방송환경에서 이중 버전과 타임스탬프에 기반을 둔 낙관적 동시성 제어 기법

이 욱 현† · 황 부 현††

요 약

방송환경은 서버(server)와 클라이언트간 대역폭이 서버에서 클라이언트쪽으로는 크고 클라이언트에서 서버쪽으로의 대역폭은 상대적으로 많이 작은 비대칭적(asymmetric) 특수한 환경이다. 또한 대부분의 방송 환경 응용 시스템들은 클라이언트측에서 발생한 주로 주식 데이터, 교통 정보와 새로운 뉴스와 같은 여러 가지 다양한 정보를 검색하는 읽기전용 즉 질의 거래들을 허락한다. 그러나, 기존의 여러 가지 동시성 제어 기법들은 이러한 특수성을 고려하지 않음으로써 꽤 높은 데이터 경쟁 상태의 방송 환경에 적용될 때 성능 감소가 일어난다. 이 논문에서는 방송환경에서 가장 적절한 OCC/2VTS(Optimistic Concurrency Control based on 2-Version and TimeStamp)를 제안한다. OCC/2VTS는 캐쉬 내에 두 버전을 사용함으로써 타임스탬프 기법으로 클라이언트가 질의 거래를 자체 해결 할 수 있도록 하였다. 질의 거래 시작 후 2번의 무효화 방송을 통해 읽기 연산 대상 데이터 항목의 값이 바뀌지 않는다면 질의 거래가 갱신 거래의 완료와 상관없이 무사히 완료된다. 그 결과 첫째, 서버에게 완료 요구를 위해 정보를 보내는 기회가 감소하고 무효화 보고서 내에 갱신된 최신의 값을 포함하여 클라이언트들에게 방송함으로써 최근 데이터 값을 서버에게 요구하는 기회를 줄여 비대칭적 대역폭을 효율적으로 활용한다. 둘째, 질의 거래의 완료율을 최대한 높여 처리율을 향상시킨다.

Optimistic Concurrency Control based on 2-Version and TimeStamp for Broadcast Environment : OCC/2VTS

Uk-Hyun Lee† · Bu-Hyun Hwang††

ABSTRACT

The broadcast environment is asymmetric communication aspect that is typically much greater communication capacity available from server to clients than in the opposite direction. In addition, most of mobile computing systems only allow the generation of read-only transactions from mobile clients for retrieving different types of information such as stock data, traffic information and news updates. Since previous concurrency control protocols, however, do not consider such a particular characteristics, the performance degradation occurs when those schemes are applied to the broadcast environment having quite a high data contention. In this paper, we propose OCC/2VTS (Optimistic Concurrency Control based on 2-Version and TimeStamp) that is most appropriate for broadcast environment. OCC/2VTS lets each client process and commit query transactions for itself by using two version data in cache. If the values of appropriate data items are not changed twice by invalidation report after a query transaction starts, the query transaction is committed safely independent of commitment of update transactions. OCC/2VTS decreases the number of informing server for the purpose of commitment. Due to broadcasting the validation reports including updated recent values, it reduces the opportunity of requesting a recent data values of server as well. As a result, OCC/2VTS makes full use of the asymmetric bandwidth. It also improves transaction throughput by increasing the query transaction commit ratio as much as possible.

키워드 : 방송환경(BroadCast Environment), 이동 클라이언트/서버(Mobile Client/Server), 비대칭적 대역폭(Asymmetric Bandwidth), 질의 거래(Query Transaction), 낙관적 동시성 제어(Optimistic Concurrency Control), 두 버전(2-Version), 타임스탬프(TimeStamp)

1. 서 론

1.1 배경

무선 통신 기술의 발달은 이동 정보 서비스의 기능을 크게 증가시켰고 많은 이동 컴퓨팅 응용을 현실화시켰다. 이동 쇼핑 도우미와 같은 이동 정보 서비스들이 많은 쇼핑몰의 매출액 향상에 도움을 주었고 이동 휴대폰 또는 팜탑 컴퓨터를 통한 금융 정보 서비스가 이미 실현되었다. 사용자들은 어디에 있든 상관없이 이동 컴퓨터를 사용하여 정보에 즉각적인 접근을 요구하기 때문에 실시간 교통 정보, 항해 시스템 그리고 실시간 주식 정보 시스템과 같은 다양한 혁신적인 응용들도 계속해서 출현될 것이다.

※ 본 논문은 한국과학재단 2000년도 목적기초연구 지역대학 우수 과학자 지원 연구(2000-1-51200-005-2)비에 의하여 연구되었음.
† 정 회 원 : 전남대학교 대학원 전산학과
†† 정 회 원 : 전남대학교 전산학과 교수, 전남대학교 정보통신연구소
논문접수 : 2001년 1월 19일, 심사완료 : 2001년 5월 4일

이동 컴퓨팅 응용에서 가장 중요한 숙제 중의 하나는 이동 클라이언트(mobile Client)에 의해 제기된 거래들에게 어떻게 효율적으로 데이터 전달을 하는가이다. 제한된 대역폭, 이동 단말기의 전기 공급 능력 한계성과 무선통신망 자체의 불안정과 같은 이동 컴퓨팅 환경의 치명적인 제약점으로 인해 효율적이고 비용 효과적인 이동 컴퓨팅 시스템의 설계는 유선망 분산 시스템에서는 고려할 필요가 없는 해결하여야 할 많은 새로운 문제들을 가지고 있다[1, 17, 20-22]. 그 중 가장 중요한 것은 일관성 있는 데이터 전달 문제이다. 그러한 문제의 해결책은 무선통신망 대역폭의 한계와 질적으로 떨어지는 전송 때문에 더욱 난관에 부딪친다. 그럼에도 불구하고 최근에는 여러 데이터 전달 방법들이 이동 클라이언트들의 각기 다른 데이터 요구사항을 충족시키기 위해 제안되었다.

정보 서버(Server)로부터 이동 클라이언트들 거래로 데이터 항목을 전달하는데 기본적으로 두 가지 접근법이 있는데 그것은 요구(on-demand) 방식과 데이터 방송(data broadcast) 방식이다. 요구 방식은 거래에 의한 데이터 항목들의 요구가 있을 때에만 서버로부터 그 정보가 보내어진다. 이 접근법은 간단하나 많은 이동 클라이언트들이 있는 환경에서는 적절하지 못하다. 많은 거래들이 각기 다른 데이터 항목을 원한다면 데이터 항목을 기다리는 시간이 너무 길어질 것이다[23, 24].

그것과는 대조적으로 방송 접근법은 서버가 주기적으로 데이터 항목을 하나 하나 계속해서 이동 클라이언트들에게 방송하는 기법이다. 어떤 데이터 항목을 기다리는 거래들이 있다면 그 항목이 방송되어질 때 각 거래들은 무선 채널을 통해 해당 데이터 항목을 얻으면 될 것이다. 이와 같이 방송은 같은 데이터 항목에 대한 여러 요구사항을 만족시킬 수 있기 때문에 데이터를 전달하는데 드는 비용이 이동 클라이언트들의 수에 무관하다. 결과적으로 요구 방식보다 훨씬 더 대역폭을 효율적으로 사용할 수 있다. 그러므로 방송 기반 데이터 전파는 대역폭 효율성이 주요 관심사인 이동 컴퓨팅 및 무선 환경에서 상당한 양의 정보와 데이타를 많은 이동 클라이언트들에게 전달하는데 적절한 정보 전달 방법이다.

1.2 동기

앞에서 언급된 것처럼 많은 데이타베이스 응용분야에서 특히, 동시에 수행하는 엄청나게 많은 수의 클라이언트들을 가진 응용에서 데이터 전달을 위해 방송 기술이 요구되어진다. 예를 들어, 경매와 같은 전자상거래는 단지 소수에게 낙찰될지라도 수 만 명의 사용자들이 참여한다. 낙찰이 이루어질 때의 갱신 데이터는 신속하고 일관성 있게 전파되어져야 한다. 다행히 데이타베이스의 상대적으로 작은 부분 즉, 경매의 현재 상황과 같은 데이터만이 방송을 요구한다. 그러나, 클라이언트가 서버와 통신하기 위해 필요한 통신 대역폭이 상당히 제한되어 있다. 그러므로, 갱신에 대해 통신하도록 허락되어지는 시간에 클라이언트가 작은 대역폭의 무선망을 사용하여 경매의 현재 상태를 보내기 위해 방송 매체를 사용하는 것이 일반적이다. 즉, 방송환경은 서버와 클라이언트간 대역폭이 서버에서 클라이언트쪽으로는 크고 클라이언트에서 서버쪽으로의 대역폭은 상대적으로 많이 작은 비대칭적(Asymmetric)인 특수한 환경이다.

무선환경 및 이동 컴퓨팅 환경을 위한 여러 가지 동시성 제어 기법들이 국내외에서 제안되었다. 이전 연구 노력의 대부분이 주로 이동 컴퓨팅 환경은 통신 비용이 비싸고 통신자체가 안전하지 못하다는 특성을 갖는다는데 기반을 두어 방송환경의 특수성을 제대로 고려하지 않았다. 클라이언트가 자체에서 발생한 거래를 제어 완료하지 못하고 서버에게 상당히 많이 의존하는 형태라면 클라이언트측에서 서버측으로 정보 요구가 자주 생기게 된다. 이것은 각 클라이언트에게 주어진 짧은 시간 안에 상대적으로 작은 대역폭을 가지고 윗방향(uplink) 통신을 해야하는 방송환경에서는 거래 실행이 지연되어 전체 성능면에서 단위 시간당 처리율이 떨어진다. 그러므로 비대칭적 방송환경 특수성을 감안하여 클라이언트가 서버로의 윗방향 정보 요구는 가능하면 줄이는 거래 동시성 제어 기법이 필요하다.

또한 기존의 제안된 기법들은 대부분 질의 거래에 대해서는 특별히 고려하지 않았다. 대부분의 이동 컴퓨팅 시스템에서의 거래들은 주로 클라이언트측에서 발생되는 읽기전용(read_only) 거래들이다. 이러한 거래들은 주식 정보, 교통정보와 새로운 뉴스 등 여러 가지 다양한 종류의 정보를 요구 검색하는 것들이다. 읽기전용 거래는 읽기 연산으로만 이루어진 것으로 이후 질의(query) 거래라 칭한다. 실제적으로 날씨 예보와 교통 정보 시스템과 같은 대부분의 많은 방송 시스템에서 갱신 거래가 질의 거래보다 발생하는 확률이 훨씬 낮다. 이러한 실제 상황을 고려하더라도 질의 거래를 갱신 거래 방해 없이 즉, 질의 거래와 갱신 거래의 충돌연산으로 인한 철회 없이 질의 거래를 최대한 많이 실행 완료시키는 동시성 제어 기법이 연구되어야 한다.

