

이 논문에서 트랜잭션 T_A는 D_{Ra}객체들 만을 수정 대상으로 하며, 그 중에서 일부 객체 들을 하나의 수정 단위로 CX 잠금을 걸고 변경 작업을 수행한다고 가정한다. 이때, D_{Ra}중 트랜잭션 T_A에 의해 현재 CX 잠금을 설정하고 수정 중인 객체들을 UDO_A(UpDating Objects of T_A)라 한다. ($UDO_A \quad D_{Ra}$)

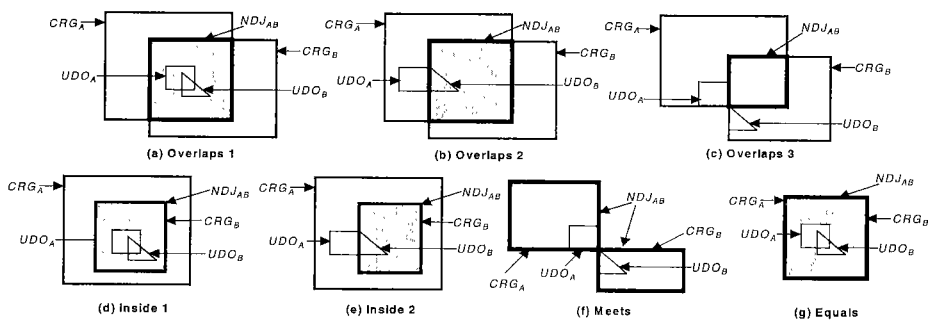
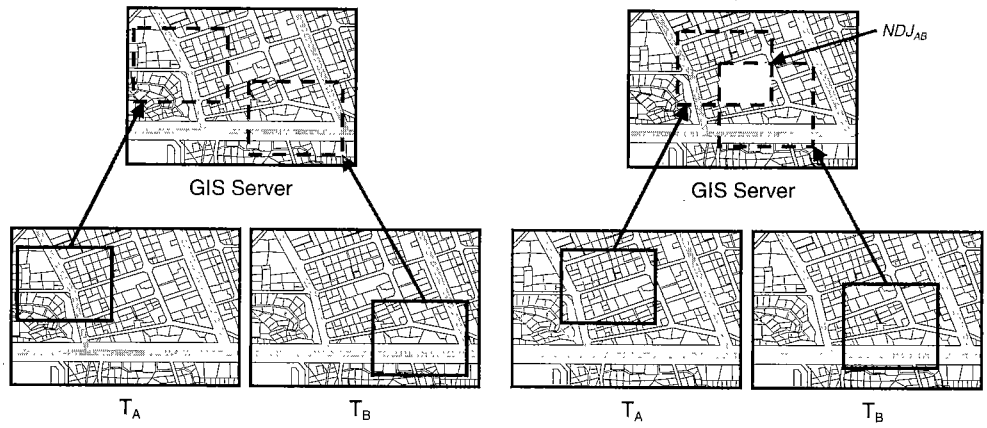


그림 5 NDJAB의 존재와 변경 충돌 가능성



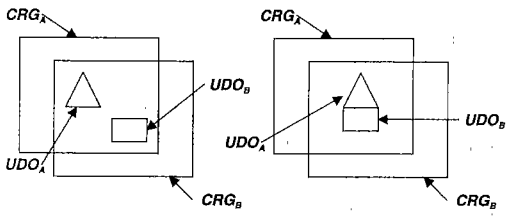
(a) Parallel execution (b) Cooperative update

그림 6 CR 잠금에 의한 동시성 제어

두 CR 잠금 사이에 NDJ_{AB}가 존재할 경우, 두 트랜잭션은 동시 수정 시 변경 충돌이 발생할 수 있는 객체들을 포함하고 있다. 이것을 그림 5에서 보이고 있다. 그림 5는 두 CR 잠금이 Non Disjoint할 경우에 전역 공간 관련성을 가진 객체들이 존재한다는 것을 보이고 있다. 여기서, 'Covers'는 그림 5 (d), (e)와 경우가 같으므로 생략하였다.

그림 6은 CR 잠금의 특성에 의한 동시성 제어 기법을 설명하고 있다. 먼저 그림 6의 (a)에서는 NDJ_{AB}가 존재하지 않는 경우를 나타내고 있다. 이 때에는 두 트랜잭션이 전역 공간 관련성에 의한 종속성을 지닌 공간 객체들을 동시 수정할 가능성이 없으므로 서로 병렬 수행될 수 있다. 그러나 그림 6의 (b)에서는 NDJ_{AB}가 존재하므로 전역 공간 관련성에 의한 종속성으로 인해 충돌이 발생할 수 있기 때문에 병렬 수행을 허용할 수 없다. 이 때에는 두 트랜잭션이 협동 작업으로 수정을 진행하여야 한다. 지도 수정 트랜잭션은 사용자와의 상호작용에 의해 진행되며 수정 결과의 공간 관련성에 의한 종속성이 고려된 정확성을 사용자의 판단에 근거한다. 그러므로 CR 잠금을 통하여 공간 관련성에 의해 종속적이지 않은 객체들의 수정은 병렬 수행 될 수 있도록 보장 된다.

그런데, NDJ가 존재하더라도 두 클라이언트 수정 트랜잭션에서의 수정 공간 객체가 전역 공간 관련성에 의한 종속성이 없다면 동시 수정을 허용하더라도 변경 충돌이 발생하지 않을 수 있다. 그림 7 (a)에서 UDO_A와 UDO_B에 전역 공간 관련성에 의한 종속성이 존재하지 않으므로 두 트랜잭션 T_A와 T_B는 동시에 실행될 수 있다. 그러나, 그림 7 (b)에서 UDO_A와 UDO_B에 전역 공간 관련성에 의한 종속성이 존재하므로 두 트랜잭션 T_A와 T_B는 동시에 실행될 수 없다.



