방송환경에서 갱신 거래 우선 낙관적 동시성 제어 기법

이 욱 현[†] · 황 부 현[††]

요 약

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 방송 환경에 적용될 때 성능 감소가 일어난다. 이 논문에서는 방송환경에서 가장 적절한 OCC/UTF(Optimistic Concurrency Control with Update Transaction First)를 제안한다. OCC/UTF는 갱신 거래에 의해 무효화된 데이터를 먼저 읽기 연산한 질의 거래가 비직렬 가능성으로 인한 철회 없이 갱신된 새로운 값을 다시 읽는다. 그럼으로써 직렬 순서가 유지되어 갱신 거래의 완료와 상관없이 해당 질의 거래가 무사히 완료된다. 그 결과 첫째, 서버에게 질의 거래 완료 요구를 할 필요가 없으며 무효화 보고서 내에 갱신된 최신의 값을 포함하여 클라이언트들에게 방송함으로써 최근 데이터 값을 서버에게 요구하는 기회를 줄임과 동시에 서버는 새로운 값을 재방송할 필요가 없기 때문에 비대칭적 대역폭을 효율적으로 활용한다. 둘째, 질의 거래의 완료율을 최대한 높여 처리율을 향상시킬 수 있다.

Optimistic Concurrency Control with Update Transaction First for Broadcast Environment : OCC/UTF

Uk-hyun Lee[†] · Bu-hyun Hwang[††]

ABSTRACT

Most of mobile computing systems allow mostly read-only transactions from mobile clients for retrieving various 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 previous schemes are applied to the broadcast environment. In this paper, we propose OCC/UTF(Optimistic Concurrency Control with Update Transaction First) that is most appropriate for broadcast environment. OCC/UTF lets a query transaction, that has already read the data item which was invalidated by update transaction, read again the same data item without the abort of the query transaction due to non-serializability. Therefore, serializable order is maintained and the query transaction is committed safely regardless of commitment of update transactions. In OCC/UTF, Clients need not require server to commit their query transactions. Because of broadcasting the validation reports including values updated recently to clients, it reduces the overhead of requesting recent values from the server and the server need not also re-broadcast the newest values. As a result, OCC/UTF makes full use of the asymmetric bandwidth. It can also improve transaction throughput by increasing the commit ratio of query transactions as much as possible.

키워드 : 방송환경(broadcast environment), 이동클라이언트/서버(mobile client/server), 비대칭적 대역폭(asymmetric bandwidth), 질의 거래(query transaction), 낙관적 동시성 제어(optimistic concurrency control), 갱신 거래 우선(update transaction first)

1. 서 론

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 만명의 사용자들이 참여한다. 낙찰이 이루어질 때의 갱신 데이터는 방송 매체를 통해 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 보낸다. 즉, 방송환경은 서버와 클라이언트간 대역폭이 서버에서 클라이언트쪽으로는 크고 클라이언트에서 서버쪽으로의 대역폭은 상대적으로 많이 작은 비대칭적(Asymmetric)인 특수한 환경이다[1-7].

무선환경 및 이동 컴퓨팅 환경을 위한 여러 가지 동시성

※ 본 논문은 한국과학재단 2000년도 목적기초연구 지역대학 우수 과학자 지원 연구(2000-1-51200-005-2)비에 의하여 연구되었음.
† 정 회 원 : 인하대학교 강의전임교수
†† 정 회 원 : 전남대학교 전산학과 교수
논문접수 : 2001년 9월 6일, 심사완료 : 2002년 2월 2일

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, 교통 정보와 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.

본 논문의 목적은 다음과 같다. 낮은 데이타 경쟁 상태를 가정한 방송환경에서 동시성 제어를 위해 클라이언트측 데이터 캐싱과 타임스탬프에 기반한 낙관적 거래 스케쥴링을 사용할 때 그 환경의 특수성을 극복하고 성능 향상에 기여할 수 있는 효율적인 동시성 제어 기법을 제안하고자 한다. 우리의 기법은 (1) 상호 일관성과 (2) 현재성이라는 두 가지 특성을 만족한다. 상호 일관성은 (a) 서버가 상호적으로 일치하는 데이터를 유지하고 (b) 클라이언트들은 상호적으로 일치하는 데이터를 읽을 수 있다. 현재성은 클라이언트들이 항상 방송 주기 시작 시점의 최근 데이터를 보는 것을 보장한다.

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 송신을 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 현재성과 일관성을 효율적으로 성취할 수 있으며 거래 동시성과 처리율을 최대한 향상시키는 동시성 제어 기법을 연구 제안하고자 한다.

본 논문의 구성은 다음과 같다. 제 2장에서는 방송환경에서 제안된 기존의 논문들을 살펴본다. 또한, 성능 향상을 위해 제안된 기존의 동시성 제어 기법 OCC-UTS를 자세히 소개하고 방송환경 본연의 특성을 살려 적용할 때 나타나는 문제점에 대한 사례를 제시한다. 제 3장에서는 방송환경의 특수성을 최대한 극복하고 동시성을 향상시킨 새로운 낙관적 동시성 제어 기법을 제안한다. 제 4장에서는 본 논문에서 제안한 동시성 제어 기법의 정확성을 검증하고, 제 5장의 성능 평가에서는 제안된 기법과 기존의 기법을 분석하고 성능을 비교한다. 결론과 향후연구는 제 6장에 나타나 있다.