1.3 목적

본 논문의 목적은 다음과 같다. 적당히 높은 데이타 경쟁 상태를 가정한 방송환경에서 동시성 제어를 위해 클라이언트측 캐쉬 버전의 이중화와 함께 낙관적 거래 스케쥴링을 사용할 때 그 환경의 특수성을 극복하고 성능 향상에 기여할 수 있는 효율적인 동시성 제어 기법을 제안하고자 하는 것이다.

앞에서 살펴본 바와 같이 기존의 이동 컴퓨팅 환경에서 제안된 동시성 제어 기법은 방송환경에 적용될 때 많은 질의 거래들의 철회와 윗방향 통신량이 많음으로 인해 최대한의 성능을 발휘할 수 없다. 그러므로, 방송환경을 위한 새로운 낙관적 동시성 제어 기법을 제안하는데 있어 본 논문은 통신 대역폭의 비대칭적 특수성을 감안해 윗방향 정보 송

신을 최소화하는 방법을 찾는데 초점을 둔다. 또한, 실제 방송 시스템의 거래 형태를 고려하여 질의 거래의 완료율을 최대한 높여 거래 처리율을 향상시키는 방법을 고안하고자 한다.

또한, 본 논문에서는 기존의 이동 컴퓨팅 환경에서 제안된 동시성 제어 기법과 다르게 클라이언트/서버 간 분산환경의 본연의 취지를 살리고자 주요 역할을 클라이언트에게 최대한 전담시키는 클라이언트 역할 최대화에 기본 철학을 둔다. 클라이언트측 활용은 방송환경과 기존의 분산환경을 연관시키는데 필요한 개념 설정을 쉽게 해 주며, 방송환경의 특수성을 감안할 때 비대칭적 대역폭을 최대한 활용할 수 있는 방법을 제공해 준다.

본 논문에서는 비대칭적 방송환경을 고려하여 데이터 일관성을 효율적으로 성취할 수 있으며 거래 동시성과 처리율을 최대한 향상시키는 동시성 제어 기법을 연구 제안하고자 한다.

1.4 논문의 구성

본 논문의 구성은 다음과 같다. 제2장에서는 방송환경에서 제안된 기존의 논문들을 살펴보고, 제3장에서는 성능 향상을 위해 제안된 기존의 동시성 제어 기법 OCC-UTS를 자세히 소개하고 방송환경 본연의 특성을 살려 적용할 때 나타나는 문제점에 대한 사례를 제시한다. 제4장에서는 방송환경의 특수성을 최대한 극복하고 동시성을 향상시킨 새로운 동시성 제어 기법을 제안한다. 제5장에서는 본 논문에서 제안한 동시성 제어 기법의 정확성을 검증하고, 제6장의 성능 평가에서는 제안된 기법과 기존의 기법을 분석하고 성능을 비교한다. 결론과 향후연구는 제7장에 나타나 있다.

2. 관련연구

방송환경에서 데이터 전달 방법에 관한 많은 연구가 국내외에서 있었다. Push 기법과 Pull 기법 그리고 이 두 가지를 혼합한 기법에 대한 성능 비교 분석이 이루어졌다[1]. 그리고 Broadcast Disk 즉, 무선 채널을 통해 방송되는 데이터 구조에 관한 많은 연구가 있었다. 데이터의 특성에 따라 다른 속도를 가지고 주기적으로 회전하는 Broadcast Disk를 제안하고 그의 성능 분석을 하였다[2]. 쓰기 연산이 이루어진 데이터를 Broadcast Disk를 통해 클라이언트들에게 갱신 결과를 알리는 여러 가지 방법, 즉 validation, auto-prefetch, propagation 기법을 비교하는 연구가 이루어졌다[3].

또한, 방송환경에서 동시성 제어에 관련된 연구 논문들이 국내외에서 발표되었으나 많은 논문들이 거래 개념을 도입하지 않았다. [5]의 논문에서는 이동 클라이언트/서버 컴퓨팅 환경에서 데이터 캐싱과 방송을 기반으로 한 방법을 처음으로 제안했다. 이 방법에서는 서버가 변화된 데이터 항목을 알리는 무효화(invalidation) 메세지를 주기적으로 방송하면 이동 클라이언트는 무선 채널을 통해 이 무효화 내용을 듣는다. 이 논문에 따르면, 이전 연구 노력의 대부분이 방송 보고서(broadcast report) 구조의 최적화[8], 그룹기반 캐쉬 무효화 전략[9], 그리고 분산된 캐싱 사용[7]과 같은 성능 분야에 초점을 두었다. 그것들은 주로 이동 컴퓨팅 환경이 통신 비용은 비싸고 통신자체가 안전하지 못한 특성을 갖는다는데 기반을 둔다. 그러나, 거래 개념과 거래 실행을 위한 동기화 문제가 이들 논문에서는 다루어지지 않았다.

그 외 거래 개념을 도입한 연구 논문들도 발표되었으나 이러한 논문들은 캐싱 기술을 고려하지 않거나 방송 기법의 효율성을 살리지 못했다는 문제점이 있다. [11]에서는 이동 클라이언트/서버 컴퓨팅 환경에서 거래들의 직렬화(serialization) 검증 부분을 이동 클라이언트에게 일부분 허락하는 방송 기술을 사용하여 거래들이 실행되어진다. 이 접근법의 초점은 방송 기술의 사용과 검증 작업을 이동 클라이언트에게 부분적으로 맡기는 것과 철회(abort)되어질 거래들을 일찍 알아낼 수 있다는 것이다. 그러나, 이 접근법은 이미 완료된 거래들과 충돌이 일어나는 현재의 거래들은 그것이 어떤 거래일지라도 완료되어질 수 없다. 이 결과 매우 낮은 거래 처리율을 보였다. 또한, 이 방법은 이동 클라이언트 내 데이터 캐싱 기술을 도입하지 않았다. [12]에서는 이동 클라이언트/서버 데이타베이스시스템에서 거래 스케쥴링을 위한 새로운 유효화(validation) 기법이 제시되었다. 특히, 유효화 단계에서의 기다리는 시간(waiting time)이 제거되고 거래들의 완료 순서를 재배열함으로써 처리율이 향상되었다. 그러나, 이 기법의 성능은 알려지지 않았고 이동 클라이언트내 캐싱 기술을 쓰지 않았다. [13]에서는 거래 관리에 있어서 방송 기술이 push 기반 데이터 전달을 위해 채택되었다. 예를 들어, 서버는 클라이언트의 구체적인 요구없이 데이터를 반복적으로 방송한다. 그리고 이 논문에서는 읽기 전용 거래들의 일관성과 현재성을 보장하는 문제가 언급되어진다. 그러나, 이동 클라이언트내 캐싱 기술은 역시 고려되지 않았다. [14]에서는 직렬화 순서 그래프 검사에 기반을 둔 접근법을 사용하였으나 클라이언트들이 계속해서 방송에 귀 기울이고 있어야 함을 요구한다. 이러한 방법은 통신 단절과 같은 무선 환경의 특성으로 인해 많은 문제점이 노출된다.

그 후 방송 기법을 도입한 동시성 제어 논문도 국내에서 발표되었으나 질의 거래가 많은 방송환경의 특수성을 제대한 감안하지 않아 성능 감소의 문제점을 내포하고 있다. 즉, [15]에서는 이동 클라이언트/서버 컴퓨팅 환경에서 한 서버에 의한 무효화를 알리는 방송의 의미를 이용한 거래 캐쉬 데이터의 새로운 일관성 유지 규약이 제시되었다. 일관성 검사의 대부분의 역할을 이동 클라이언트에게 넘긴 방법으로 접근된 데이터에 대한 일관성 검사와 완료 규약이 캐쉬 무효화 처리과정의 일부분으로서 분산 형태로 실현되었다. [11]에 비해 거래 처리율이 향상되었고 이동 거래 처리에서 요구되는 무선 통신 횟수를 감소시켰다. 그러나, 이 논문은

질의 거래가 많은 방송환경에 적용될 때 갱신 거래와의 연산 충돌로 인해 많은 질의 거래들의 철회가 일어나며 그 결과 성능 감소가 나타난다. 자세한 내용은 제3장 문제제시에서 보여줄 것이다.

3. 문제제시

동시성 제어 기법 중 이동 클라이언트/서버 컴퓨팅 환경에서 서버에 의한 무효화를 알리는 방송의 의미를 이용한 거래 캐쉬 데이터의 일관성 유지 규약인 OCC-UTS(Optimistic Concurrency Control with Update TimeStamp)가 논문 [15]에서 제시되었다. 이 방법은 질의 거래가 많은 방송환경의 특수성을 고려하지 않음으로 인해 질의 거래들이 그들과 함께 수행되는 갱신 거래에 의해 봉쇄되어 전체 성능을 현격히 급감시키는 문제의 심각성을 보여주고 있다. 이러한 문제점이 3.1절에 자세히 나타나 있다.

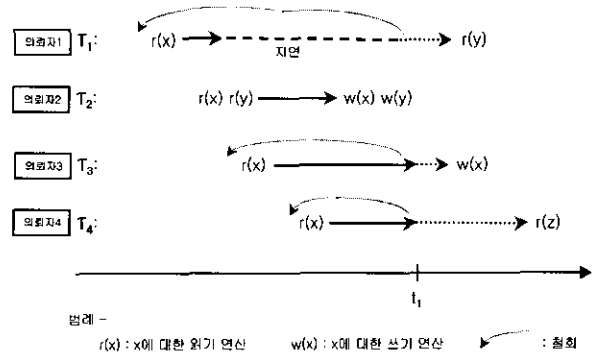
3.1 OCC-UTS를 질의 거래가 많은 환경에 적용했을 때의 문제점

OCC-UTS는 방송환경의 특수성을 충분히 이해하지 못하고 질의 거래에 대한 처리는 거의 고려하지 않았다. 그러므로 앞에서 언급했듯이 이동 클라이언트측 거래가 주로 질의 거래로 이루어진 방송 응용 시스템의 경우에 질의 거래의 철회율이 높아져 전체 거래 처리율이 많이 떨어진다. 제4장 성능평가에서 이것이 검증될 것이다.