(a) T_A and T_B : parallel (b) T_A or T_B : wait execution

그림 7 전역 공간 관련성을 고려한 동시성 제어

이러한 동시 실행 여부는 CX 잠금을 통하여 결정한다. 그러므로 CX 잠금은 다음과 같이 지도 수정 작업 시 병렬성을 높일 수 있다. 첫째, 두 트랜잭션 T_A와 T_B의 CR 잠금이 NDJ_{AB}를 가지더라도 현재 수정중인 객체들이 상호 전역 공간 관련성에 의한 종속성이 없을 경우 동시 수정을 허용하여 병렬성을 높인다. 둘째, 전역 공간 관련성에 의한 종속성이 있을 경우에는 하나의 수정 단위가 변경되는 동안만 대기하게 하여 병렬성을 높일 수 있다.

5.3 잠금 모드간의 호환성

지금까지 제시된 잠금 들의 호환성은 표 1과 같다.

표 1 잠금 호환성 표

	CS	COD	CR	CX
CS	yes	yes	yes	yes
COD	yes	yes	yes	N/A
CR	yes	yes	yes	yes
CX	yes	yes	yes	no

클라이언트 트랜잭션에서의 공간 데이타 수정을 위해 표 1에 적용한 호환성 규칙은 다음과 같다. 첫째, CX 잠금 후 수정 중인 객체를 CS 잠금 또는 CR 잠금을 걸고 읽는 것을 허용하는 것은 데이타의 부정확성 문제를 초래하지만, 이 논문은 동시성 지원을 위해 이를 허용한다. 즉, 지도 수정이 긴 트랜잭션이므로 공간 데이타가 변경되고 있는 동안의 데이타베이스 일관성 위반은 허용한다. 그렇지만, 지도의 수정은 CX 잠금 권한을 얻어야 하며 CX 잠금은 서로 호환되지 않으므로 lost update는 발생하지 않는다. 그리고, CS 잠금에 의한 dirty read는 COD 잠금으로 탐지되며, CR 잠금에 의한 dirty read는 협동 작업에 의한 변경 전파 프로토콜로 변경 내용이 통보된다. CR 잠금과 CX 잠금이 설정되어 있더라도 같은 이유로 다른 클라이언트에서 지도를 캐싱할 수 있도록 CS 잠금을 허용한다.

둘째, 한 클라이언트에서 지도 수정을 위해서는 사용자가 지정하는 수정 영역 내의 객체 집합에 대하여 CR 잠금을 설정하고 실제 수정할 객체들에 대하여 CX 잠금을 필요에 따라 설정한다. 이때 다른 클라이언트에서도 동일한 혹은 수정 영역이 겹치는 영역 내의 객체 집합에 대하여 CR 잠금을 설정하고 협동 작업을 수행할 수 있도록 허용한다.

6. 변경 전파 프로토콜

이 장에서는 새로 제시한 잠금에 기반하여 동시성 및 일관성을 제어하는 변경 전파 프로토콜을 제시한다. 기술의 편의를 위해 다음과 같이 용어를 정의한다.

- *Coordinator* : 수정 후 *mid-commit*을 발생 시키는 클라이언트.
- *Participant* : 협동 작업을 하는 클라이언트. 즉, CR 잠금을 설정하였으며 그 CR 잠금의 영역이 *Coordinator*와 Non Disjoint 관계인 클라이언트.
- *Agent* : *Coordinator*가 발생 시키는 트랜잭션 연산을 대행하여 다른 *Participant*에게 전파시키는 서버. 이 논문에서는 공간 데이타베이스 서버가 역할을 담당한다.

6.1 트랜잭션 연산

이 논문에서는 클라이언트-서버 환경의 트랜잭션 통신 프로토콜을 클라이언트-서버 트랜잭션 연산을 도입하여 기술한다. 클라이언트-서버 트랜잭션 연산은 전통적 트랜잭션 연산을 확장하고 작업 결과를 통보하는 연산 등을 추가한 것이다. 통신 프로토콜을 위한 트랜잭션 연산은 사용자의 결정에 의해 실행되는 연산(이탤릭체로 표기)과 시스템에 의해 자동으로 실행되는 연산으로 나뉘며 일부 트랜잭션 연산은 하위 연산을 가지고 있다.

사용자에 의한 트랜잭션 연산은 다음과 같다.

- *request-CR-lock* : 수정 영역을 지정하고 서버에게 수정 영역 내의 객체 집합에 대한 CR 잠금의 설정을 요청한다.
- *request-CX-lock* : CR 잠금한 객체 집합 내의 수정할 객체들에 대한 CX잠금 설정을 서버에게 요청한다.
  - reply-CX-ok : *request-CX-lock*에 대하여 협동 작업을 하는 클라이언트 트랜잭션들에서 잠금 충돌이 없음을 알린다.
  - reply-CX-conflict : *request-CX-lock*에 대하여 협동 작업을 하는 클라이언트 트랜잭션들에서 잠금 충돌이 존재함을 알린다.
  - wake-up : CX 잠금 설정 대기를 하고 있는 클라이언트 트랜잭션에게 잠금이 설정되었으므로 대기를 해제할 것을 알린다.
- *mid-commit* : 서버에게 CX 잠금 후 변경한 객체들의 수정 결과(DELTA)를 전달하고 SR기반 2PC를 시작할 것을 요청한다.
  - *reply-mid-accept* : *mid-commit*에 대한 승인을 서버에게 알린다.
  - *reply-mid-reject* : *mid-commit*에 대한 거부를 서버에게 알린다.
  - send-global-mid-commit : *Participant*에 *mid-commit*을 완료한다.
  - send-global-mid-abort : *Participant*에 *mid-commit*을 취소한다.
- *release-CR-lock* : CR 잠금의 해제를 요청한다.
  - reply-release-CR-lock-ok : CR 잠금의 해제 요청에 대한 허락을 알린다.
  - reply-release-CR-lock-wait : CR 잠금의 해제를 대기할 것을 알린다.
  - permit-release-CR-lock : CR 잠금 해제를 대기하는 클라이언트에게 CR 잠금 해제 요청에 대한 허락을 알린다.

시스템에 의해 자동 실행되는 트랜잭션 연산은 다음과 같다.

