이동 컴퓨팅 시스템에서 캐쉬 록을 이용한 동시성 제어 기법

김 치 연[†] · 황 부 현[††]

요 약

이동 컴퓨팅 환경에서 이동 호스트가 사용하는 캐쉬는 이동 호스트의 전력 소모를 줄이면서 시스템 성능을 향상시킬 수 있는 중요한 장치이다. 이동 호스트는 자주 접근하는 데이타를 서버로부터 캐쉬하여 사용한다. 캐쉬를 사용할 때 발생하는 문제점은 이동 호스트에서 캐쉬된 데이타를 사용하여 수행되는 판독 전용 트랜잭션과 서버에서 수행되는 갱신 트랜잭션의 수행 순서에 따라 트랜잭션 직렬성이 파괴될 수 있다는 점이다. 지금까지 제안되어 온 캐싱 기법을 다룬 많은 연구들은 트랜잭션 직렬성 유지보다는 주로 캐쉬 내 데이타와 서버에 저장된 데이타의 일관성에 초점을 두고 있다. 본 연구에서는 이동 호스트가 캐쉬를 사용할 때, 판독 전용 트랜잭션과 갱신 트랜잭션 사이의 직렬성을 보장할 수 있는 두 가지 캐싱 기법을 제안하고, 이 중 한 가지 방법에 기반을 둔 동시성 제어 기법을 제안하고자 한다. 제안하는 동시성 제어 기법은 록킹 방법에 기반을 두고 있으며, 캐쉬 록이라는 새로운 록 모드를 도입함으로써 트랜잭션 직렬성을 유지하도록 하였다.

A Concurrency Control Method using Cache Lock in Mobile Computing Systems

Chi Yeon Kim[†] · Bu Hyun Hwang[††]

ABSTRACT

A cache used by a mobile host in mobile computing environments is the important device that can reduce the power consumption and improve the system performance. A mobile host uses the cache to store the data that can be frequently accessed. When read-only transactions and update transactions under the caching environment are executed, the serializability of the transactions can be destroyed. Many studies focus on the consistency between the data in the cache and the data in the server rather than maintenance of transaction serializability. In this paper, we propose two caching strategies in which transaction serializability is maintained between read-only transactions and update transactions. And we propose a concurrency control method that uses one of two caching strategies. The proposed concurrency method, which is based on the locking, guarantees the serializability of transactions by introducing the new lock mode that is called a cache lock.

1. 서 론

최근에 급속히 확산되고 있는 이동 컴퓨팅 환경은 무선 통신 기술의 발전이 가져다 준 편리한 환경이다. 사용자들은 이동 가능한 컴퓨터나 PDA(Personal Digital Assistance)를 사용하여 자신의 위치나 장소에 상관없이 데이타베이스에 접근한다. 이동 사용자들은 전자 메일이나 주식 정보, 또는 여행 중에 필요한 여러

※본 논문은 1997년도 한국 과학 재단 핵심 전문 연구비에 의하여 연구되었음.

† 정 회 원:전남대학교 전산학과 시간강사
†† 정 회 원:전남대학교 전산학과 교수

논문접수:1997년 7월 23일, 심사완료:1997년 11월 24일

가지 정보들을 이동 컴퓨터를 사용하여 이동 트랜잭션을 수행함으로써 얻고자 한다[1]. 이러한 형태의 요구는 이미 존재하고 있는 분산 시스템의 설계 목적과는 많이 달라서 분산 시스템을 기반으로 한 새로운 형태의 이동 컴퓨팅 시스템을 필요로 하게 되었다.

이동 컴퓨팅 시스템은 사용자들의 새로운 요구를 반영하기 위해 대두된 시스템이기 때문에 다음과 같은 점에서 기존의 시스템과는 다르다. 첫 번째는 무선 통신망을 사용하는 것이다[7]. 사용자의 이동 컴퓨터는 휴대가 가능하도록 소형화되고 있고, 또한 사무실이나 가정과 같은 정적인 환경 뿐 아니라 여행 중이거나 출장시에도 사용 가능하도록 하기 위해서 무선 통신망을 매체로 하여 데이타베이스에 접근한다. 하지만 무선 통신망은 유선 통신망에 비하여 신뢰성이 낮고 현재 할당되어 있는 대역폭이 그리 넉넉하지 못하다는 단점을 갖고 있다[4]. 두 번째는 사용자의 이동성이다. 사용자의 이동성은 이동 컴퓨팅 환경의 가장 큰 특징이라 할 수 있는데, 이동 사용자들은 이동 중에도 서버와 연결을 유지하면서 작업을 수행하기를 원한다. 하지만 여기에서의 한계는 이동 컴퓨터가 갖는 전력이다[5]. 따라서 이동 컴퓨터는 자신의 밧데리 소모를 최대한 줄이기 위한 방법을 모색하게 되었는데, 이 중 하나가 연산하지 않을 때는 서버와의 연결을 끊고 단절하는 것이다[6, 8, 9]. 만약 이동 호스트가 캐쉬를 갖고 있다면, 서버와 연결이 끊어질 때에도 연산을 계속할 수 있으나 이러한 단절로 인해 캐쉬내의 데이타와 서버에 저장된 데이타 상태가 달라질 수 있다. 이 때에는 재연결시 서버내의 정보를 이용하여 캐쉬내의 데이타를 수정하는 과정이 필요하다.

무선 통신망이 갖는 대역폭의 한계를 극복하기 위해 이동 컴퓨터는 캐쉬를 사용할 수 있다. 이동 호스트에서 데이타를 접근할 때마다 서버에 접근하는 것은 전력 소모나 통신 비용의 측면에서 그리 바람직하지 못하다. 따라서 이동 호스트는 자주 사용하는 데이타를 자신의 캐쉬에 저장해 두고 연산을 하는데, 이 때 캐쉬된 데이타는 서버에 저장된 데이타와 일치되어야 한다.

이동 컴퓨팅 환경에서 서버와 이동 호스트는 방송(broadcasting)을 통하여 상호 작용한다. 이동 호스트가 캐쉬를 사용할 때 캐쉬된 데이타의 일관성을 유지하기 위해 대부분의 방법에서는 주기적인 방송 방법을 사용하였다. 이동 호스트가 캐쉬한 데이타를 다루는 방법인 캐싱 기법은 캐쉬내 일관성 유지 기법과 대치 기법으로 분류할 수 있다[13]. [12]에서는 주기적인 방송을 통하여 캐쉬된 데이타의 일관성을 유지하는 방법을 제안하였다. 즉, 일정 시간 동안 갱신된 데이타 항목들에 대한 정보를 주기적으로 방송함으로써 이동 호스트의 캐쉬에 있는 데이타가 가장 최근의 값을 갖도록 하였다. 이 방법은 갱신된 데이타 값을 캐쉬에 반영하는 데에 초점을 두고 있어서 서버에서 수행되는 갱신 트랜잭션과 이동 호스트에서 수행되는 트랜잭션 사이의 직렬화 순서가 유지되지 않는 경우가 발생한다. 캐쉬를 사용할 때 캐쉬의 일관성이 위배될 수 있는 또 다른 경우는 이동 호스트가 서버와 단절되는 경우이다. 이동 호스트가 단절되면 방송된 메시지를 수신하지 못하여 갱신된 데이타 값을 캐쉬에 반영하지 못하게 된다. 또한 단절이 아니어도 이동 호스트가 셀의 경계를 넘어 이동하는 경우 방송 메시지를 수신하지 못하여 캐쉬내 데이타의 일관성이 파괴되는 경우도 발생한다.