이 기법은 각 이동 클라이언트는 한 거래를 완료하고자 할 때 그것이 질의 거래이든 갱신 거래이든 상관없이 거래 이름과 그 거래에 의해 읽혀진 데이터 항목들의 캐쉬 타임스탬프 및 그 거래에 의해 갱신된 데이터 항목들의 새로운 값을 완료 요구 메시지와 함께 서버에게 보낸다. 서버는 이 완료 요구 메시지들을 하나하나 처리하는데 각 거래에 의해 읽혀진 데이터 항목의 타임스탬프가 서버측 데이타베이스시스템에 있는 데이터 항목의 타임스탬프보다 작으면 그 거래를 철회리스트에 포함시킨다. 그렇지 않다면 완료리스트에 포함시킨 후 그 거래에 의해 갱신된 데이터 항목들의 새로운 값을 데이타베이스시스템에 반영하고 그것들의 타임스탬프 값을 갱신한다. 그 후 방송에 계속 귀 귀울이고 있는 이동 클라이언트는 무효화 보고서에 딸려온 완료리스트와 철회리스트를 살펴서 완료요구를 했던 거래이름이 들어가 있는지 조사한다. 어느 리스트에 속했냐에 따라서 이동클라이언트가 해당 거래를 처리한다. 또한, 각 지역 캐쉬내의 데이터 항목이 방송된 무효화 보고서에 속해 있다면 그것들의 타임스탬프를 비교해서 캐쉬된 데이터쪽이 작다면 해당 항목을 없애고 그렇지 않다면 타임스탬프를 방송 시점 타임스탬프로 바꾼다. 그리고 없애버린 데이터 항목 중 하나를 현재 진행되고 있는 어떤 거래가 읽었다면 그 거래는 바로 철회된다. 예 3.1에서 이러한 문제가 자세히 설명되고 있다.

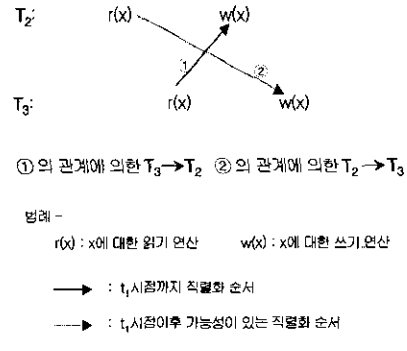
예 3.1 OCC-UTS를 질의 거래가 많은 시스템에 적용했을 경우의 봉쇄 :

질의 거래 T_1, T_4 그리고 갱신 거래 T_2, T_3가 방송환경의 각각의 이동 클라이언트측에서 실행하고 있다고 가정한다.(OCC-UTS의 시스템 모델에서는 한 시점에서 한 클라이언트내에 단지 한 거래만이 실행된다고 정했다.) T_1은 T_2보다 거래 시작이 먼저 된 후 무선환경의 단절 특성에 따른 실행 지연으로 완료가 늦어진 경우라 가정한다(그림 3.1).



(그림 3.1) OCC-UTS를 적용했을 경우 거래들의 철회

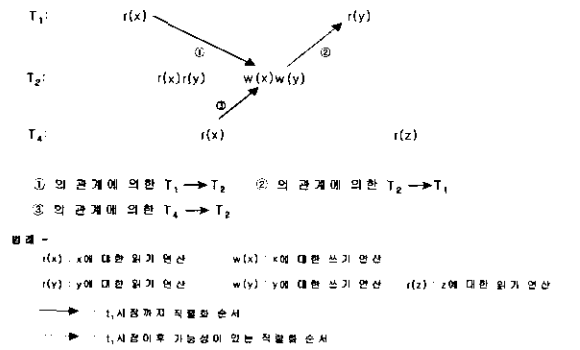
먼저 갱신 거래 T_2와 질의 거래 T_3간의 관계를 살펴보자. 시간 t_1에서 T_2가 완료되고 서버로부터 무효화 보고서가 방송되어지면 갱신거래 T_3는 t_1이후 T_2와 비직렬화의 가능성으로 인해 철회된다(그림 3.2).



(그림 3.2) 비직렬화 가능성의 T_2와 T_3의 거래 스케쥴

즉, t_1시점에서 방송된 무효화 보고서에 x가 포함되고 그 시점에서 실행 중이며 x를 이미 읽은 T_3는 비직렬화 가능성으로 인해 철회된다. 갱신거래 뿐만 아니라 질의 거래인 T_1과 T_4도 마찬가지이다(그림 3.3).

T_2보다 먼저 거래가 실행된 질의 거래 T_1은 T_2와의 사이에 $T_1 \rightarrow T_2$의 직렬 순서가 보장되어야 하나 완료가 지연



(그림 3.3) 비직렬화 가능성의 T_2와 T_1 및 T_2와 T_4 거래 스케줄

됨에 따라 t_1시점에서 x가 무효화되어지고 데이터 y로 인해 발생되는 $T_2 \rightarrow T_1$의 가능성으로 인해 그것이 단지 읽기 연산만으로 이루어진 질의 거래일지라도 철회된다. 더 문제가 되는 경우는 T_4와 같은 질의 거래인 경우인데 T_4의 r(x)와 T_2의 w(x)로 인한 $T_4 \rightarrow T_2$의 직렬 순서가 두 거래 완료까지 보장되어야 한다. 그러나, t_1시점에서 데이타 x를 포함하여 방송된 무효화 보고서에 의해 x를 이미 읽은 T_4는 그 뒤의 읽기 연산이 어떤 데이터를 읽든지 상관없이 즉, T_2가 갱신한 데이터 x와 y가 아닌 z를 읽어 $T_4 \rightarrow T_2$의 직렬 순서가 어긋나지 않음에도 불구하고 그 시점에서 실행중인 T_4는 철회된다. □

4. 방송환경에서 이중 버전과 타임스탬프에 기반을 둔 낙관적 동시성 제어 기법

새로이 제안하고자 하는 기법은 방송환경에서 캐쉬내에 이중 버전을 가지고 타임스탬프에 기반을 둔 낙관적 동시성 제어 기법(optimistic concurrency control based on 2-version and timestamp : OCC/2VTS)이라 칭한다.

이동 거래가 완료되면 거래에 의해 접근된 데이터 항목에 대한 거래 일치성이 보장되어야 한다. 이 논문에서 이동 갱신 거래들을 검증하기 위해 거래를 우선 수행하고 나중에 유효화 단계를 거치는 낙관적 동시성 제어 기법을 사용한다. 낙관적 기법이 이동 클라이언트들과 서버 사이에 주고 받는 메시지 수를 최소화 하기 때문이다. 낙관적 동시성 제어 기법을 사용할 때 서버는 매 거래 완료 시점에서 그 거래를 포함하는 실행이 직렬화를 만족하는지 아닌지를 검사할 필요가 있다. 직렬화를 만족하지 않으면 완료를 하려는 거래는 철회되고 그렇지 않으면 완료를 무사히 마친다. 낙관적 동시성 제어에서 거래를 유효화하는데 있어서 후방향과 선방향의 두 가지 유효화 과정 형태가 있다[25]. 후방향 유효화는 완료하려는 거래가 최근에 완료된 다른 거래에 의해 무효화되어지지 않는다는 것을 보장하는 것을 검사한다. 선방향 유효화는 완료하려는 거래가 다른 어떤 활동중인 거래와 충돌하지 않는다는 것을 보장하는 것을 검사한다.

이동 클라이언트/서버 컴퓨팅 환경에서 서버가 실행중인 이동 거래들에 대해 잘 모르는 상황이면 후방향 유효화 방법을 선택하는 것이 더 바람직하다[15]. 유효화 될 거래의 읽기연산 항목 집합이 이미 완료된 다른 거래의 쓰기연산 항목 집합과 비교되는 순수한 후방향 유효화 방법과는 달리 OCC/2VTS에서는 완료를 시도하는 이동 거래를 포함한 실행이 직렬화하는지 아닌지를 그 거래에 의해 읽혀진 각 캐쉬내 데이터 항목의 타임스탬프와 서버에서 유지된 마지막 갱신 거래 타임스탬프와 비교함으로써 검사한다. 유효화 과정 대부분을 클라이언트들에게 넘김으로써 유효화 과정이 진정으로 분산되어 행해질 수 있다. 또한, 타임스탬프를 사용하는 점을 효과적으로 이용하여 캐쉬내 두 버전을 유지하도록 함으로써 동시성을 향상시키고 즉, 읽기 거래의 데이터 일치성을 어느 수준까지 보장할 수 있고 버전 타임스탬프를 비교하는 과정에서 질의 거래의 완료나 철회 등의 처리를 이동 클라이언트 자체에서 행하는 타임스탬프 기법을 낙관적 기법과 함께 사용하고 있다.

OCC/2VTS는 직렬화를 정확성 검증기준으로 사용한다. 직렬화는 데이타베이스시스템에서 거래를 위해 가장 공통적으로 사용하는 정확성 검증기준이다. 이 기준은 동시에 수행하는 모든 거래들의 결과가 거래들이 직렬 순서로 수행되어진 것과 같다는 것을 말한다. 그리고 질의 거래에 대해서는 strict consistency[18]를 유지해야 한다. 즉, 동시에 수행되는 갱신 거래들의 완료 순서와 일치된 순서로 직렬 수행된 후의 결과 값을 각 질의 거래가 읽는다는 것이다. 그러므로, 이동 클라이언트들 측에서 수행중인 모든 질의 거래들이 일치하는 데이타베이스시스템을 보는지를 검사해야 한다. 더욱이 OCC/2VTS는 3장의 문제제시에서 살펴보았던 문제점들을 해결하기 위해, 즉 질의 거래의 철회율을 최대한 줄이기 위해 각 클라이언트의 캐쉬내에 각 항목마다 두 버전 데이터와 타임스탬프를 함께 유지하도록 한다. 그 결과, 한 회 방송주기의 무효화 작업이 있는 한 구 버전을 사용함으로써 일치하는 데이타베이스시스템을 거래가 보도록 하여 질의 거래의 철회를 상당히 줄일 수 있다.

OCC/2VTS의 시스템 모델은 다음과 같다. 이동 클라이언트의 사용자들은 읽기와 쓰기 연산으로 이루어진 이동 거래를 일으킴으로써 데이타베이스에 자주 접근할 것이다. 한 이동 클라이언트에 의해 어느 시각에 단지 한 거래만 실행할 수 있다고 가정한다. 즉, 클라이언트는 한 거래가 마쳐진 후에야 다른 거래를 일으킬 수 있다. 또한 각 이동 거래는 먼저 읽기 연산에 의해 읽혀진 데이터 항목들의 부분집합만을 갱신할 수 있다고 가정한다. 데이타베이스는 서버에 의해 저장되고 유지된 데이터 항목들의 집합이다. 갱신은 이동 클라이언트에 의해 발생되지만 궁극적으로 데이타베이스에 반영되어야 한다. 이 논문의 범위는 요구해서 얻어지는(pull_based) 데이터 전달로 한정짓는다. 클라이언트는 서버에게 데이터

를 요구하고 서버는 주로 클라이언트들에게 무효화 정보를 방송하기 위해 방송매체를 사용한다. 또한, 이동 거래 실행은 원자적(atomic)이다. 즉, 거래는 완료되거나 철회된다. 캐쉬 데이터 일치성을 보장하기 위해 서버는 주기적으로 무효화 보고서를 방송하고 모든 실행중인 이동 클라이언트들은 이 보고서를 기다리고 그 내용에 따라 그들의 캐쉬를 무효화시킨다. 캐쉬내 이중버전에 대한 접근은 한 데이터 항목마다 하나의 포인터가 존재하여 신버전과 구버전 사이의 이동이 자유자재로 가능하다.