- request-CS-lock : 클라이언트측 캐쉬 매니져가 서버에 공간 데이타를 요청한다. 즉 CS 잠금의 설정을 요청한다.
- release-CS-lock : 클라이언트 캐쉬에서 교체된 공간 객체들의 CS 잠금 해제를 서버에 요청한다.
- release-CX-lock : 서버에게 CX 잠금의 해제와 대기중인 클라이언트들의 변경 트랜잭션들 중 하나를 깨울 것을 요청한다.
- detect-validity : 클라이언트의 캐쉬에 유지 중인 객체들에 대한 유효성을 탐지할 것을 요청한다.
- svr-rcv-any-operation : any-operation 연산을 수신한 서버, 즉 *Agent*에서의 해당 응답 연산
- clnt-rcv-any-operation : any-operation 연산을 수신한 클라이언트 트랜잭션에서의 해당 응답 연산

6.2 mid-commit

점진적 변경 전파 기법을 사용하기 위하여 이 논문에서는 *mid-commit*[1] 개념을 도입한다. *mid-commit*은 점진적 변경 전파 기법으로 캐쉬 트랜잭션들의 동시성을 높이고, 전역 공간 관련성으로 인해 발생하는 지도 수정 결과의 오류를 없애기 위해 협동 작업을 수행한다. 클라이언트 수정 트랜잭션들의 협동 작업은 기존의 2PC를 확장한 SR기반 2PC[1]로 지원한다.

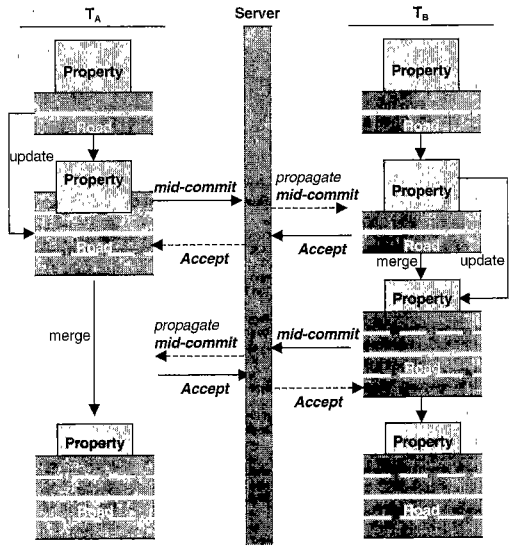


그림 8 *mid-commit*의 개념

그림 8은 *mid-commit*의 개념을 설명한 그림이다. 그림 8에서 T_A의 도로(Road) 객체가 수정되었을 때 수정 전과의 차이를 *DELTA*라 하며, *DELTA*는 CX 잠금이 설정된 수정 객체들을 log에 기록한 후 *mid-commit*때 수정된 결과와의 차이를 구한 것이다. *DELTA*를 서버로부터 전파 받은 T_B는 변경 내용을 확인하고 자신의 데이타 집합에 *DELTA*를 병합한다.

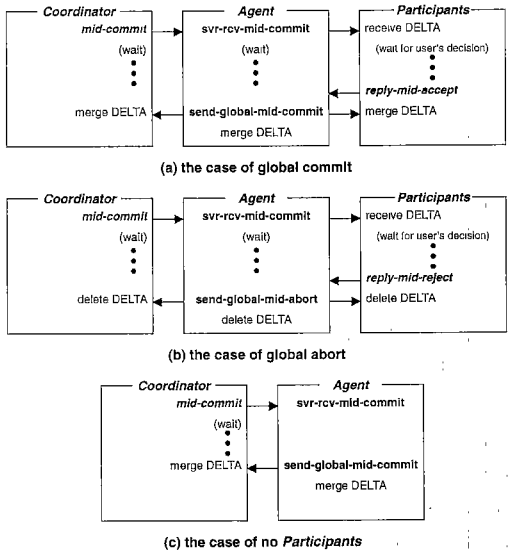


그림 9 공간 관련성 기반 2PC(SR 기반 2PC)

그림 9는 SR 기반 2PC의 개념을 설명한 그림이다. 그림 9 (a)에서는 *Participant*가 모두 *mid-commit*을 승인(*mid-accept*) 하였을 때이고, 이때 global-mid-commit이 된다. 그러나 그림 9 (b)에서 처럼 *Participant*중 승인하지 않는(*mid-reject*) 클라이언트가 하나라도 있을 때에는 *mid-commit*이 허락되어 질 수 없다. 즉 global-mid-abort가 된다. 그림 9 (c)에서는 *Participant* 없이 *mid-commit*을 발생시킨 클라이언트 단독으로 수정하는 경우이다. 이때에는 협동 작업이 필요치 않으므로 서버가 *mid-commit*을 전파하지 않으며 바로 global-mid-commit을 *Coordinator*로 보내어 commit하게 한다.

6.3 지도 수정 트랜잭션 모델

그림 10은 CR 잠금과 CX 잠금을 이용한 지도 수정 트랜잭션 모델이다. 이것은 트랜잭션 연산들을 이용하며 그 내용은 다음과 같다.

```
(Map Update Transaction)
Begin T_A
request-CR-lock(D_RA)
   request-CX-lock(UDO_A1)
      (Update on UDO_A1)
         :
      mid-commit(UDO_A1)
   release-CX-lock(UDO_A1)
                                  } an update cycle
   request-CX-lock(UDO_A2)
      (Update on UDO_A2)
         :
release-CR-lock(D_RA)
commit T_A
End T_A
```

그림 10 지도 수정 트랜잭션 모델

지도 수정 트랜잭션은 CR 잠금의 영역을 설정하고 서버로 잠금 설정을 요청(*request-CR-lock*)한다. 그리고 CR 잠금 된 객체 집합 중에서 객체들에 대하여 CX 잠금을 서버에 요청(*request-CX-lock*)한다. 잠금 권한을 획득한 후 그 객체들에 대한 수정 작업을 수행하며 수정 완료 후 변경한 결과를 서버로 전달해서 서버로 하여금 변경한 결과를 협동 작업을 하는 클라이언트들로 전파(*mid-commit*)하도록 한다. 이렇게 한 단위의 수정 객체들에 대한 변경이 끝나고 *mid-commit*이 되면 CX 잠금은 해제되며, 다음 수정 객체 들을 설정하고 동일한 과정으로 처리한다. 마지막으로 모든 수정이 종료되면 CR 잠금을 해제하고 트랜잭션을 완료하게 된다. 이 논문에서는 deadlock을 방지하기 위하여 한 수정 단위 내에 하나 이상의 CX 잠금 설정을 허용하지 않는다.