이동 컴퓨팅 환경에서 캐쉬는 유선 환경에서보다 여러 가지 이유로 유용한 장치이다. 따라서 많은 연구들이 데이타를 캐쉬하는 방법이나 캐쉬내 데이타의 일관성을 유지하기 위해 제안되어지고 있는데, 트랜잭션의 직렬가능한 수행을 고려한 연구들은 많지 않다. 따라서 본 연구에서는 이동 호스트가 캐쉬를 사용하는 경우, 트랜잭션이 하나의 방송 주기 동안 수행을 완료하지 못하는 경우라 해도 수행되는 트랜잭션의 직렬성을 유지할 수 있는 두 가지 캐싱 기법을 제안하고자 한다. 그리고 이러한 캐싱 기법을 기반으로 하여, 록을 이용한 동시성 제어 기법을 제안한다. 제안하는 동시성 제어 방법의 정확성 기준으로는 기존의 유선 환경에서 트랜잭션 수행의 정확성을 측정하는 척도로 가장 많이 사용되는 직렬성(serializability)[2]을 사용하고 있다.

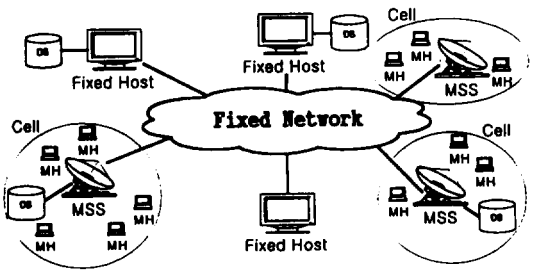
본 연구의 구성은 다음과 같다. 2장에서는 캐싱 기법에 관하여 제안되어 온 연구들을 살펴보고 문제점들을 정의하며, 3장에서는 제안하는 두 가지 캐싱 기법에 대하여 트랜잭션의 직렬성 유지에 초점을 두고 일관성이 유지되는 캐싱 알고리즘과 간접 충돌의 해결 방법에 대하여 기술한다. 4장에서는 록을 이용한 새로운 동시성 제어 방법을 소개하며, 5장에서는 결

론과 앞으로의 연구 방향을 기술한다.

2. 관련 연구

2.1 이동 컴퓨팅 환경 모델

이동 컴퓨팅 환경의 모델은 (그림 1)과 같다. 이동 컴퓨팅 환경은 고정 호스트(FH:Fixed Host)와 이동 호스트(MH:Mobile Host)로 구성되어 있다. 고정 호스트중 이동 호스트와 통신할 수 있는 무선 인터페이스를 갖춘 호스트를 서버 또는 이동 지구국(MSS:Mobile Support Station)이라고 하는데[3], 이동 호스트(이동 컴퓨터)들이 트랜잭션을 처리하는데 필요한 데이타를 제공하거나 이동 호스트가 다른 호스트들과 통신할 수 있는 통로를 제공한다. 또한 서버는 유선 통신망에 연결되어 있는 고정 호스트들이 제출한 작업들을 처리하기도 한다. 그리고 하나의 서버에 의해 제어되는 지리적인 영역을 셀 이라 한다.



(그림 1) 이동 컴퓨팅 환경 모델
(Fig. 1) The mobile computing environment model

이동 호스트는 이동 컴퓨터나 여러 가지 휴대용 통신 기기가 될 수 있고, 이동 호스트의 사용자를 이동 사용자라 한다. 이동 호스트가 다른 호스트와 통신하기 위해서는 이동 지구국을 경유해야 한다. 이동 컴퓨팅 환경에서 이동 사용자는 데이타베이스의 검색이나 원하는 작업을 수행하기 위한 방법으로 트랜잭션을 제출한다. 이동 호스트에서 수행되는 트랜잭션을 이동 트랜잭션[11]이라 하는데, 이동 호스트에서는 주로 판독 전용 트랜잭션이 수행되고, 반면에 이동 지구국에서는 기록 연산으로 구성되는 갱신 트랜잭션이 수행된다.

2.2 문제 정의

본 절에서는 이동 컴퓨팅 환경의 캐쉬 관리 기법이나 동시성 제어를 다룬 기존의 연구들과, 기존의 방법에서 발생하는 문제점에 대하여 살펴본다.

[10]에서는 이동 컴퓨팅 환경에서 판독 전용 트랜잭션을 지원하기 위한 캐쉬 정책을 제안하였다. 이동 컴퓨팅 환경의 가장 두드러진 특징은 사용자의 이동성이라 할 수 있는데, 이러한 이동성으로 인하여 캐쉬내 일관성 유지가 어려워진다. 캐쉬내 일관성을 유지하기 위한 방법으로 무효화 메시지(invalidation message)를 송수신하는 방법이 있다. 무효화 메시지는 서버에서 갱신된 항목을 이동 호스트에 알리는 방송 메시지이다. 캐쉬를 사용할 때 이동 호스트가 이동으로 인하여 이 메시지를 수신할 수 있는 기회를 상실하게 되면 캐쉬내 일관성은 보장되기 어렵다. [10]에서는 이러한 경우에 캐쉬내 일관성을 유지하는 방법을 제안하였다. 트랜잭션들 사이의 직렬성을 유지하기 위하여 다중 버전을 제공하고 있고, 버전 관리를 위해 서버에 두 개의 카운터를 유지하였다. 또한 서버에서 메시지를 방송할 때 타임스탬프를 함께 방송함으로써 이동 호스트는 캐쉬내 데이타를 갱신할 때 캐쉬의 타임스탬프도 갱신한다. 캐쉬 일관성은 캐쉬내의 데이타가 캐쉬의 타임스탬프보다 작은 타임스탬프를 갖는 트랜잭션이 갱신한 결과만을 보유함으로써 유지됨을 보였다. 이동 호스트가 서버에 데이타를 요구하는 경우, 캐쉬의 타임스탬프와 서버에서 유지되는 타임스탬프에 따라 요구하는 데이타를 전달하기도 하고 요구를 철회하기도 한다. 이동 컴퓨터는 서버로부터 갱신된 데이타 항목에 대한 무효화 메시지를 받으면 메시지에 따라 캐쉬내에 있는 데이타를 재캐쉬하는데 이 때 이동 호스트의 요구가 대부분 같은 시간대에 발생하게 되어 특정 시간에 트래픽이 집중될 수 있다.