서버는 완료된 거래들에 의해 최근 갱신된 데이터 항목들을 추적하고 매 L초마다 $ts_i = iL$인 시간에 무효화 보고서를 방송한다고 가정하자. 타임스탬프 ts_i시점에서 무효화 보고서는 현재의 타임스탬프 ts_i와 (j, t_j, v_j)의 리스트로 구성된다. 여기서 j는 데이터 항목, t_j는 $ts_i - \omega L \le t_j \le ts_i$($\omega$는 무효화 방송 윈도우)인 j의 마지막 갱신 타임스탬프이고 v_j는 j의 갱신된 값이다. 특히, 어떤 하나의 완료된 거래에 의해 갱신된 모든 데이터 항목들은 같은 타임스탬프를 갖는다. 또한, 무효화 보고서 $IR(ts_i)$를 구축하기 위해 서버는 다음과 같이 정의되는 갱신 리스트 $U(ts_i)$를 유지한다 :

$U(ts_i) = \{[j, t_j] \mid j$는 $ts_i - \omega L$과 ts_i 사이에 완료된 거래들에 의해 갱신된 데이터 항목이고 t_j는 $ts_i - \omega L \le t_j \le ts_i$인 j가 마지막 갱신된 타임스탬프$\}$

무효화 보고서를 받자마자 이동 클라이언트는 캐쉬내에 j를 유지해야할지 말지를 결정하기 위해 캐쉬 내에 있는 (j, t^c_j) (여기서 t^c_j는 캐쉬내 j에 대한 타임스탬프임) 리스트와 비교하여 캐쉬 내 데이터 타임스탬프 값이 작다면 즉, t^c_j가 t_j보다 작다면 캐쉬내 데이터 항목의 신버전을 구버전으로 이동시키고 신버전에는 보고서 내 새로운 값 v_j로 채운다. 즉, 해당 항목의 값을 무효화시킨다. t^c_j가 t_j보다 작지 않다면 j의 값은 그대로 유지되고 단지 타임스탬프 값만 현재의 타임스탬프 ts_i로 바뀐다. 이동 클라이언트는 또한 보고서를 받았던 마지막 시간을 가리키는 ts_{lb}를 유지한다. ts_{lb}는 이동 클라이언트가 통신 단절로부터 복구된 후 마지막 받은 보고서의 타임스탬프를 알 수 있도록 신뢰할만하게 유지된다.

이동 거래들의 직렬화를 성취하기 위한 이동 클라이언트 측 알고리즘은 다음과 같다. 각 이동 클라이언트들은 각 거래가 갱신 거래인지 질의 거래인지를 거래 수행 시작시점에서 안다고 가정한다. 질의 거래일 경우에 각 읽기 연산 대상 데이터 항목들의 캐쉬내 타임스탬프를 시작시점에 파악하여 그것들 중 가장 큰 값을 거래 완료 때까지 기억한다. 또한, 각 이동 클라이언트들은 갱신 거래를 위해 다음과 같은 두 가지 집합을 유지한다.

$ReadSet(T_{id}) = \{ [j, t^c_j] \mid j$는 갱신 거래 T_{id}에 의해 읽혀진 데이터 항목, t^c_j는 j의 캐쉬 타임스탬프$\}$

$NewValues(T_{id}) = \{ [j, v_j] \mid j$는 갱신 거래 T_{id}에 의해 갱신된 데이터 항목, v_j는 j의 새로운 값$\}$

각 무효화 리포트가 도착할 때 각 이동 클라이언트는 접근된 데이터에 대한 거래 일관성을 체크하고 <표 4.1>과 같은 알고리즘을 수행한다.

〈표 4.1〉 이동 거래를 처리하기 위한 클라이언트측 알고리즘

```
1. On receiving r_T(j){
     if j's recent version is in the cache {
          if T is query(read_only) transaction {
               if there is the value of version cache that is equal
               to or less than the preserved timestamp {
                    answer by the value ;
               }
               else { abort T; }
          }
          else { answer by using the recent value in its version cache;}
     }
     else {
          go uplink to the server and load j's recent version into
          its cache ;
          if T is query(read_only) transaction {
               if there is the value of version cache that is equal
               to or less than the preserved timestamp {
                    answer by the value ;
               }
               else { abort T; }
          }
          else { answer by using the recent value in its version cache; }
     }
  }
2. On receiving w_T(j) {
          throw j's past version in its cache ;
          copy j's recent version to j's past version in its cache;
          update the value of j's recent version of its cache;
     }
3. On receiving commit_T request {
          if T is update transaction {
               send a RequestCommit message, along with Tid,
               ReadSet and Newvalues ;
          }
          else {commit T; }
     }
4. On receiving the invalidation report IR(ts_i) {
          if T_id appears in CommitList { commit T;}
          if T_id identifier appears in AbortList { abort T; }
          if (ts_i - ts_lb > w_L) {
                    drop all past version in the entire cache ;
                    move all recent version to past version
          }
          else {
               for every item j's recent version in its cache {
                    if there is a pair (j, t_j, v_j) in IR(ts_i) {
                         if t^c_j < t_j {
                              copy j's recent version to past version ;
                              insert value v_j into j's recent version ;
                         }
                         else {t^c_j = ts_i ; }
                    }
               }
     }
     if the version of any data item was dropped from its cache {
          if there is an active update transaction T such that its
          ReadSet contains any dropped data item {
               abort T; }
          }
     }
```

읽기 연산을 수행하는 과정은 다음과 같다. 이동 클라이언트는 필요로 하는 데이터 항목의 신버전이 자체 캐쉬내에 있으면 즉시 캐쉬된 데이터를 사용하던지 없다면 서버에 요구하여 서버측 데이타베이스내의 해당 데이터 값과 타임스탬프 값을 얻어 구버전의 존재 유무와 상관없이 신버전 자리를 채운다. 특히, 질의 거래의 읽기 연산은 다음과 같이 처리된다. 질의 거래 읽기 연산에서는 거래 시작 시점에서 기억된 타임스탬프와 같은 타임스탬프를 가지거나 없다면 차선책으로 작은 타임스탬프를 가진 해당 데이터 항목의 버전이 존재하면 그 버전 값을 읽는다. 그러한 버전이 없다면 그 거래보다 먼저 완료된 갱신 거래의 해당 데이터 항목에 대한 쓰기 연산과 그 질의 거래의 읽기 연산의 충돌로 인한 비직렬화 가능성을 막기 위해 그 질의 거래는 철회된다. 마지막 읽기 연산까지 이러한 경우가 발생되지 않으면 그 질의 거래는 무사히 완료된다. 이와 같이 서버에게 어떤 완료요구 없이 각 클라이언트가 직렬성을 위배하지 않으며 질의 거래를 독립적으로 처리할 수 있도록 한다.

쓰기 연산을 수행하는 과정은 다음과 같다. 해당 데이터 항목의 신버전 값을 갱신하기 전에 기존 신버전 값을 구버전으로 옮긴다. 그리고 신버전의 타임스탬프 값도 갱신한다. 갱신 거래 T에 대한 완료를 해야할 때 거래의 구분자 T_{id} 및 $ReadSet(T_{id})$과 $Newvalues(T_{id})$와 함께 *RequestToCommit*메시지를 서버에게 보낸다.

그 후에 각 무효화 리포트가 도착할 때 갱신 거래 T에 대한 완료를 요구했던 이동 클라이언트는 접근된 데이터에 대한 거래 일관성을 체크하기 위해 $IR(ts_i)$ 즉, 무효화 보고서와 함께 실려온 *CommitList*와 *AbortList*를 주시한다. 그리고, 해당 T_{id}가 이 두 개의 리스트 중 어느 곳에 있는지를 검사한다. T_{id}가 *CommitList*에 있다면 T는 무사히 완료된다. 그러나, T_{id}가 *AbortList*에 나타나면 T는 철회되고 T가 갱신했던 캐쉬내 데이터 항목의 신버전의 값과 타임스탬프는 의미가 없으므로 제거한다. 무효화 보고서 내용에 따라 캐쉬내 해당 데이터 항목들을 무효화하는 과정은 앞서 시스템 모델에서 언급된 바와 같다. 또한, 각 클라이언트의 캐쉬내 무효화된 데이터 항목을 $ReadSet(T_{id})$에 포함하고 있는 실행 중인 이동 갱신 거래 T_{id}가 있다면 그 거래는 철회된다.

방송환경에서 통신 단절이 발생할 경우, 각 이동 클라이언트는 $ts_i - ts_{lb} > \omega L$와 같이 ωL보다 더 오래 단절된다면 그것의 거래를 철회한다. 즉, $ts_i - ts_{lb}$를 가지는 이동 거래는 무효화 방송 윈도우 ω값까지는 단절로 인한 문제를 극복할 수 있다.

서버가 처리해야 할 자세한 알고리즘은 <표 4.2>에 나타나 있다.

〈표 4.2〉 이동 갱신 거래를 처리하기 위한 서버측 알고리즘

```
1. dequeue a message m from Que in the interval [ts_{i-1}, ts_i ] {
   if there exists at least one j in m.ReadSet such that t^c_j < t_j {
          put m.T_id in the next AbortList report ;
   }
   else {
          put m.T_id in the next CommitList report ;
          commit the transaction in the server;
          (i.e., install the values in the m.Newvalues into the
          database)
          recompute U(ts_i) such that all data item in m.Newvalues
       has the same timestamp t_j and ts_{i-1} < t_j < ts_i ;
   }
   }
2. if it is time to broadcast IR(ts_i) {
          piggyback CommitList and AbortList with IR(ts_i) ;
          broadcast IR(ts_i) ;
   }
3. if it is asked for r_T(j) by a mobile client {
          broadcast the message containing (j's value, t_j) ;
   }
```

이동 거래들의 직렬성을 유지하기 위해 서버는 이동 클라이언트들로부터 *RequestToCommit*메시지들을 받아 큐를 유지한다. 각 *RequestToCommit*메시지 m은 $m.T_{id}$, $m.ReadSet$과 $m.Newvalues$와 같은 항목필드들을 갖는다.