그림 10에서 하나의 수정 단위(an update cycle)는 변경 후 수정 내용을 전파하는 단위로서 CX 잠금을 설정하는 단위이다. 즉, 이 논문에서 제시하는 지도 수정 트랜잭션 모델은 긴 트랜잭션을 수정 단위로 분할하고 그 단위로 점진적 완료를 허용한다. 점진적 완료를 도입하는 이유는 첫째, 동시성을 높이기 위함이다. 수정 단위가 분할됨으로써 잠금의 단위가 줄어들고, 잠금 충돌의 가능성이 감소하기 때문이다. 둘째, 변경 충돌이 발생할 경우 rollback 단위를 하나의 수정 단위로 제한할 수 있어 그 비용을 줄일 수 있기 때문이다.

6.4 트랜잭션 연산별 알고리즘

이 절에서는 주요 트랜잭션 연산의 알고리즘과 이를 이용한 변경 전파 프로토콜을 설명한다.

6.4.1 CR 잠금의 설정

트랜잭션 T_A의 CR 잠금 설정 알고리즘은 그림 11과 같다. *request-CR-lock* 연산은 CR 잠금의 영역을 정의하여 *Agent*로 보낸 뒤 자신의 log에 기록한다. *Agent*에서 수행되는 svr-rcv-request-CR-lock 연산은 T_A의 CR 잠금 영역 CRG_A와 Non Disjoint한 영역을 가지는 CR 잠금을 가진 클라이언트 트랜잭션들을 찾아서 *Participant*로 저장하고 CR 잠금을 설정한다. 또한 CR 잠금의 설정 요청 시에는 T_A의 캐쉬 유효성을 COD 잠금을 통해 검사하여 T_A에게 결과를 통보한다.

```
function request-CR-lock
var
CRG_A : the region of CR lock of T_A
CRL_A : the CR lock of T_A
TR_ID_A : the transaction id of T_A
CODL_A : the COD lock of T_A
begin
  define CRG_A

  send request-CR-lock msg
      with TR_ID_A and CRG_A to Agent

  for (timeout)
    wait for response

  If (there is CODL_A) then
    update own cache with CODL_A

  write CRL_A to log
end
```

```
function svr-rcv-request-CR-lock (TR_ID_A, CRG_A)
var
NDJ_CLIENT , a client in Participants of T_A
CRL_A  the CR lock of T_A
CODL_A  the COD lock of T_A
begin
  for (CRG_i in all client transactions)
  begin
    check Non Disjoint state
    if (Non Disjoint) then
      add CRG_i's TR_ID_i to Participants
  end

  set CRL_A with CRG_A

  for (NDJ_CLIENT in Participants)
    add TR_ID_A to NDJ_CLIENT's Participants

  get the CODL_A

  send reply-request-CR-lock msg
end
```

그림 11 CR 잠금의 설정 알고리즘

6.4.2 CX 잠금의 설정

트랜잭션 T_A의 CX 잠금 설정 알고리즘은 그림 12와 같다. *request-CX-lock* 연산은 수정 객체들을 정의하여 *Agent*로 보낸 뒤 잠금 충돌 여부를 *Agent*로 부터 기다린다. *Agent*에게서 reply-CX-ok가 왔을 때에는 충돌이 없는 것으로 잠금이 설정되어 권한을 얻은 것이므로 수정을 해 나갈 수 있다. 그러나 reply-CX-conflict가 왔을 때에는 충돌이 있는 것이므로 wake-up이 올 때까지 대기하여야 한다. *Agent*에서 수행되는 svr-rcv-request-CX-lock 연산은 잠금 호환성에 따라 reply-request-CX-ok 혹은 reply-request-CX-conflict를 응답 메시지로 보내어 동시성을 제어한다.

```
function request-CX-lock
var
TR_ID_A : the transaction id of T_A
UDO_A : the UpDating Objects of T_A
CXL_A : the CX lock of T_A
begin
  define the object set for CXL_A
      and get UDO_A

  send request-CX-lock msg
      with TR_ID_A and UDO_A to Agent

  for (timeout)
    wait for response

  If (reply-CX-conflict) then
  begin
    wait for wake-up
    If (wake-up) then
      write its own UDO_A to log
  end
  else
    write its own UDO_A to log
end
```

```
function svr-rcv-request-CX-lock (TR_ID_A, UDO_A)
var
NDJ_CLIENT : a client in Participants of T_A
BLCK_LIST_A : the blocked transaction id list due to T_A
CXL_A  the CX lock of T_A
begin
  get TR_ID_A's Participants from catalog

  for (NDJ_CLIENT in Participants)
  begin
    check global SR dependency
    If (dependent) then
      add TR_ID_A to NDJ_CLIENT's BLCK_LIST_A
  end

  if (there is a global SR dependent NDJ_CLIENT) then
  begin
    write UDO_A with blocked status to catalog
    send reply-CX-conflict
  end
  else
  begin
    set CXL_A with UDO_A
    send reply-CX-ok
  end
end
```