[11]에서는 이동 트랜잭션의 동시성 제어를 위해 록킹 방법을 사용할 때 록 해제를 위한 부가적인 메시지 교환을 줄일 수 있는 방법을 제안하였다. 판독 록을 해제할 때 반드시 록을 얻은 사이트가 아니라도 이동 호스트가 위치하고 있는 임의의 사이트에서 해제할 수 있도록 함으로써 사용자의 이동성을 반영하고 록 해제로 인해 교환되는 메시지 수를 줄였다. 판독 록은 요청 즉시 수여 가능하고, 갱신 록은 2단계 완료 (2PC:Two-Phase Commit) 프로토콜의 첫 번째

단계에서 얻을 수 있도록 하였다. 이 연구에서 데이타베이스는 중복된 환경으로 가정하고 있고, 중복 관리를 위해서는 과반수 프로토콜(quorum protocol)을 채용하고 있다. 또한 트랜잭션의 직렬성을 위해서 버전을 사용하고 있다. 갱신 록은 완료 프로토콜의 첫 번째 단계의 시작 시점에서 갱신하는 모든 데이타에 대한 갱신 록을 얻어야 하는데 이를 위해 새로운 록 모드를 도입하였다. 그러나 이 단계에서 모든 록을 얻지 못하면 완료 프로토콜의 다음 단계를 진행하지 못하게 되어 트랜잭션의 완료가 지연될 수 있다. 또한 부분적으로 소수의 사이트에 저장되어 있는 데이타 항목의 경우에는 임의의 사이트에서 록을 해제할 수 없어 부가적인 메시지 교환을 필요로 하게 된다. 또한 이 연구에서는 캐쉬를 사용하고 있지 않다. 따라서 이동 호스트가 데이타 항목을 접근하려고 할 때에는 서버와 연결을 필요로 한다. 그리고 서버에서 발생되는 이동 트랜잭션의 일방적인 철회나 교착 상태로 인한 트랜잭션의 철회를 위해서는 부가적인 메시지 교환을 필요로 한다.

[12]에서는 캐쉬를 사용하는 환경에서 주기적인 방송으로 캐쉬내 일관성을 유지하는 세 가지 방법을 제안하였다. 서버를 자신의 셀에 있는 이동 호스트의 상태나 그 이동 호스트가 갖는 캐쉬내의 내용을 알고 있는 상태 서버(stateful server)와 자신의 셀 안에 있는 이동 호스트의 어떤 정보도 갖고 있지 않은 비상태 서버(stateless server)로 구분하였다. 그리고 서버에서 갱신된 항목들을 이동 호스트에 방송하는 무효화 방법을 동기적 방법과 비동기적 방법으로 구분하였다. 비동기적 방법은 서버에서 갱신이 발생하면 발생 즉시 이동 호스트에 알려주는 방법이고, 동기적 방법은 일정 시간 동안 모아진 갱신을 정기적으로 방송하는 방법이다. 또한 단순한 무효화 메시지대신 무효화 레포트(invalidation report)라는 용어를 사용하고 있는데, 이는 갱신되지 않은 데이타 항목에 대한 정보와 타임스탬프를 포함하고 있어서 단절된 이동 호스트가 연결될 때 캐쉬 내용 전체를 무효화하지 않는 방법을 제공한다. 따라서 무효화 레포트를 사용하면 이동 호스트가 연결된 후 그 다음의 방송 메시지를 기다려서 무효화할 항목을 선택함으로써 교환되는 메시지 양을 줄일 수 있도록 하였다. 이 연구에서는 비상태 서버가 동기적 방송 방법을 사용할 때 캐쉬 일관성을 유지하는 세 가지 알고리즘을 제안하였다. 서버에서 갱신된 데이타 항목과 그 항목에 대한 타임스탬프를 리스트로 유지하여 방송하는 TS방법과 갱신된 항목에 대한 정보만을 제공하는 AT방법, 그리고 이동 호스트가 오랜 시간 동안 단절되는 환경에 적합한 Signature 방법을 제안하였다. 이 연구에서 간과하고 있는 부분은 바로 트랜잭션의 직렬성 유지에 관한 부분이다. 주로 이동 호스트에서 데이타를 캐쉬하는 방법에 중점을 두고 있기 때문에 이 방법을 사용하면 다음 절의 예 1에서와 같은 직렬성 파괴의 경우가 발생한다.

[14]에서는 비교적 단절 시간과 갱신 패턴에 영향을 작게 받는 BS(Bit Sequence) 방법을 제안하였다. 이 방법에서 서버는 이동 호스트의 단절 시간을 고려하여 방송 구간 동안 갱신된 데이타 항목에 대한 정보를 단계적인 비트 열로 구성하여 방송하고 이동 호스트는 가장 적절한 비트 열을 수신하여 캐쉬 내용을 갱신할 수 있는 알고리즘을 제안하였다. 그러나 이 연구에서도 역시 트랜잭션의 직렬가능한 수행이 보장되지 못하고, 갱신되지 않은 항목에 대해서도 새로이 캐싱하는 거짓 무효화(false invalidation)가 발생한다.

캐싱 기법을 다룬 지금까지 제안된 대부분의 연구들에서는 캐쉬내의 일관성 유지에 초점을 두고 있다. 즉, 서버로부터 방송된 메시지를 수신하지 못하는 경우나 이동으로 발생하는 문제에 초점을 두고, 서버에 저장된 데이타 값과 캐쉬에 저장된 데이타 값의 일치 여부에 관심을 두고 있다. 하지만 트랜잭션의 직렬성이 유지되지 않고서는 올바른 데이타베이스 상태를 제공할 수 없기 때문에 이동 컴퓨팅 시스템에서 캐싱 기법은 트랜잭션의 직렬가능한 수행을 보장할 수 있어야 한다.

3. 캐싱 기법

이동 컴퓨팅 환경에서 이동 호스트는 반드시 캐쉬를 사용하는 것은 아니다. 하지만 무선 통신망에 할당되어 있는 제한된 대역폭을 효율적으로 사용하기 위하여 캐쉬는 중요한 장치이다. 지금까지 제안되어 온 캐싱 방법은 이동 호스트에서 데이타가 필요할 때마다 캐쉬함으로써 아래의 예 1과 같은 문제가 발생할 수 있다.

본 논문에서는 데이타의 캐싱 방법을 두 가지로 분류하였다. 이동 호스트에서 수행되는 이동 트랜잭션이 필요한 데이타를 미리 캐쉬한 후 수행을 시작하는 방법과 데이타가 필요할 때마다 캐쉬하는 방법이 그것이다. 이러한 캐싱 방법들은 갱신이 주기적으로 반영된다고 할 때, 이동 트랜잭션이 수행을 완료할 때까지 거치게 되는 방송 주기의 수에 따라 직렬성의 유지 여부가 달라진다. 따라서 이 장에서는 이동 환경에서 캐싱 기법을 사용할 때 이동 트랜잭션이 수행을 완료하기까지 거치게 되는 방송 주기의 수에 따라 직렬성 위배의 조건을 탐지하여 해결하는 방법을 제안하고자 한다.

다음의 예 1은 일반적인 캐싱 기법에서 이동 호스트가 필요할 때마다 데이타를 즉시 캐쉬하는 경우에 발생하는 직렬성 위배의 예이다.

[예 1] 즉시 캐싱으로 인한 트랜잭션 직렬성 위배의 예

T_1: $R_1(x)$ $R_1(y)$

T_2: $W_2(x)$ $W_2(y)$

예 1에서 T_1은 이동 호스트에서 수행되는 이동 트랜잭션으로, 이동 호스트의 캐쉬에는 x만 캐쉬되어 있고, y는 연산에 의해 접근될 때 즉시 캐쉬된다. T_2는 서버에서 수행되는 갱신 트랜잭션이다. 연산들이 $W_2(x)$ $R_1(x)$ $W_2(y)$ $R_1(y)$의 순서로 수행된다면, $R_1(x)$는 이미 캐쉬되어 있는 데이타를 읽으므로 데이타 항목 x에 대한 직렬화 순서는 $T_1 \rightarrow T_2$, $R_1(y)$는 T_2가 갱신한 값을 판독하므로 데이타 y에 대한 직렬화 순서는 $T_2 \rightarrow T_1$이 되어 트랜잭션 직렬성이 파괴됨을 알 수 있다. □

두 가지 캐싱 방법에 대하여, 이동 트랜잭션의 수행 시작 전에 필요한 모든 데이타를 얻는 캐싱 방법을 Cache-ahead 방법이라 하고, 필요할 때마다 서버에 요구하는 방법을 Cache-on-demand 방법이라 하자.