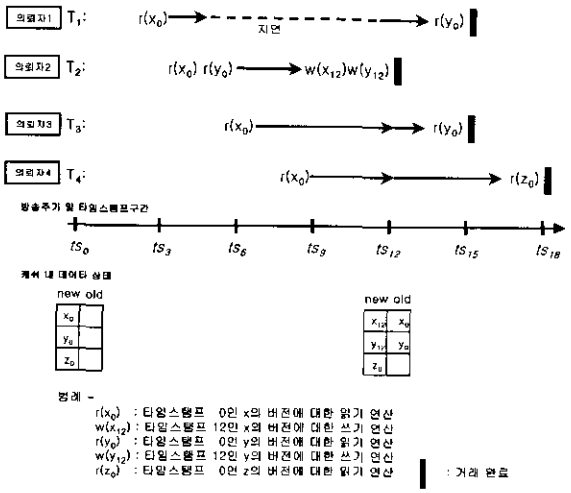
위의 정보를 가지고 서버는 $m.ReadSet$내 하나의 데이터 항목이라도 서버측 데이타베이스내 동일 데이터 항목의 타임스탬프보다 작다면 그 해당 거래는 *AbortList*에 등록된다. 그렇지 않으면 *CommitList*에 등록되고 $m.Newvalues$내의 값들을 데이타베이스에 반영하며 $m.Newvalues$내의 모든 데이터 항목들이 $ts_{i-1} < t_j < ts_i$인 같은 타임스탬프 t_j를 갖도록 $U(ts_i)$를 갱신한다.

각 방송 주기마다 무효화 보고서를 방송할 때 *CommitList*와 *AbortList*를 함께 실어 보낸다. 이동 클라이언트로부터 읽기 연산을 위한 데이터 항목 j에 대한 요구가 있을 경우에는 j의 값과 타임스탬프 t_j를 포함하는 메시지를 방송한다.

OCC/2VTS에서는 질의 거래 시작 후 2번의 무효화 방송을 통해 읽기 연산 대상 데이터 항목의 값이 바뀌지 않는다면 질의 거래가 갱신 거래의 완료와 상관없이 무사히 완료된다. 이러한 사항이 예 4.1에서 자세히 설명되어 있다.

예 4.1 OCC/2VTS를 질의 거래가 많은 시스템에 적용했을 경우의 완료 :

질의 거래 T_1, T_3와 T_4 그리고 갱신 거래 T_2가 이동 클라이언트측에서 각각 실행하고 있다고 가정한다. 또한, 매3초마다 무효화 보고서를 방송한다고 가정한다. 거래 T_1이 시작 후 시간 ts_{l2}시점까지 데이터 항목 y와 z의 방송을 통한 무효화 처리가 두 번 미만으로 발생한다고 가정한다. T_1은 T_2보다 거래 시작이 먼저 된 후 무선환경의 단절 특성에 따른 실행 지연으로 완료가 늦어진 경우라 가정한다(그림 4.1).



(그림 4.1) OCC/2VTS를 적용했을 경우 거래들의 완료

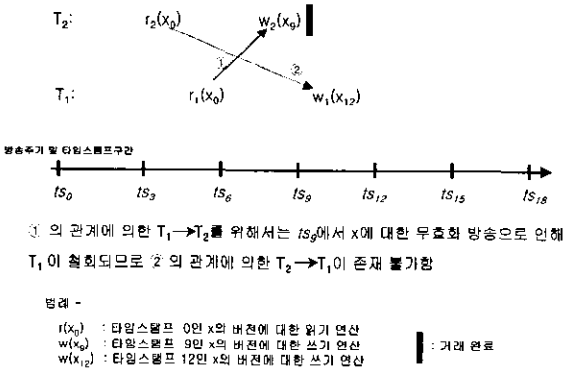
시간 ts_{12}에서 T_2가 완료된다. ts_{12}시점에서 무효화 방송에 의해 각 클라이언트의 캐쉬내 x와 y의 신버전 값은 구버전으로 옮겨진다. 그 결과, 먼저 수행을 시작한 T_1은 x를 읽은 후 ts_{12}시점에서 방송된 무효화 보고서의 내용 즉, x와 y를 무효화시키라는 내용에 영향을 받지 않고 데이터 항목 y를 읽어야 할 때 첫 읽기 연산 대상 데이터 항목 x의 버전 x_0의 타임스탬프 0와 같은 타임스탬프의 값을 가진 y의 버전 y_0가 존재하므로 그 버전 값을 읽고 무사히 완료된다. 결과적으로 $T_1 \rightarrow T_2$의 직렬 순서가 보장된다. 거래 T_3와 T_4도 마찬가지다. ts_{12}시점 전에 읽기 연산을 시작한 T_3와 T_4는 캐쉬내에 첫 번째 읽기 연산 대상 x의 신버전이 있는 상태이므로 타임스탬프 0인 x의 신버전 x_0를 읽고 난 후 y와 z를 읽어야 할 때 캐쉬내 타임스탬프 0인 버전 각각 y_0와 z_0가 존재하므로 무사히 읽기 연산을 한 다음 완료될 수 있다. 그러므로, $T_3 \rightarrow T_2$와 $T_4 \rightarrow T_2$의 직렬 순서가 거래 완료시점까지 보장된다. □

5. 정확성 검증

이동 클라이언트측 캐쉬는 클라이언트가 무효화 보고서를 처리한 직후 즉시 거래 일치 상태에 있다. 이것은 서버가 단지 직렬가능한 수행만 허락하고 무효화 보고서를 이용하여 거래적으로 일치하도록 함으로써 항상 일치성 있는 데이타베이스 상태를 유지한다. 이동 클라이언트에 의해 접근된 데이터에 대한 일치성 검사가 다음의 정리 5.1에 기반을 둔다.

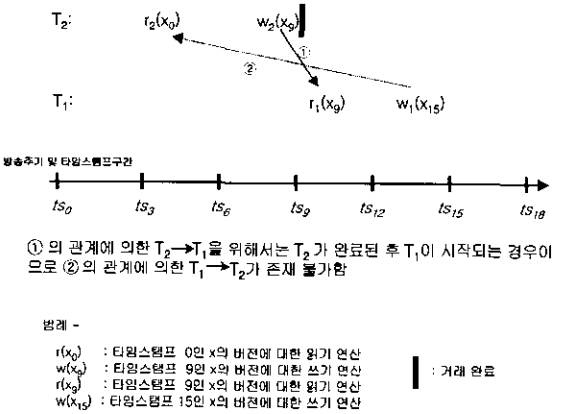
정리 5.1 이동 클라이언트에서 수행 중인 거래 T_1에 의해 접근된 어떤 데이터 항목도 무효화 보고서 $IR(ts_i)$를 처리한 후 무효화되지 않았다면 T_1을 포함한 실행은 ts_i시점까지 직렬가능하다.

증명 : T_1을 포함한 실행은 ts_i까지 직렬가능하지 않다고 가정하자. 이것은 T_1이 거래 시작과 ts_i사이 어느 시점에서 직렬화 그래프에 순환을 포함하고 있다는 것을 의미한다. 즉, T_1전에 ts_i시점에서 완료된 거래를 T_2라고 하면 $T_1 \rightarrow T_2 \rightarrow T_1$이 존재하여야 한다. 먼저 T_1, T_2가 갱신 거래일 경우를 고려해 보자. $IR(ts_i)$를 처리 후 분명히 T_1에 의해 접근된 어떤 데이터 항목도 무효화되지 않았다면 직렬화 그래프에서 충돌연산 $r_2(x)$와 $w_1(x)$로 인한 T_2에서 T_1로의 어떤 에지도 존재하지 않는다. (여기에서 T_2는 갱신 거래이므로 $r_2(x)$ 후 $w_2(x)$가 있을 것이고 T_1은 $w_1(x)$ 전에 $r_1(x)$가 있을 것이다.) 왜냐하면 T_2의 $r_2(x)$ 연산 후 $w_2(x)$연산 때문에 T_1의 x가 무효화되어야 하는데 이것은 가정과 모순이다(그림5.1).



(그림 5.1) T_1, T_2가 갱신 거래일 때 $T_1 \rightarrow T_2$일 경우 $T_2 \rightarrow T_1$은 존재하지 않음

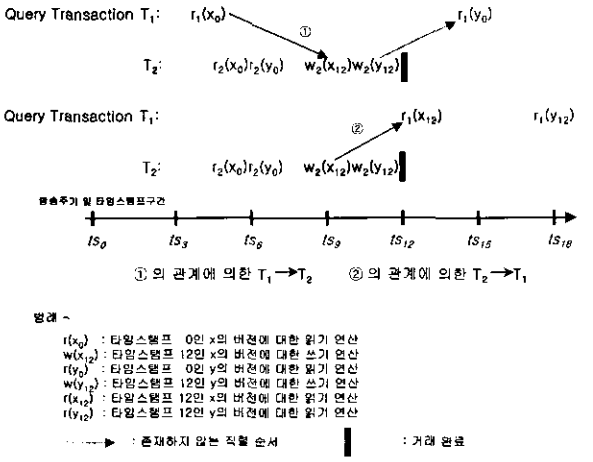
또한, 위 (그림 5.1)에서 보여주는 바와 같이 충돌 연산 $r_1(x)$와 $w_2(x)$로 인한 $T_1 \rightarrow T_2$일 경우에도 T_2에서 T_1로의 어떤 에지도 존재하지 않는다고 말할 수 있다.



(그림 5.2) T_1, T_2가 갱신 거래일 때 $T_2 \rightarrow T_1$일 경우 $T_1 \rightarrow T_2$가 존재하지 않음

위 (그림 5.2)에서 보여주는 것처럼 $T_2 \rightarrow T_1$이 가능한 경우는 T_2가 완료되어 데이타베이스에 x의 새로운 값이 반영되고 캐쉬내 무효화 작업이 이루어진 후 T_1의 $r_1(x)$가 시작하는 경우인데 이 경우는 $T_1 \rightarrow T_2$가 불가능하다.

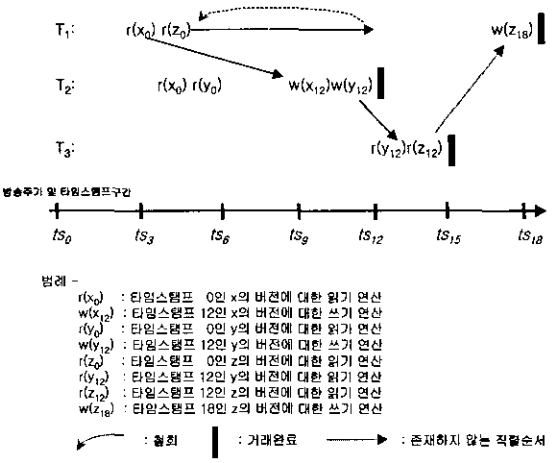
다음은 충돌 연산 $r_1(x)$와 $w_2(x)$ 경우의 T_1이 질의 거래일 때와 충돌 연산 $r_2(x)$와 $w_1(x)$ 경우의 T_2가 질의 거래일 때를 살펴보아도 결과는 마찬가지이다. 먼저 T_1이 질의 거래일 때 가능한 직렬화 순서를 (그림 5.3)에서 자세히 보여준다.