그림 12 CX 잠금의 설정 알고리즘

6.4.3 mid-commit

*mid-commit*의 SR 기반 2PC 처리 알고리즘은 그림 13과 같다. *mid-commit* 연산은 T_A의 $DELTA_A$를 *Agent*로 보내고 응답을 기다린다. *Agent*에서 수행되는 svr-rcv-mid-commit 연산에서는 T_A의 *Participant*에게 *mid-commit*을 전파하고 사용자의 판단에 의한 응답을 기다린다. 모든 *Participant*가 *mid-accept*를 보내오면 *mid-commit*은 성공하여 global-mid-commit을 T_A및 *Participant*에게 전파하여 $DELTA_A$를 병합하게 하고, *Participant* 중 *mid-reject*를 보내오는 클라이언트 트랜잭션이 있을 경우 global-mid-abort를 전파하여 *mid-commit*한 내용을 취소하게 한다.

```
function mid-commit
var
DELTA_A : the DELTA of T_A
UDO_A : the UpDating Objects of T_A
begin
  get DELTA_A from UDO_A
  send mid-commit to Agent
  for (timeout)
    wait for response
  if (send-global-mid-commit) then
    merge DELTA_A to delta stack
  else if (send-global-abort) then
    delete DELTA_A
end

function clnt-rcv-mid-commit (DELTA_A)
begin
  display DELTA_A
  wait for user's decision
  if (user's decision is accept) then
    send reply-mid-accept to Agent
  else
    send reply-mid-reject to Agent
  for (timeout) wait for response
  if (send-global-commit) then
    merge DELTA_A to delta stack
  else if (send-global-abort) then
    delete DELTA_A
end

function svr-rcv-mid-commit (TR_ID_N, DELTA_A)
var
NDJ-CLIENT : a client in Participants
begin
  get TR_ID_A's Participants from catalog

  for (NDJ-CLIENT in Participants)
    send DELTA_A to NDJ-CLIENT

  for (timeout)
    wait for responses of Participants
  if (there is any reply-mid-reject) then
  begin
    for (NDJ-CLIENT in Participants)
      send send-global-mid-abort to NDJ-CLIENT
    delete DELTA_A
  end
  else if (all reply-mid-accept) then
  begin
    for (NDJ-CLIENT in Participants)
      send send-global-mid-commit to NDJ-CLIENT
    merge DELTA_A to delta stack
  end
end
```

그림 13 *mid-commit*의 알고리즘

6.5 메시지 부하의 최소화에 대한 고려 사항

캐쉬를 사용하는 클라이언트-서버 공간 데이타베이스 환경에서는 캐쉬 사용의 이점을 최대한 얻기 위해 메시지 부하의 최소화에 대한 고려가 필요하다. 변경 전파 프로토콜에서의 메시지 부하 최소화는 동시성 제어와 일관성 유지 면에서 고려하였다.

동시성 제어 시 고려된 사항은 다음과 같다. 첫째, CR 잠금 영역 지정 시 영역의 최소화에 대한 문제이다. CR 잠금 영역의 크기가 커질 경우 동시에 수정하는 클라이언트들의 CR 잠금 영역과 겹치는 부분이 발생할 가능성이 높아진다. 즉 이는 협동 작업 가능성을 높이고 협동 작업을 위한 메시지를 추가적으로 발생시키므로 메시지 부하를 높인다. 그러므로 이 논문에서는 CR 잠금의 영역 설정 시 사용자가 직접 최소 수정 영역을 지정함으로써 불필요한 메시지 부하 증가를 방지한다.

둘째,[1]에서는 분산환경을 대상으로 하므로 global-mid-commit을 전체 사이트에 전파한다. 그러나, 다수의 클라이언트가 서버에 접근하게 되는 클라이언트-서버 환경에서 *mid-commit*의 결과를 모든 클라이언트들에게 전파할 때에는 전체 클라이언트 수가 늘어날수록 메시지 수가 비례하여 증가한다. 메시지 부하가 높아질수록 캐쉬 사용의 목적인 사용자 응답 시간 지연 최소화를 달성할 수 없으므로, 이 논문에서는 global-mid-commit의 전파 대상을 협동 작업을 하는 클라이언트들로 제한한다.

캐쉬의 일관성 유지 시 고려된 사항들은 다음과 같다. 이 논문에서는 협동 작업에 참여하지 않는 클라이언트 캐쉬의 일관성 유지를 위해서 CS 잠금 및 COD 잠금을 제시하여 탐지기반(detection-based) 방법을 사용한다. 즉 클라이언트가 서버에게 자신이 가진 데이타가 유효한지를 탐지하는 방법에 기반하여 일관성을 제어하는 것이다. 이는 회피기반(avoidance-based) 방법을 사용하는 것에 비하여 전체 프로토콜을 간단하게 해주는 이점이 있다[8]. 또한, 탐지 시기를 최대한 늦추는 낙관적 방법을 사용한다. 이를 위해 탐지 시기를 세 시점으로 제한한다. 첫 번째는 CR 잠금을 서버에 요청할 때 영역 내의 모든 객체들의 유효성을 탐지한다. 두 번째는 서버에 새로운 객체를 요청할 때 탐지한다. 끝으로 사용자가 탐지를 요청할 때 탐지를 한다. 비관적인(pessimistic) 탐지 기반 기법에서는 트랜잭션 내에서 각 객체에 접근할 때마다 서버로부터 그 객체의 유효성을 탐지함으로써 메시지 과부하가 높아지는 단점이 있었다. 그러나 낙관적 기법을 사용함으로써 메시지 과부하를 최소화 할 수 있다.

6.6 프로토콜 예제

이 절에서는 프로토콜 예제를 소개한다. 그림 14는 도로 확장 공사에 의한 지도 수정 과정을 보인 그림이다. 그림 14에서 클라이언트 A의 트랜잭션 T_A는 도로1(Road 1) 객체를 수정하고 클라이언트 B의 트랜잭션 T_B는 필지1(Property 1) 객체를 수정하며 T_A의 수정이 T_B의 수정보다 앞섬을 가정한다.

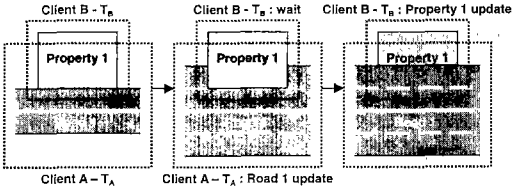


그림 14 도로 확장 공사에 의한 지도 수정

그림 14에서의 필지1 객체와 도로1 객체는 서로 Meets의 공간 관련성이 있으며, 클라이언트 A의 트랜잭션 T_A가 도로1 객체를 수정하고 클라이언트B의 트랜잭션 T_B는 필지1 객체를 수정하므로 전역 공간 관련성이 또한 존재한다. 필지 1 객체와 도로 1 객체의 수정 시, 각 트랜잭션 T_A와 T_B의 수정 후에도 이러한 Meets의 전역 공간 관련성은 유지되어야 하므로 이로 인한 변경 종속성이 존재하게 된다.