각각의 구체적인 캐싱 방법을 기술하기 전에 두 가지 캐싱 기법에서 공통적으로 사용하는 가정들을 살펴보면 다음과 같다. 첫째, 이동 트랜잭션은 판독 전용 트랜잭션이다. 이동 트랜잭션을 판독 전용 트랜잭션으로 제한하는 이유는 이동 호스트에서 갱신 연산이 수행되기 위해서 여러 사이트에 중복되어 있는 데이타에 모두 록을 설정하거나 완료 후 갱신 결과를 전파하기 위해 서버와의 잦은 연결을 요구하게 되는데, 무선 망의 낮은 속도와 신뢰성으로 인하여 시스템 성능이 저하될 수 있기 때문이다. 따라서 모든 갱신 연산은 서버에서 수행된다. 둘째, 이동 호스트에서는 한 번에 하나의 이동 트랜잭션만 수행된다. 셋째, 서버에서 수행된 갱신 트랜잭션의 결과는 주기적인 방송을 통하여 이동 호스트에 전달된다. 또한, 이동 호스트는 의도적으로 단절하지 않는 한 방송 메시지를 수신할 수 있다. 갱신의 결과가 주기적인 방송을 통하여 전달된다는 의미는 하나의 방송 구간동안 갱신된 값은 메시지로 구성되어 방송되어야 다른 이동 호스트들에서 접근할 수 있다는 의미이다.

캐쉬를 사용하는 환경에서 캐싱 기법이 달성해야 할 목표로는 두 가지가 있다. 하나는 캐쉬에 저장된 데이타와 서버에 저장된 데이타가 일관된 값을 갖도록 해야 한다는 것이고, 나머지 하나는 캐쉬를 접근하여 수행되는 트랜잭션의 직렬성이 유지되어야 한다는 것이다.

캐쉬에 저장된 데이타와 서버에 저장된 데이타의 일관성을 위하여, 서버에서 갱신 트랜잭션이 완료되면 갱신 결과는 다른 모든 호스트들에게 전달되어야 한다. 서버에서는 갱신 결과를 방송 메시지로 구성한 후, 이동 호스트들에게는 무선 망을 통한 주기적인 방송으로 갱신 결과를 알려주고, 고정 호스트들에게는 유선 망을 통해 전달한다. 갱신 결과를 갱신 트랜잭션이 완료될 때마다 이동 호스트들에게 전달하기 위해서는 갱신이 있을 때마다 서버에서 방송해야 하므로 메시지 전송이 많아지는 단점이 있다. 따라서 이동 환경에서 갱신된 데이타는 일정 시간 간격마다 주기적으로 방송된다. 서버에서 주기적으로 방송하는 메시지는 하나의 방송 구간동안 갱신된 데이타에 대한 정보로 구성된다. 캐쉬를 사용하여 수행되는 트랜잭션의 직렬가능한 수행을 보장하기 위한 방법은 3.1절과 3.2절에서 기술하고 있다.

3.1 Cache-ahead 방법

이 방법은 이동 호스트가 이동 트랜잭션을 수행하기에 앞서 접근하는 데이타 항목을 모두 자신의 캐쉬에 저장한 후 트랜잭션을 수행하는 방법이다. 트랜잭션 수행을 시작하기 전에 접근할 데이타를 한꺼번에

캐쉬하기 위해서는 접근할 데이타들을 미리 파악할 수 있어야 하는데, 이러한 과정은 선처리기(preprocessor)를 통해 수행될 수 있다.

이 방법에서 이동 트랜잭션이 수행을 완료하기까지 거치게 되는 방송 주기의 수에 따른 직렬성 유지 여부에 대하여 살펴보자. 먼저, 이동 트랜잭션이 하나의 방송 주기 안에 수행을 종료하는 경우에는 다음과 같은 스케줄을 고려할 수 있다.

① $R_i(x)$ $W_j(x)$ $W_j(y)$ $R_i(y)$

Cache-ahead 방법에서는 이동 트랜잭션이 수행을 시작하기에 앞서 모든 데이타들을 미리 캐쉬하므로, 위와 같은 히스토리에서 x가 캐쉬되는 시점과 y가 캐쉬되는 시점은 동일하다. 따라서 $R_i(y)$가 $W_j(y)$다음에 수행되기는 하지만 접근하는 데이타는 갱신 연산이 수행되기 전 값을 접근하므로 직렬가능한 수행이 가능하다.

② $W_j(x)$ $R_i(x)$ $R_i(y)$ $W_j(y)$

위와 같은 경우도 x가 캐쉬되는 시점에서 y가 함께 캐쉬되므로 갱신 연산의 영향없이 직렬가능한 수행을 할 수 있다.

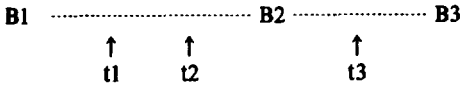
Cache-ahead 방법에서 이동 트랜잭션의 직렬성은 수행 시점이 아니라 데이타의 캐싱 시점에 영향을 받는다는 것을 알 수 있고, 또한 이동 트랜잭션이 하나의 방송 구간안에서 수행을 완료하는 경우에는 갱신 연산의 영향을 받지 않고 직렬가능하게 수행된다는 것을 알 수 있다.

다음으로, 이동 트랜잭션이 두 개 이상의 방송 주기에 걸쳐 수행되는 경우에는 아래 예 2와 같은 문제가 발생할 수 있다. 다음의 예 2를 보자.

[예 2] 데이타를 미리 캐쉬하였을 때 발생되는 직렬성 위배의 예

T_1: $R_1(x)$ $R_1(y)$

T_2: $W_2(x)$ $W_2(y)$ $Commit_2$



예 2에서 T_1은 이동 호스트에서 수행되는 이동 트랜잭션이고, T_2는 서버에서 수행되는 갱신 트랜잭션이다. B1, B2, B3은 서버에서 주기적으로 방송하는 방송 시점을 의미한다. 이동 트랜잭션은 수행에 앞서 자신이 접근하는 모든 데이타 항목을 캐쉬에 저장하고 수행을 시작하므로 예 2에서 데이타 x와 y가 캐쉬되어 있지 않았다면 t1시점에서 x, y가 모두 새로이 캐싱되며, 이미 캐쉬되어 있었다면 방송 메시지에 따라 일관된 값을 유지하게 된다. $R_1(x)$가 수행된 후, t2 시점에서 T_2가 완료되어 B2에서 갱신 결과가 방송 메시지에 포함되면, 이동 호스트는 캐쉬에 갖고 있던 y 값을 방송 메시지에 있는 값으로 갱신한다. $R_1(y)$가 수행될 때 접근하는 y값은 T_2에 의해 갱신된 값이므로, x에 대한 수행 순서는 $T_1 \rightarrow T_2$, y에 대한 수행 순서는 $T_2 \rightarrow T_1$이 되어 직렬성이 위배된다. □