(그림 5.3) T_1이 질의 거래일 경우 가능한 직렬화 순서

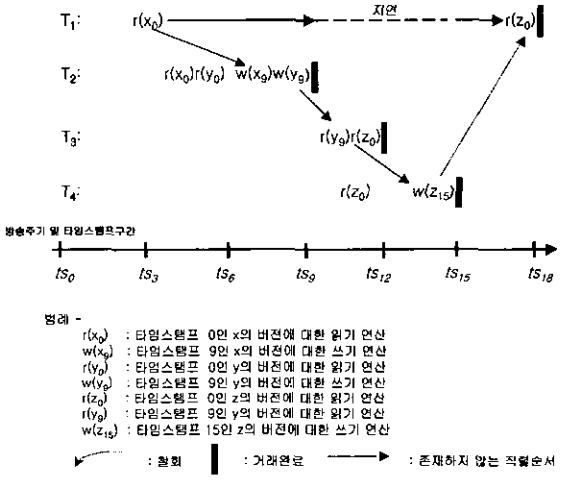
위의 (그림 5.3)의 첫 번째에서 직렬순서 $T_1 \rightarrow T_2$가 존재하는 경우에는 $T_2 \rightarrow T_1$는 존재하지 않는다. 왜냐하면 이것은 T_2가 완료되기 전에 T_1의 $r_1(x_0)$가 시작한 경우인데 지연으로 인해 T_1의 $r_1(y_0)$가 T_2가 완료된 후에 행해질지라도 거래 시작 전 기억된 타임스탬프 0(모든 거래 시작시점에서 캐쉬내 모든 데이터 항목은 타임스탬프 0을 가진 버전 값을 가진다고 가정할 때 T_1은 거래 시작 전에 대상 데이터 항목 x, y의 신버전 타임스탬프 중 가장 큰 값인 0을 기준 타임스탬프로 기억한다.)과 같은 타임스탬프를 가진 데이터 항목 y의 버전 값을 읽게 됨으로 T_2 완료 후 T_2의 결과와는 무관하게 질의 거래 T_1은 직렬 순서 $T_1 \rightarrow T_2$를 유지하면서 완료한다. 그러므로, 직렬순서 $T_2 \rightarrow T_1$는 존재할 수 없다. 위의 그림 두 번째 경우처럼 $T_2 \rightarrow T_1$이 가능하려면 T_2의 $w_2(x_{12})$가 끝나고 T_2가 완료된 후에 T_1의 $r_1(x_{12})$가 시작하는 때에만 가능하다. 즉, 이것은 $T_1 \rightarrow T_2$가 불가능하다. 그러므로 $T_2 \rightarrow T_1$인 경우에는 $T_1 \rightarrow T_2$는 존재할 수 없고 $T_1 \rightarrow T_2$인 경우에는 $T_2 \rightarrow T_1$은 존재하지 않는다. 또한, T_2가 질의 거래일 때도 T_1이 질의 거래일 때와 마찬가지이다. 결과적으로 어떤 경우일지라도 직렬화 그래프에서 $T_1 \rightarrow T_2 \rightarrow T_1$은 존재하지 않는다.

다음은 T_1전에 완료된 거래를 T_2, T_3, T_4라 하면 직렬화 그래프 상에서 다른 가능한 2가지 순환인 r, w, r, w연산의 충돌관계로 이루어지는 $T_1 \rightarrow T_2 \rightarrow T_3 \rightarrow T_1$ 또는 r, w, r, w, r 연산의 충돌관계로 이루어지는 $T_1 \rightarrow T_2 \rightarrow T_3 \rightarrow T_4 \rightarrow T_1$이 존재하여야 한다. 그러나, 먼저 $T_1 \rightarrow T_2 \rightarrow T_3 \rightarrow T_1$ 형태의 순환은 아래의 (그림 5.4)와 같이 존재하지 않는다.



(그림 5.4) $T_1 \rightarrow T_2 \rightarrow T_3 \rightarrow T_1$ 형태의 순환이 존재하지 않음

위의 (그림 5.4)에서 보는 것처럼 T_1의 $r(x_0)$와 T_2의 $w(x_{12})$로 인해 $T_1 \rightarrow T_2$, T_2의 $w(y_{12})$와 T_3의 $r(y_{12})$로 인해 $T_2 \rightarrow T_3$의 직렬 순서가 존재할 수 있다. 그러나 이 때 $T_3 \rightarrow T_1$은 존재하지 않는다. 왜냐하면, T_1은 갱신 거래이고 역시 갱신 거래인 T_2가 완료된 후 유효화 단계에서 T_2가 쓰기 연산 한 x를 먼저 읽은 T_1은 ts_{12}시점에서 철회된다. 만약 $T_2 \rightarrow T_3 \rightarrow T_1$ 직렬 순서가 존재한다면 그것은 T_1의 $r(x_0)$가 즉, T_1 거래의 시작이 T_2가 완료된 ts_{12}시점 이후 이루어진 것이며 $T_1 \rightarrow T_2$는 존재하지 않을 것이다. 또한, 다른 가능한 순환 형태 $T_1 \rightarrow T_2 \rightarrow T_3 \rightarrow T_4 \rightarrow T_1$도 아래의 (그림 5.5)와 같이 존재하지 않는다.



(그림 5.5) $T_1 \rightarrow T_2 \rightarrow T_3 \rightarrow T_4 \rightarrow T_1$ 형태의 순환이 존재하지 않음

위의 (그림 5.5)에서 보는 것처럼 T_1의 $r(x_0)$와 T_2의 $w(x_9)$로 인해 $T_1 \rightarrow T_2$, T_2의 $w(y_9)$와 T_3의 $r(y_9)$로 인해 $T_2 \rightarrow T_3$, T_3의 $r(z_0)$와 T_4의 $w(z_{15})$로 인해 $T_3 \rightarrow T_4$의 직렬 순서가 존재할 수 있다. 그러나 이 때 $T_4 \rightarrow T_1$은 존재하지 않는다. 왜냐하면, 질의 거래 T_1은 거래 시작 시점에서 기억된 타임스

탬프 0와 같거나 작은 타임스탬프를 가진 읽기 연산 대상 데이터 항목의 버전을 마지막 연산 때까지 읽기 때문에 T_4의 쓰기 연산과는 어떤 충돌도 일어나지 않는다. 만약 $T_2 \rightarrow T_3 \rightarrow T_4 \rightarrow T_1$의 직렬 순서가 존재한다면 T_4의 첫 읽기 연산이 T_3의 $w(z_{15})$가 끝나고 거래가 ts_{15}시점에서 완료된 후 시작되는 경우이므로 그 때는 $T_1 \rightarrow T_2$는 존재하지 않는다.

지금까지 살펴본 것처럼 T_1을 포함한 직렬화 그래프는 ts_i시점까지 순환이 없으며 이것은 전역적 직렬화 그래프에서 순환이 있다는 가정과 모순이다. 그러므로 현재 수행중인 T_1을 포함한 실행은 ts_i시점까지 직렬가능하다. □

정리 5.2 (OCC/2VTS의 정확성) : OCC/2VTS는 이동 거래들의 직렬가능한 수행을 보장한다.

증명 : 완료하려는 이동 거래를 T_n, 이미 완료된 거래를 T_1, T_2 등이라고 하고 우리는 n>1일 때 $T_n \rightarrow T_1 \rightarrow T_2 \rightarrow \cdots \rightarrow T_n$이 OCC/2VTS에 의한 직렬화 그래프에 존재하지 않음을 보일 것이다. 먼저 직렬가능하지 않아 그러한 순환이 있다고 가정하자. 즉, T_n이 아직 완료된 상태는 아니고 완료하려는 시점에서 T_n에서 이미 완료된 거래 T_1으로 에지가 다시 말해, $T_n \rightarrow T_1 \rightarrow \cdots \rightarrow T_n$이 있다는 것이다. 이것은 우리가 증명한 정리5.1에 대하여 모순이다. 완료하려는 각 이동 거래는 데이터 항목들의 가장 최근에 완료된 값만을 접근하기 때문에 완료될 각 이동 거래는 그 완료시점에서 직렬화 그래프상 그 거래에서 시작하는 어떤 에지도 갖지 않는다. 이것은 오직 완료된 이동 거래만을 포함하는 전역적 직렬화 그래프는 사이클을 포함하지 않음을 의미한다. 그러므로, OCC/2VTS는 직렬가능한 수행을 보장한다. □

6. 성능 평가

본 장에서는 제안한 OCC/2VTS의 성능을 분석하기 위한 분석 모델을 기술하고 방송환경에서 단위시간당 완료될 수 있는 거래들의 수인 처리량과 이동 클라이언트와 서버 사이에 주고받는 메시지 수인 메시지 비용에 따른 성능 분석을 한다. 그리고 제안한 OCC/2VTS의 성능 평가를 위해 [15]의 OCC-UTS를 비교 대상으로 한다. 우리의 성능 분석을 위한 가정은 다음과 같다.

- 데이타베이스시스템에 n개의 데이터 항목이 있고 각 거래는 a개의 데이터 항목을 접근한다.
- 데이터 접근은 동일하게 분산된다. (한 곳에 집중되는 가능성은 배제한다.)
- 데이터 항목들은 데이타베이스에서 항목당 μ갱신 비율의 지수 분포에 따라 갱신된다. 또한 우리는 동시에 수행중인 이동 거래의 수가 증가함에 따라 증가된 갱신 비율을 모델링하기 위해 가중치로서 동시 수행중인 이동 거래의 수를 사용한다. 그러므로, 데이터 항목당 갱신비율은 μ에 동시 수행중인 이동 거래의 수를 곱함으로써 계산된다.
- 각 이동 클라이언트는 반복적으로 데이터 항목들의 부분집합을 읽을 것이다. 이 부분집합에 있는 각 항목은 비율 λ으로 이동 클라이언트내에서 읽혀질 것이다.
- 캐쉬 데이터 일치성을 위해 서버는 매 L 초당 IR 즉, 무효화보고서를 방송한다.
- 이동 클라이언트들은 격자방(cell) 내에 있는 동안에 단절 된다. 즉, 사용자가 단말기의 전원을 off/on 할 때이다. 우리는 이러한 시간적 간격동안 단절될 가능성을 s로 연결될 가능성을 $1-s$라 가정한다.
- 이동 클라이언트들이 결코 단절되지 않는다면 각 거래가 모든 데이타베이스 연산을 수행하고 서버측 검증 과정을 들어가는데 M초가 걸린다. 우리의 분석 모델을 간략하게 하기 위해 각 이동 클라이언트가 무효화 보고서를 처리하자마자 거래를 시작한다고 가정한다.
- 이동 클라이언트들이 방송 주기동안 단절에서 복구되어 어떤 읽기 연산도 없을 확률 r_0는 $(1-s)e^{-\lambda L}$이고 한 주기동안 어떤 읽기 연산도 없을 확률 r_1은 $s+(1-s)e^{-\lambda L}$이다. 주기동안 데이타베이스에 어떤 갱신도 없을 확률 u_0는 $e^{-\mu L}$이고 주기동안 하나이상의 갱신이 있을 확률 u_1은 $1-e^{-\mu L}$이다.
- 이동 클라이언트에서 수행되는 거래 중 질의 거래일 확률을 q라고 하고 갱신 거래일 확률을 $1-q$라 한다.