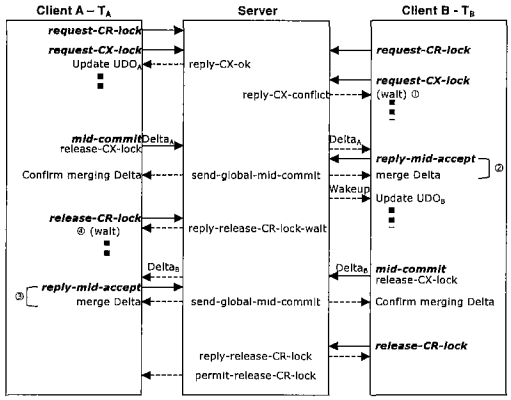


그림 15 변경 전파 프로토콜의 예

그림 15는 트랜잭션 연산을 이용하여 그림 14에 대한 변경 전파 프로토콜을 보이고 있다. 그림 15 에서 T_B의 수정 객체 필지1은 T_A의 수정 객체 도로1과 전역 공간 관련성에 의한 종속이므로 CX 잠금의 요청이 허용되지 않는다(①). 그러므로 T_B는 대기하였다가 T_A의 도로1에 대한 수정 작업이 완료된 후에 변경을 시작할 수 있다. T_A의 도로1에 대한 수정 결과는 협동 작업을 하는 클라이언트2로 *mid-commit*에 의해 전파되고 T_B와 SR기반 2PC를 수행한다(②). T_B가 *mid-accept*로 응답했을 경우 서버는 global-mid-commit을 전파하여 도로1에 대한 수정을 완료한다. 그리고 T_B는 수정작업이 완료된 후 *mid-commit*을 서버에 전파하고 서버는 T_A와 SR 기반 2PC를 수행하여 T_A의 확인을 받는다(③). ④에서 T_A는 수정 작업을 끝냈지만 T_B와 SR 기반 2PC를 수행하여 지도 수정의 정확성을 상호 검증하여야 하므로 대기하게 된다. 따라서 이러한 상호 검증을 통하여 전역 공간 관련성에 의한 변경 종속성을 제어한다.

7. 구현

7.1 시스템 구조 및 설계

그림 16은 이 논문의 실험 구현 환경과 시스템 구조이다. 서버는 객체지향 GIS 도구인 Gothic 3.0을 대상으로 구현하였으며, 클라이언트는 Win98/NT환경에서 구현하였다. 'Gothic Data Server'는 Gothic 3.0의 객체지향 DBMS이다.

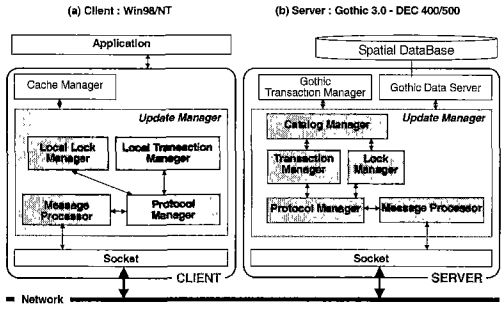


그림 16 구현 환경 및 시스템 구조

그림 16 (a)에서 구현한 클라이언트의 수정 관리 모듈(Update Manager)은 4개의 모듈로 구성되며 각각의 기능은 다음과 같다.

- 프로토콜 관리 모듈(Protocol Manager) : 변경 전파 프로토콜을 처리한다.
- 지역 트랜잭션 관리 모듈(Local Transaction Manager) : 클라이언트에서의 지역 트랜잭션을 관리한다.
- 지역 잠금 관리 모듈(Local Lock Manager) : 서버에 설정된 잠금을 임시 저장 관리한다.
- 메시지 처리 모듈(Message Processor) : 클라이언트의 요청 메시지를 서버로 보내고 서버로 부터의 메시지를 프로토콜 관리 모듈에 전한다.

그림 16 (b)에서 구현한 서버의 수정 관리 모듈(Update Manager)은 5개의 모듈로 구성되며 각각의 기능은 다음과 같다.

- 프로토콜 관리 모듈(Protocol Manager) : 변경 전파 프로토콜을 처리한다.
- 트랜잭션 관리 모듈(Transaction Manager) : 서버에 접근하는 모든 클라이언트 트랜잭션들을 전역적으로 관리한다.
- 잠금 관리 모듈(Lock Manager) : 확장된 잠금을 기록하고 잠금 요청에 대한 호환성을 제어한다.
- 정보 저장소 관리 모듈(Catalog Manager) : 프로토콜의 실행을 위해 필요한 사전 정보와 프로토콜 실행 중 생성되는 제어 정보들을 저장하고 이용하기 위한 정보 저장소를 관리한다.
- 메시지 처리 모듈(Message Processor) : 클라이언

트의 요청 메시지를 처리하여 메시지를 프로토콜 관리 모듈에 전하고, 서버가 처리한 결과를 클라이언트들에게 전파한다.

이 논문에서는 Gothic 3.0의 기존 잠금 기능과 별도의 잠금 관리 모듈을 구현하였다. 즉 실제 서버의 데이타 집합에 WRITE 잠금을 설정하고 관리하는 것은 Gothic 3.0이며, 변경 전파 프로토콜에 의한 확장된 잠금의 설정을 관리하는 것은 별도로 구현한 잠금 관리 모듈이다.

또한 Gothic 3.0의 트랜잭션 관리 모듈을 이용하는 별도의 트랜잭션 관리 모듈을 구현하였다. 왜냐하면 Gothic 3.0의 트랜잭션 관리 모듈이 서버 내의 지역 트랜잭션 관리 기능만을 가지고 있기 때문이다. 이 논문에서는 각 클라이언트들의 수정 트랜잭션들을 전역적으로 관리하는 트랜잭션 관리 모듈을 구현하였다.