예 2에서 이동 트랜잭션의 직렬성이 위배되는 원인은 동일한 이동 트랜잭션의 두 개의 판독 연산 중 하나의 연산($R_1(x)$)은 갱신 트랜잭션에 의해 데이타가 갱신되기 전에 캐쉬된 값을 접근하였고, 나머지 연산($R_1(y)$)은 갱신된 결과를 반영하여 수행되었기 때문이다. 예에서 $R_1(y)$가 B2전에 수행된다면 y값은 갱신 연산보다 먼저 캐쉬되었기 때문에 $W_2(y)$보다 선행하는 의미를 갖게 되어 충돌 관계에 사이클이 발생하지 않는다. Cache-ahead 방법에서는 하나의 이동 트랜잭션이 두 개 이상의 방송 주기동안 수행될 때 이미 캐쉬한 데이타 항목 중 일부가 방송 구간동안 갱신되어 방송 메시지에 포함되는 경우에 직렬성이 위배될 수 있다. 위 예에서는, 하나의 방송 구간동안 갱신 트랜잭션에 의해 갱신된 데이타(x, y)가 이동 트랜잭션이 방송 구간 전에 캐쉬하여 접근한 데이타(x)와 새로운 방송 구간이 시작된 다음에 접근할 데이타(y) 사이에서 교집합이 있기 때문에 직렬성이 위배되었다. 예 2에서 이동 트랜잭션이 $R_1(x)$와 $R_1(z)$로 구성되고, $R_1(y)$가 수행되는 시점에서 $R_1(z)$가 대신 수행된다면 직렬성이 위배되지 않는다. 직렬성 위배가 탐지되면 트랜잭션은 철회되어야 한다. 예 2와 같은 경우에 T_1은 방송 메시지에 의해 캐쉬에 있는 y를 접근하는 시점(t3)에서 철회된다.

Cache-ahead 방법을 사용할 때, 이동 트랜잭션의 직렬성 위배 탐지 알고리즘을 위해 필요한 자료 구조를 정의하면 다음과 같다.

- $Rset_{pre}$: 캐쉬한 데이타 항목 중 현재 방송 구간 이전에서 이미 접근한 데이타의 집합

- $Rset_{post}$: 캐쉬한 데이타 항목 중 현재 방송 구간에 접근할 데이타의 집합
- Wset: 방송 메시지에 포함된 갱신된 데이타 항목의 집합

Cache-ahead 알고리즘은 다음과 같다.

IF $((Rset_{pre} \cap Wset \neq \phi)$ *AND* $(Rset_{post} \cap Wset \neq \phi))$
THEN the MT is aborted;

(알고리즘 1) Cache-ahead 알고리즘

다음으로는 제안하는 Cache-ahead 알고리즘이 트랜잭션의 직렬가능한 수행을 보장함을 보이고자 한다.

【정리 1】 Cache-ahead 방법은 트랜잭션의 직렬가능한 수행을 보장한다.

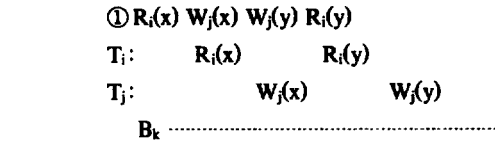
(증명) Cache-ahead 방법이 수행되는 트랜잭션의 직렬가능한 수행을 보장하지 못한다면 갱신 트랜잭션과의 충돌 관계에 사이클이 생기게 된다. 즉, $T_1 \rightarrow T_2 \rightarrow ... \rightarrow T_n \rightarrow T_1$과 같은 직렬화 순서가 존재하게 된다. 이러한 직렬화 순서는 이동 트랜잭션이 캐쉬한 데이타 항목 중 일부가 방송 메시지에 의해 갱신되었다는 의미로 이동 트랜잭션 T_1이 두 개 이상의 방송 주기동안 수행됨을 의미한다. T_1이 두 개 이상의 방송 주기 동안 수행되고, 방송 구간 이전에 캐쉬한 데이타 항목을 새로운 방송 메시지에 따라 갱신하게 되면 두 집합 Rsetpre와 Wset, Rsetpost와 Wset사이에서 각각 교집합이 발생하게 된다. 따라서 이러한 경우는 알고리즘에 의하여 T_1이 철회되므로 위와 같은 사이클은 만들어지지 않는다. 따라서 수행되는 트랜잭션의 직렬가능한 수행을 보장한다. □

3.2 Cache-on-demand 방법

이 방법은 이동 호스트가 필요할 때마다 서버에 데이타를 요청하여 캐쉬하는 방법이다.

트랜잭션 주기가 Cache-on-demand 방법에 미치는 영향에 대하여 살펴보면 다음과 같다. 먼저, 이동 트랜잭션이 하나의 방송 주기동안 수행되는 경우에는 Cache-ahead 방법과 마찬가지로 두 가지 경우를 고려할 수 있다.

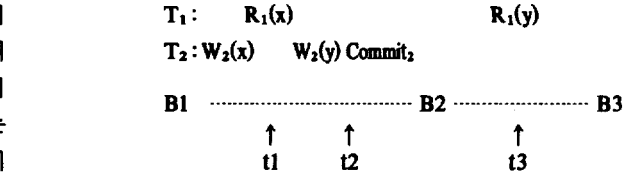
① $R_i(x)$ $W_j(x)$ $W_j(y)$ $R_i(y)$



Cache-on-demand 방법에서 이동 트랜잭션이 하나의 방송 주기 안에서 수행된다면, 이동 호스트에서 새로이 캐쉬할 데이타나 이미 캐쉬되어 있는 데이타는 Bk에서 유지되고 있는 값을 접근하게 된다. 왜냐하면 가정에 의해 갱신은 주기적인 방송 시점에서만 반영되기 때문이다. 이것은 이동 트랜잭션이 현재의 방송 구간동안 수행되는 어떤 갱신 트랜잭션의 영향도 받지 않는다는 것을 의미하게 되어 갱신 트랜잭션과의 수행 순서에 사이클을 갖는 경우는 발생하지 않는다. 또 다른 스케줄 ② $W_j(x)$ $R_i(x)$ $R_i(y)$ $W_j(y)$에 대해서도 마찬가지로 수행 순서에 사이클은 발생하지 않는다.

따라서 Cache-on-demand 방법에서 이동 트랜잭션이 하나의 방송 구간에서 수행을 완료하는 경우는 수행되는 트랜잭션의 직렬성이 위배되지 않는다. 하지만 아래와 같이 Cache-on-demand 방법에서 이동 트랜잭션이 두 개 이상의 방송 주기 동안 수행되면 직렬성이 위배되는 경우가 발생할 수 있다. 예 3을 보자.

[예 3] 데이타를 필요할 때마다 캐쉬하였을 때 발생하는 직렬성 위배의 예



예 3은 이동 호스트에서 수행되는 판독 전용 트랜잭션 T_1이 두 개의 방송 주기에 걸쳐 수행될 때 t1 이전에 이미 캐쉬에 갖고 있는 데이타 x와 t3에서 새로이 캐싱하는 데이타 y사이에서 직렬성이 파괴되는 예이다. 새로이 캐쉬하는 y값은 T_2에 의해 갱신된 값이기 때문이다. 따라서 이 경우도 예 1과 같은 수행 순서를 갖게 되어 직렬성이 파괴된다. □

Cache-on-demand 알고리즘에서 필요한 자료 구조를 정의하면 다음과 같다.