6.1 처리율 분석

제안한 OCC/2VTS는 각 이동 클라이언트가 캐쉬내 두 버전 데이터를 가지며 서버는 완료를 시도하는 이동 거래를 포함한 실행이 직렬가능한지 아닌지를 그 거래에 의해 읽혀진 캐쉬내 데이터 타임스탬프와 서버측에서 관리되는 마지막 갱신 타임스탬프와 비교함으로써 검사한다. 이동 클라이언트측 캐쉬의 데이터 적중률을 h라 하자. 우리는 각 이동 클라이언트가 무효화 보고서를 처리한 직후 거래를 시작한다고 가정했기 때문에 지역 캐쉬 내에 있는 모든 무효화된 데이터 항목들은 구 버전으로 옮겨짐을 주목해야 한다. 이동 클라이언트가 a개의 항목들을 접근하고 완료하는 거래가 처리를 하는 동안 서버와 주고받는 메시지의 수는 $2a(1-h)+1$이다.

OCC/2VTS는 질의 거래일 경우 완료되기 전 해당 데이터 항목이 한 번 갱신되더라도 구 버전을 취함으로써 완료하도록 완료 가능성을 높였으므로 거래 완료 가능성을 계산하기 위해 먼저 거래가 질의 거래일 경우와 갱신 거래일 경우 두 경우를 나누어 계산한다. 또한, 이동 클라이언트들에 의해 접근된 a개의 항목들을 두 개의 그룹으로 나눈다. 캐쉬 내에서 유효한 ah 데이터 항목들과 캐쉬내에서 해결할 수 없는 $a(1-h)$ 데이터 항목들이다.

각 거래가 지역 캐쉬내 데이터를 가지고 모든 데이타베이스 연산을 수행하고 서버측 검증 과정으로 들어가는 데 걸리는 전체 시간을 E초라 하자. 이동 거래를 무사히 완료

하기 위한 조건은 다음과 같다. 캐쉬내 유효한 ah 데이터 항목들이 E초 동안 갱신되지 않아야 하고 캐쉬 내에 있지 않은 $a(1-h)$ 데이터 항목들은 이동 클라이언트가 그것들을 읽은 후로는 갱신되지 않아야 한다.

캐쉬내 유효한 ah 데이터 항목들이 E초동안 갱신되지 않을 확률 p_0는 다음과 같이 계산된다. 먼저 질의 거래에 대한 p_0'는 다음과 같다. OCC/2VTS는 각 클라이언트측 캐쉬가 두 버전 데이터로 이루어져 있으므로 거래가 시작된 후 2번 이상 방송주기 무효화 보고서로 인해 각 데이터 항목이 무효화되지 않는 한 이중 버전이 존재함으로써 데이터가 갱신되지 않을 확률이 버전을 고려하지 않을 때보다 2배가된다. 그러므로 아래와 같이 p_0'가 계산된다.

$$p_0' = e^{-\mu E/2} \times e^{-\mu E/2} \times \cdots \times e^{-\mu E/2} \text{ (전체적으로 } ah\text{배수)} = (e^{-\mu E/2})^{ah}$$

그리고 갱신 거래에 대한 p_0''는 다음과 같다.

$$p_0'' = e^{-\mu E} \times e^{-\mu E} \times \cdots \times e^{-\mu E} \text{ (전체적으로 } ah\text{배수)} = (e^{-\mu E})^{ah}$$

$a(1-h)$ 데이터 항목들이 이동 클라이언트가 그것들을 읽은 후에 갱신되지 않을 확률 p_1은 다음과 같이 계산된다.(각 데이터 항목을 처리하는 데 걸리는 시간(초)이 같다고 가정한다.)

$$p_1 = e^{-\mu E \frac{a(1-h)}{a(1-h)}} \times e^{-\mu E \frac{a(1-h)-1}{a(1-h)}} \times \cdots e^{-\mu E \frac{1}{a(1-h)}} \; e^{-\mu E \frac{a(1-h)+1}{2}}$$

질의 거래일 경우는 $a(1-h)$ 데이터 항목들이 갱신되지 않을 확률과는 무관하게 완료될 수 있으므로 이동 거래 완료 가능성은 다음과 같이 정리된다.

$$\begin{aligned} commit[\text{OCC/2VTS}] &= q \times p_0' + (1-q) \times (p_0'' \times p_1) \\ &= q \times (e^{-\mu E/2})^{ah} + (1-q) \\ &\quad \times (e^{-\mu Eah} \times e^{-\mu E\{a(1-h)+1)/2\}}) \end{aligned}$$

그리고 C를 동시 수행중인 거래 수라 할 때 처리율은 다음과 같이 계산된다.

$$Throughput[OCC/2VTS] = \frac{C \times Commit[OCC/2VTS]}{E}$$

OCC-UTS에서는 질의 거래의 처리율 향상을 따로 고려하지 않았기 때문에 질의 거래와 갱신 거래 두 경우를 따로 고려할 필요 없다. 그러므로, 캐쉬내 유효한 ah 데이터 항목들이 E초동안 갱신되지 않을 확률 p_0는 다음과 같이 계산된다.

$$p_0 = e^{-\mu E} \times e^{-\mu E} \times \cdots e^{-\mu E} \text{ (전체적으로 } ah\text{배수)}$$

그리고, $a(1-h)$ 데이터 항목들이 이동 클라이언트가 그것들을 읽은 후에 갱신되지 않을 확률 p_1은 다음과 같이 계산된다.(각 데이터 항목을 처리하는 데 걸리는 초가 같다고 가정한다.)

$$p_1 = e^{-\mu E \frac{a(1-h)}{a(1-h)}} \times e^{-\mu E \frac{a(1-h)-1}{a(1-h)}} \times \cdots e^{-\mu E \frac{1}{a(1-h)}} \; e^{-\mu E \frac{a(1-h)+1}{2}}$$

그러므로, OCC-UTS에서 이동 거래 완료 가능성은 다음과 같다 :

$$Commit[OCC/UTS] = P_0 \times P_1 = e^{-\mu E \frac{(a+ah+1)}{2}}$$

그리고 C를 동시 수행중인 거래 수라 할 때 처리율은 다음과 같이 계산된다.

$$Throughput[OCC/UTS] = \frac{C \times e^{-\mu E \frac{(a+ah+1)}{2}}}{E}$$

위의 식들에서 보는 것처럼 처리율 비교나 완료율 비교가 같은 의미이므로 우리는 두 비교 대상 기법의 완료율 비교만 살펴보겠다. 제안한 OCC/2VTS의 완료율 $Commit$[OCC-2VTS]는 OCC-UTS의 완료율 $Commit$[OCC-UTS]와 비교해 볼 때 $q \times (e^{-\mu E/2})^{ah} + (1-q) \times (e^{-\mu Eah} \times p_1) - e^{-\mu Eah} \times p_1$ 만큼 차이가 있다. (여기에서는 $a(1-h)$ 데이터 항목들이 갱신되지 않을 확률은 간략하게 p_1으로 표시한다.)

제안한 OCC/2VTS가 더 성능면에서 더 우수함을 보이려면 위의 식이 0보다 크면 되므로 다음과 같이 정리된다.

$$\begin{aligned} & q \times (e^{-\mu E/2})^{ah} + (1-q) \times e^{-\mu Eah} \times p_1 - e^{-\mu Eah} \times p_1 > 0 \\ \Leftrightarrow\ & q \times (e^{-\mu E/2})^{ah} + e^{-\mu Eah} \times p_1 - q \times (e^{-\mu Eah} \times p_1) - e^{-\mu Eah} \times p_1 > 0 \\ \Leftrightarrow\ & q \times ((e^{-\mu E/2})^{ah} - e^{-\mu Eah} \times p_1) > 0 \\ \Leftrightarrow\ & q > 0 \text{ and } (e^{-\mu E/2})^{ah} - e^{-\mu Eah} \times p_1 > 0 \\ \Leftrightarrow\ & q > 0 \text{ and } (e^{-\mu E/2})^{ah} - e^{-\mu Eah} \times e^{-\mu E\{a(1-h)+1)/2\}} > 0 \\ \Leftrightarrow\ & q > 0 \text{ and } (e^{-\mu E/2})^{ah} > e^{-\mu E\{(ah+a+1)/2\}} \\ \Leftrightarrow\ & q > 0 \text{ and } ah < ah + a + 1 \\ \Leftrightarrow\ & q > 0 \text{ and } a > -1 \end{aligned}$$

결과적으로 h값에 상관없이 q가 0보다 크면 즉, 질의 거래가 있다면 완료율이 더 향상됨을 보여주므로 처리율 향상이 기대된다.

6.2. 메시지수 분석

OCC/2VTS에서는 OCC-UTS와 달리 질의 거래 처리 과정이 다음과 같다. 질의 거래의 두 번째 이후 읽기 연산을 처리할 때 대상 데이터 항목이 있는지 없는지를 이동 클라이언트 캐쉬내 두 개의 버전 데이터 타임스탬프를 비교하

여 첫 번째 읽기 연산 대상 데이터 항목 타임스탬프와 같은 버전이 없으면 자체적으로 그 거래를 철회하고 그러한 경우가 거래 마지막 연산까지 발생하지 않으면 완료 처리한다. 즉, 클라이언트 내 질의 거래 처리 과정에서 완료요구 메시지를 서버에게 보낼 필요 없이 클라이언트가 거래를 자체적으로 해결하므로 전체 거래 실행 중 서버와 주고받는 메시지 수는 감소한다.

또한, 무효화 보고서 방송을 통한 무효화 과정에서 무효화될 데이터 항목의 새로운 값을 보고서 내에 포함시킴으로써 클라이언트측 거래의 읽기 연산시에 신버전을 서버에게 요구하는 윗방향 통신을 위한 메시지 수를 줄일 수 있다.

결과적으로, 비대칭적 대역폭을 효율적으로 사용함으로써 방송환경의 제약점을 보완함과 동시에 클라이언트 서버간 불필요한 통신 즉, 메시지 수를 줄임으로써 통신 비용이 비싸고 전송 상태가 좋지 않은 무선 환경의 제약점을 보완하였다.