7.2 구현 예

그림 17은 구현된 수정 관리 모듈을 이용하여 두 클라이언트에서 각각 건물과 필지 객체를 동시에 수정하는 예를 보이고 있다. 현재 클라이언트 화면에 출력된 데이타 집합은 창원 UIS에서 사용되는 실제 데이타이다. 또한 지도 화면 위의 사각형은 사용자에 의해 설정된 CR 잠금의 영역이다. 클라이언트 1은 건물 객체를,

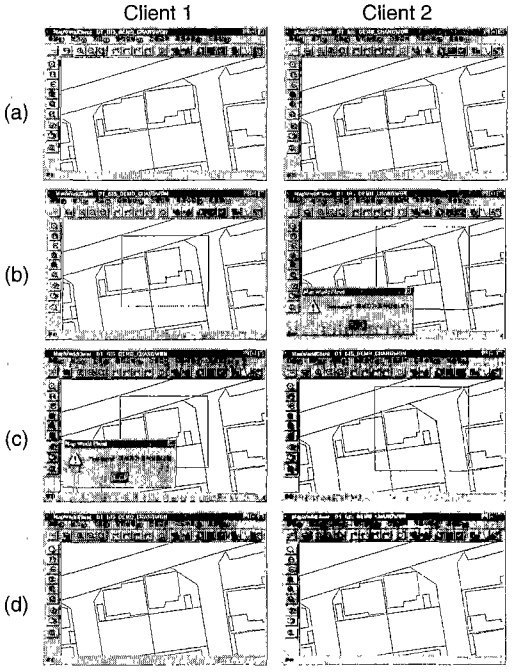


그림 17 구현된 수정 관리기를 이용한 지도 수정 예

그리고 클라이언트 2는 필지 객체를 수정한다. 클라이언트1은 건물 객체를 수정 후 DELTA를, 서버를 통하여 클라이언트2로 전파한다. 클라이언트2가 클라이언트1의 수정 결과를 받아들이고, 그림 17 (c)에서 자신의 수정 결과를 클라이언트1로 전파한다. 이러한 협동 작업 과정의 제어는 CR 잠금, CX 잠금, SR기반 2PC에 의해 제어된다.

7.3 평가

이 절에서는 제안한 시스템의 평가를 위하여 매개 변수를 기술하고, 이 매개 변수들로부터 수식을 유도하여 한 서브-트랜잭션의 commit 당 메시지 개수와 협동 작업에 참여치 않는 클라이언트 캐쉬의 일관성을 유지하기 위한 메시지 개수를 평가한다.

먼저, 평가의 범위를 좁히고 다양한 변수들을 간소화하기 위해 두 가지 사항을 가정한다.

- 협동 작업자의 수정 결과에 대한 판단(voting)은 승인(accept)됨을 가정한다.
- 한 객체 단위로 변경 전파를 실시한다.

이 가정에 의하여, 평가에 사용되는 매개 변수를 다음과 같다.

- C_T : 전체 클라이언트 수
- C_P : 협동 작업에 참여하는 클라이언트 수($C_T=C_P$)
- N_R : 한 서브-트랜잭션에서 READ하는 객체 수
- N_W : 한 서브-트랜잭션에서 WRITE하는 객체 수

제안 시스템에서 한 서브-트랜잭션에서의 commit당 메시지 개수는 첫째, CR 잠금의 설정, 둘째, CX 잠금의 설정, 셋째, *NDJ-relation*을 가진 클라이언트와의 SR 기반 2PC 프로토콜, 넷째, CR 잠금의 해제의 합으로 구성된다. 먼저, CR 잠금 설정을 위해서는 서버에 CR 잠금 설정 요청을 해서 설정 응답을 받음으로 2개의 메시지($M_{CR\ 잠금\ 설정}$)가 소요된다. 다음으로 CX 잠금 설정을 위한 메시지도 같은 이유로 2개의 메시지 ($M_{CX\ 잠금\ 설정}$)가 소요된다. 단 CX 잠금의 설정을 위한 메시지는 N_W의 수만큼 반복되어야 한다.

SR 기반 2PC 프로토콜을 위한 메시지 개수는 다음 식으로 표현된다.

- $M_{SR\ 기반\ 2PC}=2+3C_P$

즉, *mid-commit*을 발생시키는 클라이언트가 서버로 *mid-commit*을 보내는 메시지와 서버가 SR 기반 2PC의 결과를 통보하는 메시지, 그리고 서버가 SR 기반 2PC를 수행하기 위해 협동 작업하는 클라이언트들에게 *mid-commit*을 전파하고 사용자들의 판단(voting)을 얻은 뒤 SR 기반 2PC의 결과를 다시 전파해 주는 메시지로 구성된다. 그리고, 이 메시지 역시 N_W의 수만큼

반복되어야 한다.

그러므로, 전체 메시지 개수는 다음 식으로 표현할 수 있다.

- $M = M_{CR\ 잠금\ 설정} + N_W(M_{CX\ 잠금} + M_{SR\ 기반\ 2PC}) + M_{CR\ 잠금\ 해제} = 2 + N_W(2+2+3C_P) + 3 = 5 + N_W(4+3C_P)$

본 논문에서는 5.1절에서 설명되어 진 바와 같이, 클라이언트 캐쉬의 일관성을 유지하기 위하여 낙관적 탐지기반의 일관성 유지 기법을 적용함으로써 클라이언트 캐쉬의 일관성을 유지하기 위한 추가적인 메시지는 필요치 않는 다. 이는 클라이언트가 서버에 새로운 데이터를 요청 시 캐쉬의 유효성을 확인하므로 추가적인 메시지가 발생치 않으며, 클라이언트의 수정 작업 시에는, CR잠금 설정을 요청 시 클라이언트 캐쉬의 유효성을 확인하므로 추가적인 메시지 없이 Dirty Read 와 Lost Update 문제 또한 발생시키지 않는 다.