- $Rset_{pre}$: 캐쉬한 데이타 항목 중 현재 방송 구간 이전에서 이미 접근한 데이타의 집합
- $Rset_{miss}$: 캐쉬에 저장되어 있지 않아 서버로부터 새로이 캐쉬할 데이타 항목의 집합
  $Rset_{miss}$는 현재 방송 구간이 지나면 $Rset_{pre}$에 추가된다.
- Wset : 방송 메시지에 포함된 갱신된 데이타 항목의 집합

이러한 자료 구조를 이용하여 직렬성 위배를 탐지하는 방법은 알고리즘 2와 같다.

Whenever MH cache a data item y from a server
insert y into $Rset_{miss}$;
IF $((Rset_{pre} \cap Wset \neq \phi)$ *AND* $(Rset_{miss} \cap Wset \neq \phi))$
THEN abort the MT;
ELSE send data item y to the MH's cache;
IF (this time is the broadcasting time)
THEN for each data item x in $Rset_{miss}$
append $Rset_{pre}$;

(알고리즘 2) Cache-on-demand 알고리즘

다음으로는 제안하는 Cache-on-demand 방법이 트랜잭션의 직렬가능한 수행을 보장함을 보이고자 한다.

【정리 2】 Cache-on-demand 방법은 트랜잭션의 직렬가능한 수행을 보장한다.

(증명) 증명을 위해 Cache-on-demand 알고리즘이 트랜잭션의 직렬성을 위배한다고 가정하자. 트랜잭션의 직렬성이 위배된다면 이동 트랜잭션과 갱신 트랜잭션 사이의 충돌 관계에 사이클이 발생한다. 즉, $T_1 \rightarrow T_2 \rightarrow ... \rightarrow T_n \rightarrow T_1$과 같은 직렬화 순서가 존재한다. 이동 트랜잭션으로 만들어진 이와 같은 사이클은 이동 트랜잭션 T_1이 두 개 이상의 방송 구간에서 수행된다는 것을 의미한다. 왜냐하면 T_n과의 에지가 생성되기 위해서는 T_n이 갱신되어 완료된 값이 방송 메시지에 포함되어야 하기 때문이다. 이러한 경우는 $Rset_{pre}$와 Wset, $Rset_{miss}$와 Wset 사이에서 각각 교집합을 갖게 되고, 알고리즘에 의해 T_1이 철회되므로 위와 같은 사이클은 만들어지지 않는다. 따라서 수행되는 트랜잭션의 직렬성이 보장된다. □

3.3 간접 충돌

간접 충돌(indirect conflict)은 다음의 예 4에서 보는 것처럼 세 개 이상의 트랜잭션 사이에서 발생되는 충돌로 서로 다른 데이타 항목 상에 나타나는 직렬화 순서가 전체적으로 사이클을 이루어 트랜잭션의 직렬성이 파괴되는 경우이다. 따라서 이동 호스트가 데이타를 캐쉬하는 방법에 따라 이러한 간접 충돌을 해결할 수 있는 방법이 필요하다.

[예 4] 간접 충돌의 예

T_1 : $R_1(x)$ $R_1(z)$
T_2 : $W_2(x)$ $W_2(y)$ $Commit_2$
T_3 : $W_3(y)$ $W_3(z)$ $Commit_3$

예 4는 판독 전용 트랜잭션 T_1과 갱신 트랜잭션 T_2, T_3이 수행될 때 발생하는 간접 충돌의 예이다. 즉, 데이타 항목 x에 대한 직렬화 순서는 $T_1 \rightarrow T_2$, y에 대한 순서는 $T_2 \rightarrow T_3$, z에 대해서는 $T_3 \rightarrow T_1$이 되어 데이타 항목 각각에 대해서는 직렬가능하나 전체적으로 보면 충돌 관계에 사이클이 발생되어 직렬성이 파괴된다. □

앞 절에서 제안한 두 가지 캐싱 기법에 있어서 간접 충돌 문제를 살펴보면, Cache-ahead 방법과 Cache-on-demand 방법에서 T_1이 트랜잭션이 두 개 이상의 방송 주기에 걸쳐 수행되는 경우에 간접 충돌이 발생할 수 있다.

이러한 간접 충돌은 위에서 제안한 두 개의 캐싱 기법으로 해결이 가능하다. 두 가지 캐싱 방법에서 모두 이미 갱신된 데이타 항목(Wset)과 갱신하거나 ($Rset_{post}$)/새로이 캐쉬할 데이타 항목($Rset_{miss}$)에 교집합이 있으므로 탐지가 가능하다.

4. 동시성 제어 기법

지금까지는 서버에 저장된 데이타와 캐쉬에 저장된 데이타가 동일한 값을 유지하고, 이동 트랜잭션의 직렬성이 유지되는 두 가지 캐싱 기법을 제안하였다. 이 장에서는 이동 호스트가 캐쉬 일관성 유지를 위해

Cache-on-demand 방법을 사용할 때 록을 이용한 동시성 제어 기법에 대하여 제안하고자 한다. 이 방법을 CMC-MT (Concurrency control Method using Cache _lock for Mobile Transactions) 방법이라 하자.

제안하는 방법에서 사용하고 있는 가정은 다음과 같다. 첫째, 캐쉬 일관성 유지를 위해 Cache-on-demand 방법을 사용한다. 즉, 이동 호스트는 데이타가 필요할 때마다 서버에 요구하고 갱신은 방송 시점에 반영된다. 둘째, 록의 설정과 해제는 기본적으로 2PL(Two-Phase Locking) 방법을 사용한다. 셋째, 이동 호스트에서는 한 번에 하나의 이동 트랜잭션만 수행된다.

제안하는 방법에서는 이동 호스트가 서버로부터 데이타를 읽어갈 때 갱신 트랜잭션과의 직렬성 유지 여부를 판단하기 위해 필요한 새로운 록 모드를 도입하였다. 새로운 록 모드는 캐쉬 록으로 캐쉬 록이 설정되어 있을 때는 다른 록의 설정을 방해하지 않는다. 즉, 캐쉬 록은 다른 록 타입과 양립 가능하다. 이동 호스트는 서버로부터 데이타를 캐쉬할 때 서버에 캐쉬 록을 얻고 데이타를 캐쉬한다. 캐쉬 록이 설정되어 있는 데이타 항목을 갱신 트랜잭션이 갱신하고자 할 때에는 직렬성 검사를 위해 Pre_set을 유지한다.

임의 데이타 항목에 유지되는 Pre_set은 갱신 트랜잭션이 갱신 록을 요구할 때 그 데이타에 이미 캐쉬 록을 갖고 있는 트랜잭션이 있는 경우나 Pre_set이 공집합이 아닌 트랜잭션에 의해 해제된 록을 얻었을 때 트랜잭션 식별자(id)로 구성된다. Pre_set은 Pre_set안에 있는 트랜잭션이 완료될 때 초기화된다. 또한 Pre_set은 록킹 방법에서 발생할 수 있는 간접 충돌의 탐지 및 해결에도 사용된다. Pre_set의 관리 방법은 아래의 예 5에서 자세히 기술한다.

CMC-MT에서 사용되는 록 테이블은 〈표 1〉과 같다.