7. 결론 및 향후 연구

본 논문에서는 방송환경에서 이동 클라이언트/서버 시스템을 위한 효율적인 동시성 제어 기법인 OCC/2VTS를 제안했다. 방송환경의 특수성을 고려한 OCC/2VTS는 기존의 이동 컴퓨팅 환경을 위한 동시성 제어 기법 OCC-UTS 또는 그 밖의 다른 기법과 비교될 때 다음과 같은 장점을 가지고 있다.

첫째, OCC/2VTS는 캐쉬내 이중 버전을 사용함으로써 질의 거래 처리시 타임스탬프 기법으로 클라이언트가 자체 해결을 할 수 있도록 하였다. 질의 거래 시작 후 2번의 무효화 방송을 통해 읽기 연산 대상 데이터 항목이 바뀌지 않는 한 즉, 첫번째 읽기 연산 대상 데이터 타임스탬프와 같은 두번째 이후 읽기 연산 대상 데이터 버전이 존재하는 한, 질의 거래가 동시 수행중인 갱신 거래의 완료와 상관없이 무사히 완료된다. 물론, 구 버전을 읽게 되어 거래들이 데이타의 최신의 값을 읽지 않게 될 수도 있다. 그러나, 그것은 질의 거래가 많은 환경에서 처리율 및 응답시간과 최신의 데이터 값을 읽는 것 중 어느 것을 중요시하는가에 달려있다. 게다가, 거래가 시작될 때 질의 거래인지 갱신 거래인지를 클라이언트가 인식하도록 하여 서버에 의한 전역적 거래 스케쥴링에서 질의 거래들은 거의 고려될 필요가 없기 때문에 OCC/2VTS는 거래 스케쥴링에 참여하는 거래의 수를 줄였고 거래 동시성 및 처리율을 향상시켰다.

둘째, OCC/2VTS는 클라이언트가 서버쪽으로 정보를 요구하는 메시지 수를 최대한 줄임으로써 비대칭적 대역폭을 가진 방송환경의 특수성을 효율적으로 잘 이용하였다. 위에서 언급한 것처럼 질의 거래를 클라이언트 내에서 자체 해결하도록 함으로써 서버에게 완료 요구를 위해 정보를 보내는 기회를 줄여 상대적으로 작은 대역폭을 가진 또한 한 클라이언트에게 허락되어지는 대역폭 사용 시간이 짧은 방송환경을 잘 극복할 수 있도록 하였다. 그리고 무효화 보고서 내에 무효화 될 데이터의 새로운 값을 포함시켜 방송하도록 함으로써 클라이언트측 거래가 읽기 연산시 신버전의 값이 없을 경우 클라이언트에게 요구하는 기회도 줄였다.

우리는 성능 평가를 통하여 OCC/2VTS의 성능의 우수성을 검증하고 이 기법이 방송환경을 위한 적절한 동시성 제어 기법임을 확인하였다. 처리율을 비교하기 위해 완료율을 계산해 본 결과 비교 대상인 OCC-UTS보다 질의 거래가 존재한다면 단위 시간당 거래 완료율이 커지고 결국 처리율이 향상됨을 검증하였다. 또한 메시지 수에 있어서도 무효화 보고서에 최신의 값을 포함하도록 하여 읽기 연산시 윗방향 정보 요구를 줄이고 질의 거래를 클라이언트가 자체적으로 해결하도록 하여 완료요구를 위한 서버쪽으로의 통신을 줄여 OCC-UTS보다 한 거래를 완료하는데 더 적은 수의 메시지가 필요함을 증명했다.

이것은 방송환경의 응용 시스템이 대부분 질의 거래로 이루어진다는 점을 감안할 때 거래 처리율 향상과 효율적인 대역폭 사용을 통해 OCC/2VTS가 기존의 어떤 다른 기법보다 방송환경에서 적합한 동시성 제어 기법임을 증명한다.

우리의 논문은 적절히 높은 데이타 경쟁상태의 방송환경에서 2번 이하 방송 주기의 무효화 보고서를 통해 캐쉬 내 데이터 항목의 값이 바뀌지 않은 한 즉, 캐쉬내 두 버전 데이타를 사용해서 질의 거래의 읽기 연산시 같은 타임스탬프 값의 데이터를 마지막 읽기 연산 때까지 읽게 함으로써 동시에 수행중인 거래들의 직렬성을 보장하였다. 이것은 캐쉬 내 두 버전을 사용하는 비용이 거래 처리율을 향상시키는 중요성보다 그리 크지 않다고 감안한 결과이다.

앞으로의 연구에서는 이러한 캐쉬 비용을 감안하면서 고도의 데이터 경쟁상태에서 여러 방송 주기 동안의 무효화 과정에서도 데이터 일관성을 보장 할 수 있는 다중 버전 제어 기법을 연구해 볼 것이고 서버들 상호간에 일관된 데이터를 관리하고 클라이언트들 상호간에도 일관된 데이터를 읽을 수 있도록 하는 상호 일관성과 클라이언트들이 방송 시작 주기 시점의 새로운 데이터 값을 항상 볼 수 있도록 보장하는 현재성을 만족시키는 동시성 제어 기법을 연구하고자 한다. 그것을 위해 데이터 검증기준을 직렬성이 아닌 update consistency[18]를 고려해봄으로써 방송환경에 좀 더 적합한 기법을 연구할 계획이다.

참 고 문 헌

[1] S. Acharya, M. Franklin and S. Zdonik, "Balancing Push and Pull for Data Broadcast," *Proceedings of ACM SIGMOD Conference on Management of Data*, May 1997.

[2] S. Acharya, R. Alonso. M. Franlikn and S. Zdonik, "Broad-

cast Disks : Data Management for Asymmetric Communication Environments," *Proceedings of ACM SIGMOD Conference on Management of Data*, May 1997.

[3] S. Acharya, M. Franklin and S. Zdonik, "Disseminating Updates on Broadcast Disks," *Proceedings of the 22nd VLDB Conference*, Mumbai, India, 1996.

[4] D. Aksoy, M. Franklin, "On-Demand Broadcast Scheduling," Technical Report, CS-TR-3854, University of Maryland, 1997.

[5] D. Barbara and T. Imielinsky, "Sleepers and Workholics : Caching in Mobile Environment," *Proceedings of ACM SIGMOD Conference on Management of Data*, pp.1-12, June 1994.

[6] J. Cay, K. Tan and B. C. Ooi, "On Incremental Cache Coherency Schemes in Mobile Computing Environments," *Proceedings of the 13th International Conference on Data Engineering*, pp.114-123, April 1997.

[7] C. F. Fong, C. S. Lui and M. H. Wong, "Quantifying Complexity and Performance Gains of Distributed Caching in a Wireless Network Environment," *Proceedings of the 13th International Conference on Data Engineering*, pp.104-113, April 1997.

[8] J. Jing, A. Elmagarmid, A. Helal and R. Alonso, "Bit-Sequences : An Adaptive Cache Invalidation Method in Mobile Client/Server Environment," *ACM/Baltzer Mobile Networks and Applications*, Vol.2, No.2, 1997.

[9] K.L.Wu, P.S.Yu and M.S.Chen, "Energy-efficient Caching for Wireless Mobile Computing," *Proceedings of the 12th International Conference on Data Engineering*, pp.336-343, Feb 1996.

[10] M. J. Franklin, M. J. Carey and M. Livny, "Transactional Client-Server Cache Consistency : Alternatives and Performance," *ACM Transactions on Database Systems*, Vol.22, No.3, pp.315-363, September 1997.

[11] D. Barbara, "Certification Reports : Supporting Transactions in Wireless Systems," *Proceedings of the 17th International Conference on Distributed Computing Systems*, pp. 466-473, May 1997.

[12] Y. Lee and S. Moon, "Commit-Reordering Validation Scheme for Transaction Scheduling in Client-Server Based Teleputing Systems : COREV," *Proceedings of International Conference on Information and Knowledge Management*, pp.59-66, 1997.

[13] E. Pitoura, "Supporting Read-Only Transactions in Wireless Broadcasting," *Proceedings of the 9th International Workshop on Database and Expert Systems Applications*, pp.428-433, 1998.

[14] E. Pitoura and P. Chrysanthis, "Scalable Processing of Read-Only Transactions in Broadcast Push," *International Conference on Distributed Computing Systems*, Austin, 1999.

[15] S. Lee, C. Hwang, W. Lee and H. Yu, "Caching and Concurrency Control in a Mobile Client/Server Computing Environment," 한국정보과학회 논문지, Vol.26, *No.8*, August 1999.

[16] J. Shanmugasundaram, A. Nithrakashyap, R. Sivasankaran, and K. Ramamritham, "Efficient Concurrency Control for Broadcast Environments," *ACM SIGMOD*, 1999.

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

[18] P. M. Bober and M. J. Carey. "Multiversion Query Locking," *Proceedings of the VLDB Conference*, Vancouver, Canada, August 1992.

[19] W. Weihl, "Distributed Version Management for Read-Only Actions," *IEEE Transactions on Software Engineering*, 13(1), January 1987.

[20] M. Franklin and S. Zdonik, "Data In Your Face : Push Technology in Prospective", in *Proceedings of 1998 ACM SIGMOD Conference*, Seattle, 1998.

[21] S. Hameed and N. H. Vaidya, "Efficient Algorithms for Scheduling Single and Multiple Channel Data Broadcast," Technical Report 97-002, Department of Computer Science, Texas, A&M University, Feb 1997.

[22] T. Imielinski and B. R. Badrinath, "Mobile Wireless Computing : Challenges in Data Management," *Communications of the ACM*, Vol.37, No.10, Oct. 1994.

[23] T. Imielinski, S. Viswanathan and B.R. Badrinath, "Data on Air : Organization and Access," *IEEE Transactions on Knowledge and Data Engineering*, Vol 9, No. 3, pp.353-372.

[24] H. V. Leong and Si, A, "Data broadcasting strategies over multiple unreliable wireless channels," in *Proceedings of the 4th International Conference on Information and Knowledge Management*, pp.96-104, November 1995.

[25] T. Harder, "Observations on Optimistic Concurrency Control Schemes," Information Systems, Vol.9, No.2, pp.111-120, 1984.

이 욱 현

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

1992년 이화여자대학교 전자계산학과 (이학사)

1997년 한국과학기술원 정보및통신공학과 (공학석사)

2000년~현재 전남대학교 전산학과 박사 과정

1992년~1997년 포스데이타 근무

1997년~2000년 광주은행 근무

1999년~현재 동강대학 초빙교수

관심분야 : 분산데이타베이스, 이동컴퓨팅, 방송응용시스템, 동시성 제어 등

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