위 수식을 비교하기 위해 동기식 중복 일관성 제어 시스템의 commit당 메시지 개수를 간략히 수식화하였다.

- $M_{sync} = 2(1+C_T)N_R + 3(1+C_T)N_W$

두 시스템의 commit 당 메시지 개수에 대한 비교는 N_R과 C_T매개 변수에 의해 판단할 수 있다. C_P는 C_T보다 적으며, 동기식 제어 기법은 READ하는 객체 수(N_W) 및 서버에 접속하는 전체 클라이언트 수(C_T)에 비례하는 메시지를 필요로 한다. 따라서, 제안 시스템은 동기식 중복 일관성 제어 시스템 보다 적은 commit당 메시지 개수를 필요로 함을 알 수 있다.

8. 결론 및 향후 연구

이 논문에서는 서버의 공간 데이타가 클라이언트 캐쉬에 동적 중복 저장 되어있는 클라이언트-서버 공간 데이타베이스 환경에서 클라이언트 지도 수정 트랜잭션들의 동시성과 캐쉬 일관성 제어를 지원하기 위한 기법들을 다루었다. 동시성 지원을 위해서는, CR 잠금 및 CX 잠금을 제시하여 캐쉬를 사용하는 클라이언트-서버 공간 데이타베이스 환경에서 공간 객체간의 공간 관련성에 의한 변경 종속성으로 인하여 기존의 제어 기법으로 지원할 수 없는 클라이언트 트랜잭션들의 향상된 동시성을 지원하였다. 또한 클라이언트 캐쉬의 일관성 유지를 위해서는, CS 잠금 및 COD 잠금을 제시하여 메시지 부화의 최소화를 고려한 탐지기반의 일관성 유지 기법을 제시하였다. 그리고 SR 기반 2PC를 도입하여 클라이언트-서버 공간 데이터베이스 환경에서 일관성을 제어하는 변경 전파 프로토콜을 설계 및 구현하였다. 이 논문에서 다루지 못한 부분은 사용자들의 데이터 수정 작업 양식에 따라 추가적으로 발생할 수 있는 부하를 연구하여 변경 전파 프로토콜을 확장하는 것이며 또한, 수정 대상이 되는 공간 객체가 차지하고 있는 영역이 큰 경우의 동시성 향상 기법에 관한 향후 연구가 필요하다.

참 고 문 헌

[1] Jin-oh Choi, Young-sang Shin, and Bong-hee Hong, Update Propagation of Replicated Data in Distributed Spatial Databases, Proc. of 10th International Conference on DEXA '99, pp. 952-963, 1999

[2] Michael J. Franklin, Miron Livny, and Eugene J. Shekita, Data Caching Tradeoffs in Client-Server DBMS Architectures, Proc. of 1991 ACM SIGMOD International Conference on Management of Data, vol. 20, pp. 357-366, 1991

[3] Michael J. Franklin, Micahel J. Carey, and Miron Livny, Local Disk Caching for Client-Server Database Systems, Proc. of 24th International Conference on VLDB, pp. 641-654

[4] Am-suk Oh, Jin-oh Choi, and Bong-hee Hong, An Incremental Update Propagation Scheme for a Cooperative Transaction Model, Proc. of International Workshop on DEXA '96, pp. 353-362, 1996

[5] Kevin Wilkinson, and Marie-Anne Neimat, Maintaining Consistency of Client-Cached Data, Proc. of 21th International Conference on VLDB, pp. 122-133, 1990

[6] M.J. Egenhofer, Reasoning about binary topological relations, 2th International Symposium, SSD'91, pp. 143-160, 1991

[7] Gail E. Kaiser, Cooperative Transactions for Multiuser Environments, Modern Database Systems, Addison Wesley, pp. 409-433, 1995

[8] Michael J. Franklin, Michael J. Carey, and Miron Livny, Transactional Client-Server Cache Consistency : Alternative and Performance, ACM TODS, vol. 22, no. 3, September 1997

[9] R. Laurini, Fundamentals of Spatial Information

Systems, Academic Press, 1992

[10] H. berenson, P. Nernstein, J. Gray, J. Melton, E. O'Neil, and P. O'Neil, A Critique of ANSI SQL Isolation Levels, Proc. of the 1995 ACM SIGMOD, pp. 1-10, 1995

[11] J. N. Gray, R. A. Lorie, and G. R. Putzolu, Granularity of Locks in a Large Shared Data Base, Proc. of the 1st International Conference on VLDB, pp. 428-451, September 1975

[12] Alfons Kemper, and Guido Moerkotte, Chapter 15. Transaction Control, Object-Oriented Database Management: Applications in Engineering and Computer Science, Prentice-Hall, pp. 379-410, 1994

[13] 정일영, 이종민, 황종선, 데이타 객체의 집중에 기반한 클라이언트 캐쉬의 일관성 유지, 한국정보과학회 논문지, vol. 25, no 2, pp. 264-279, 1998

[14] 최진오, 홍봉희, 분산된 지리정보시스템에서 새로운 잠금 기법을 이용한 중복된 공간 데이타의 변경 전파, 한국정보과학회 논문지, vol 26, no 9, pp. 1061-1072, 1999

[15] 신영상, 최진오, 홍봉희, 클라이언트-서버 환경에서 캐쉬 공간 데이타의 변경 전파, '99 한국정보과학회 봄 학술발표논문집, vol 26, no 1, pp. 86-88, 1999

[16] 신영상, 최진오, 조대수, 홍봉희, 클라이언트 변경 트랜잭션에서 동시성 및 일관성 제어, '99 한국정보과학회 가을 학술발표논문집, vol 26, no 2, pp. 323-325, 1999

신 영 상

1998년 2월 부산대학교 컴퓨터공학과 공학사. 2000년 2월 부산대학교 컴퓨터공학과 공학석사. 현재 University of Wisconsin-Madison, 전산학과 박사과정. 관심분야는 인터넷 데이타베이스, XML, 트랜잭션, 공간 데이타베이스, 클라이언트-서버 및 분산 데이타베이스

홍 봉 희

정보과학회논문지 : 데이타베이스
제 28 권 제 1 호 참조