〈표 1〉 록 양립성 테이블
〈Table 1〉 Lock compatibility table

Hold \ Request	cl	rl	wl
cl	y	y	y
rl	y	y	n
wl	n	n	n

cl:cache_lock, rl:read_lock, wl:write_lock

[예 5] CMC-MT 방법 적용 예

T_1: $R_1(x)$ $R_1(y)$

T_2: $W_2(x)$ $W_2(y)$

이동 호스트의 캐쉬에 x와 y를 갖고 있지 않다고 가정하자. 이동 호스트는 $R_1(x)$를 수행하기 위해 x를 서버에 요청하면서 캐쉬 록을 요구한다. 예에서 이미 설정되어 있는 충돌 록이 없다고 가정하면 설정될 수 있다. 만약 기존에 설정되어 있는 록이 요구한 록과 양립 가능하지 않다면 그 록이 해제될 때까지 트랜잭션은 블록된다. 캐쉬 록이 설정된 후 서버에서는 $W_2(x)$의 수행을 위해 x에 대한 갱신 록을 요구한다. x에 설정되어 있는 록이 캐쉬 록이므로 갱신 록은 설정될 수 있다. 단, 여기서 갱신 록을 설정하기 전에 이미 캐쉬 록을 갖고 있는 트랜잭션을 $Pre_set(T_2)$에 삽입한다. 즉, $Pre_set(T_2)=\{T_1\}$이 된다. 다음으로 T_2가 y에 대한 갱신 록을 요청한다면 이미 설정되어 있는 충돌 록이 없다면 수여되고, 충돌 록이 이미 설정되어 있다면 블록된다. 록을 수여받았다고 가정하면 T_2는 성공적으로 완료할 수 있고, 갖고 있던 모든 록을 해제한다. 그 후 이동 호스트에서 데이타 항목 y에 대한 연산을 수행하기 위해 서버에 캐쉬을 요청할 때, 트랜잭션의 직렬성이 위배되는지 여부를 검사한다. 만약 T_1이 하나의 방송 주기안에 수행을 마치는 경우라면 새로이 캐쉬하는 y값은 T_2에 의해 갱신된 값이 아니라 가장 최근에 방송 메시지에 의해 마지막으로 갱신된 값이므로, 직렬가능한 순서를 만들 수 있다. 그러나 $R_1(y)$가 $R_1(x)$와 동일한 방송 구간에서 수행되지 못하는 경우에는 직렬가능한 수행이 불가능하다. 왜냐하면, y는 T_2에 의해 갱신된 값을 캐쉬하기 때문이다. 따라서 이러한 경우를 탐지하기 위하여 y를 캐쉬하는 시점에서 데이타 y를 갱신하여 완료된 트랜잭션 T_2의 Pre_set을 검사한다. $Pre_set(T_2)=\{T_1\}$이므로 직렬성이 위배됨을 탐지한다. Pre_set 검사로 직렬성 파괴가 탐지되면 서버는 요구한 데이타 대신 철회 결정을 전달한다. 이동 트랜잭션이나 갱신 트랜잭션이 종료되면 갖고 있던 모든 록은 해제된다. □

록킹 방법을 사용할 때 문제점 중 하나는 교착 상태의 발생이다. 제안하는 방법에서는 이동 호스트에서 수행되는 판독 전용 트랜잭션이 사용하는 캐쉬 록

은 갱신 록의 수여를 방해하지 않으므로 교착 상태를 발생시키지는 않는다. 따라서 교착 상태는 기존의 유선 환경에서 수행되는 판독 연산과 기록 연산 사이에서 발생할 수 있다. 이러한 교착 상태는 대기 그래프(WFG : Wait-For Graph)를 이용하여 탐지할 수 있다[2]. 또 하나의 문제는 간접 충돌의 발생이다. Pre_set을 사용하여 간접 충돌을 해결하는 방법은 다음과 같다. 예 4와 같은 간접 충돌이 있을 때 T_1이 z를 캐쉬하는 시점에서 유지되는 자료 구조는 다음과 같다.

x에 대하여, Pre_set(T_2)={T_1}
  /* T_1이 캐쉬 록을 수여 받은 후 갱신 록을 설정하였으므로 */
y에 대하여, Pre_set(T_3)={T2}
  /* Pre_set이 공집합이 아닌 T_2가 해제한 록을 수여받았으므로 */

먼저, T_1은 z를 갱신한 T_3의 Pre_set이 있는지 검사한다. 이 집합이 공집합이 아니라면 스택에 넣고 그 트랜잭션의 Pre_set이 있는지 검사한다. 결국, 체인으로 연결되어 있는 Pre_set에 T_1이 포함되어 있다면 간접 충돌로 인식하고 이동 트랜잭션을 철회한다.

제안하는 CMC-MT 알고리즘은 다음과 같다.

```
when a mobile transaction MT or update transaction T request a new lock
  case requested lock type
  read_lock: IF (there is no conflict with setting lock)
               THEN get a lock;
               ELSE the T is blocked;
             break;
  cache_lock: for requested data item x
          IF (there is any transaction Ti that update x)
            THEN for each transaction Tj in Pre_set(Ti)
              WHILE (Pre_set(Tj) is not empty)
                IF (Pre_set(Tj) inlcude MT)
                    THEN abort the MT;
                    ELSE Tj = Pre_set(Pre_set(Tj))
          ELSE IF (there is no conflict with setting lock)
                  THEN get a lock;
                       send x to the MH;
                  ELSE the MT is blocked;
             break;
  write_lock: IF (lock is set with only cache_lock)
                THEN for each transaction Ti that have the cache_lock
                     Pre_set(T) = Ti;
                     get a lock;
              ELSE IF (lock is released by a transaction Ti that
                       Pre_set is not empty)
                   THEN Pre_set(T) = Ti;
                   ELSE the update transaction is blocked;
             break;
```

(알고리즘 3) CMC-MT 알고리즘

다음으로는 제안하는 CMC-MT 알고리즘이 수행되는 트랜잭션의 직렬성을 보장함을 보인다.

【정리 3】 CMC-MT 알고리즘은 트랜잭션의 직렬성을 보장한다.

(증명) 증명을 위해 제안하는 알고리즘이 트랜잭션의 직렬성을 보장하지 못한다고 가정하자. 수행되는 트랜잭션의 직렬성을 보장하지 못한다면 n개의 트랜잭션 사이에서 $T_1 \rightarrow T_2 \rightarrow \ldots \rightarrow T_n \rightarrow T_1$과 같은 직렬화 순서가 존재하게 된다. 이동 트랜잭션이 판독 전용 연산만으로 구성되고, Cache-on-demand 방법을 사용할 때 이와 같은 사이클은 T_1이 이동 트랜잭션인 경우에 발생한다. T_1이 이동 트랜잭션이라면 T_1과 충돌 관계를 갖고 있는 T_2와 T_n은 갱신 트랜잭션임을 알 수 있다. 또한, 직렬화 순서에서 $T_n \rightarrow T_1$의 에지는 T_n이 갱신한 결과를 T_1이 접근한다는 것을 나타내며, 이렇게 되기 위해서는 T_n의 갱신 결과가 방송 시점에서 방송되어야 하므로, T_1은 두 개 이상의 방송 주기 동안 수행되었다는 것을 알 수 있다. 이와 같은 경우는 알고리즘에 의해 T_1이 데이타에 대한 캐쉬 록을 요구할 때 새로이 캐쉬하려는 데이타가 갱신 트랜잭션에 의해 갱신되었음을 인지하고, Pre_set 검사를 통해 그 갱신 트랜잭션의 Pre_set에 이동 트랜잭션 T_1이 포함되어 있다는 것을 알 수 있다. 이러한 결과에 의해 이동 트랜잭션은 요구한 데이타 대신 서버로부터 철회 결정을 전달받는다. 따라서 이동 트랜잭션 T_1이 철회되고 위와 같은 사이클은 만들어지지 않는다. 따라서 제안하는 방법은 트랜잭션의 직렬성을 보

장한다. □

5. 결론 및 앞으로의 연구 방향

본 연구에서는 이동 컴퓨팅 환경에서 이동 컴퓨터가 캐쉬를 사용할 때 수행되는 트랜잭션의 직렬성을 유지할 수 있는 두 가지 캐싱 기법을 제안하였다. 각 방법들에서 발생하는 트랜잭션 직렬성 위배가 발생하는 경우를 트랜잭션이 몇 개의 방송 주기동안 수행되는지에 따라 기술하였고, 직렬성 위배의 경우를 해결할 수 있는 알고리즘을 제안하였다. 그리고 일관성이 유지되는 캐싱 기법에 기반을 두고 록을 기반으로 한 동시성 제어 방법을 제안하였다.

제안한 두 개의 캐싱 기법은 적용되는 응용에 따라 그 성능이 달라질 수 있다. 첫 번째 방법인 Cache-ahead 방법은 이동 호스트가 트랜잭션의 시작 전에 접근하는 모든 데이타를 캐쉬하는 방법으로 트랜잭션이 하나의 방송 주기에 끝날 수 있다면 갱신 트랜잭션의 영향을 받지 않고 직렬성을 유지하며 효율적으로 수행될 수 있다. 두 번째 방법인 Cache-on-demand 방법은 이동 호스트가 필요할 때마다 데이타를 서버에 요구하는 방법으로, 갱신이 드문 환경에서 적합한 방법이다. 그리고 각 캐싱 기법에 대하여 세 개 이상의 트랜잭션 사이에서 발생할 수 있는 간접 충돌 문제 또한 제안하는 방법으로 해결할 수 있음을 보였다. 제안한 두 가지 캐싱 기법은 지금까지 제안되어 온 캐싱 기법들에 비하여 수행되는 이동 트랜잭션들이 일관성 있는 데이타베이스를 접근하도록 하기 위해 트랜잭션의 직렬성 유지에 초점을 두었다.

본 연구에서는 또한 Cache-on-demand 방법에 기반을 두고 이동 트랜잭션의 스케쥴에 록을 사용하는 동시성 제어 기법을 제안하였는데, 캐쉬 록을 사용함으로써 이동 트랜잭션이 캐쉬한 데이타와 갱신 트랜잭션이 갱신한 데이타 사이의 직렬성 유지 여부를 검사할 수 있게 하였다. 트랜잭션의 직렬성을 유지하기 위하여 필요한 자료 구조와 알고리즘을 제안하였다. 또한 제안하는 방법이 직렬성을 유지함을 보이기 위한 증명을 포함하고 있다.

앞으로 진행할 연구의 방향은 두 가지 캐싱 기법이 갖는 오버헤드를 해결할 수 있는 방법의 모색과 CMC-MT 알고리즘에 대해 록 관리를 좀 더 효율적으로 할 수 있는 방법에 대한 연구가 이루어져야 할 것으로 생각된다. 그리고 트랜잭션의 직렬성을 유지하기 위해서는 다중 버전을 제공하는 것도 하나의 해가 될 수 있는데, 이 방법에 대한 연구도 필요할 것으로 생각된다. 마지막으로 모의실험을 통한 알고리즘의 성능 평가도 남아 있는 작업의 하나이다.

참 고 문 헌

[1] Brian Marsh, Fred Douglis, Ramon Caceres, "*System Issues in Mobile Computing*", Technical Report Aachen University at Carnegie Mellon University Matsushita Information Technology Laborratory MITL-TR-50-93, Feb. 1993.

[2] Bernstein P. A, V Hadzilacos, "*Concurrency Control and Recovery for Database Systems*", Reading, Mass., Addison-Wesley, pp. 11-17, 1987.

[3] Margaret H. "*Mobile Computing and Databases: Anything new?*", SIGMOD Record, 1995.

[4] Pictoura E., B. Bhargava, "*Maintaining Consistency of Data in Mobile Distributed Environments*", In Proceedings of 15th International Conference on Distributed Computing System(ICDCS95), May 1995.

[5] Rafael Alonso, Henry F. Korth, "*Database System issues in Nomadic Computing*", ACM SIGMOD 1993.

[6] Tomasz Imielinski, "*Data Management for Mobile Computing*", SIGMOD Record, 1993.

[7] Tomasz Imielinski, B. R. Badrinath, "*Wireless Mobile Computing: Chanllenges in Data Management*", COMMUNICATIONS of the ACM, 1994.

[8] M. Satyanarayanan, James J. Kistler, Lily B. Mummert, "*Experience with Disconnected Operation in a Mobile Computing Environment*", CMU-CS-93-168, 1993.

[9] James J. Kistler, M. Satyanarayanan, "*Disconnected Operation in the Coda File System*", 1992, ACM

[10] Man Hon Wong, Wing Man Leungm, "*A Caching Policy to Support Read-only Transactions in a*

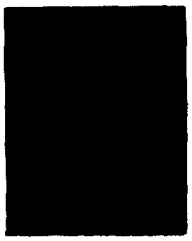
Mobile Computing Environment", CS-TR-95-07, May 1995.

[11] Jin Jing, Omran Bukhres, Ahmed Elmagarmid, "*Distributed Lock Management for Mobile Transactions*", Proceedings of the 15th IEEE International Conference on Distributed Computing Systems(ICDCS '95), May, 1995.

[12] Daniel Barbara, Tomasz Imielinski, "*Sleepers and Workaholics: Caching Strategies in Mobile Environments*", VLDB Journal, Dec., 1995.

[13] George Y. LIU, Gerald Q. Maguire, JR, "*A Mobility-Aware Dynamic Database Caching Scheme for Wireless Mobile Computing and Communications*", DATABASES AND MOBILE COMPUTING, Volume 4, No. 3, 1996.

[14] Jin Jing, Omran Bukjres, Ahmed Elmagarmid, "Bit-Sequences:A New Cache Invalidation Method in Mobile Environments", Technical Report CSD-TR-94-074, May, 1995.



김 치 연

1992년 2월 전남대학교 전산통계학과(학사)
1994년 2월 전남대학교 대학원 전산통계학과(이학석사)
1997년 2월 전남대학교 대학원 전산통계학과(박사과정수료)
1994년 3월~1996년 2월 순천 공업 전문대학 시간강사
1996년 3월~1997년 2월 전남대학교 전산학과 조교
1997년 8월~현재 전남대학교 전산학과 시간강사
관심분야:분산 시스템, 이동 컴퓨팅, 실시간 시스템

황 부 현

1978년 숭실대학교 전산학과 졸업(학사)
1980년 한국과학기술원 전산학과(공학석사)
1994년 한국과학기술원 전산학과(공학박사)
1981년~현재 전남대학교 전산학과 교수
관심분야:데이타베이스 보안, 이동 컴퓨팅, 트랜잭션 관리, 분산 처리 시스템 등