2. 관련 연구

방송환경에서 동시성 제어에 관련된 연구 논문들이 국내외에서 발표되었으나 많은 논문들이 실시간(real-time)의 쟁점이 없어 방송 기법의 효율성을 살리지 못했다는 문제점이 있다. [8]에서는 이동 클라이언트/서버 컴퓨팅 환경에서 거래들의 직렬화(serializability)[9] 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. [10]에서는 거래 관리에 있어서 방송 기술이 push 기반 데이터 전달을 위해 채택되었다. 예를 들어, 서버는 클라이언트의 구체적인 요구 없이 각 데이터 항목의 여러 버전을 버전넘버와 함께 반복적으로 방송한다. 그리고 이 논문에서는 읽기 전용 거래들의 일관성과 현재성을 보장하는 문제가 언급되어진다. 그러나, 이 방법은 상당한 방송 주기 횟수 증가와 그럼으로써 상당한 응답 시간(response time)의 증가를 가져온다. 이것은 매우 제한적이며 융통성이 부족하다. [11]에서는 직렬화 순서 그래프 검사에 기반을 둔 접근법을 사용하였으나 클라

이언트들이 계속해서 방송에 귀 기울이고 있어야 함을 요구한다. 이러한 방법은 통신 단절과 같은 무선 환경의 특성으로 인해 많은 문제점이 노출된다.

그 후 방송 기법을 도입한 OCC-UTS(Optimistic Concurrency Control with Update TimeStamp)[12]도 국내에서 발표되었으나 질의 거래가 많은 방송환경의 특수성을 제대한 감안하지 않아 성능 감소의 문제점을 내포하고 있다. [8]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. 이러한 문제점이 2.1절에 자세히 나타나 있다.

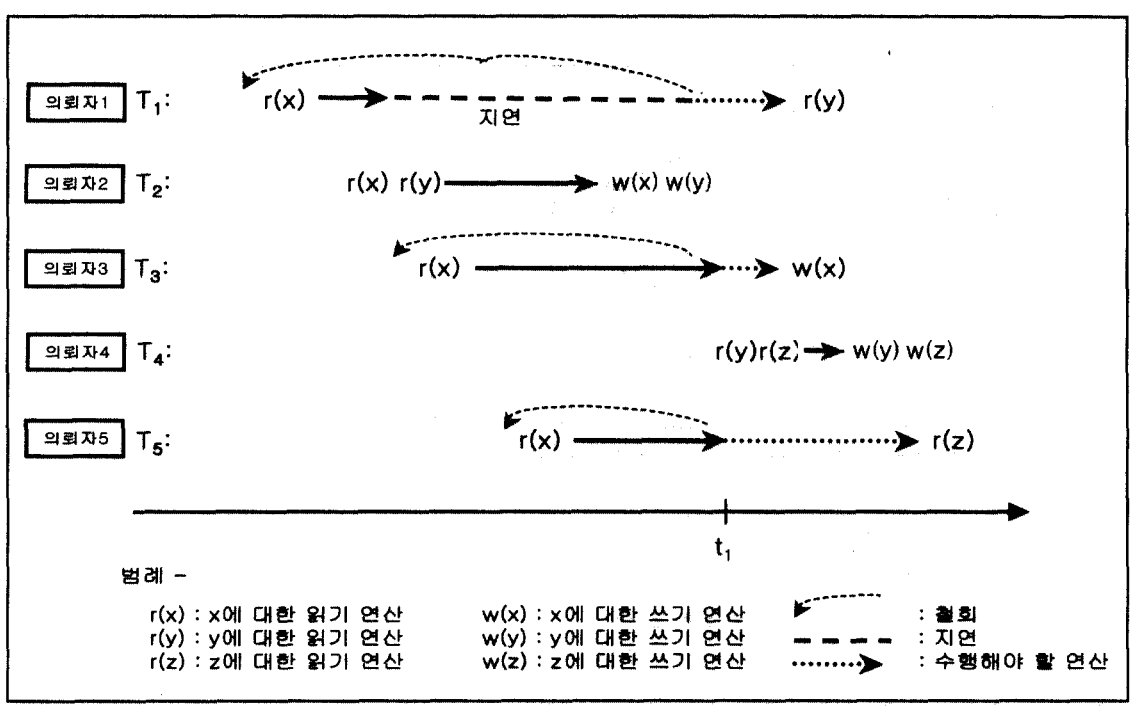
2.1 OCC-UTS를 질의 거래가 많은 환경에 적용했을 때의 문제점

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.

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 캐쉬 내의 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 하나라도 현재 진행되고 있는 어떤 거래가 읽었다면 그 거래는 바로 철회된다. 예 2.1에서 이러한 문제가 자세히 설명되고 있다.

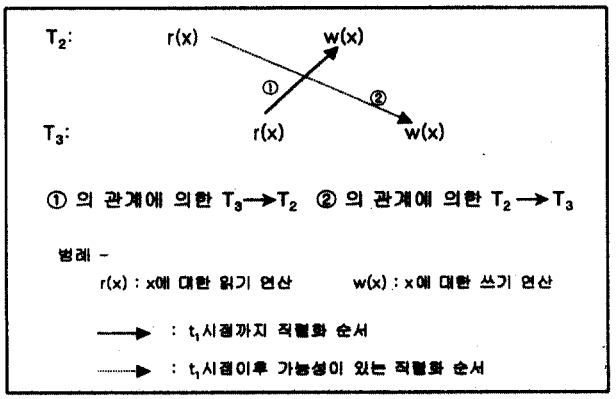
예 2.1 OCC-UTS를 질의 거래가 많은 시스템에 적용했을 경우의 철회 :

질의 거래 T_1, T_5 그리고 갱신 거래 T_2, T_3 및 T_4가 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 2.1).



(그림 2.1) OCC-UTS를 적용했을 경우 거래들의 철회

먼저 갱신 거래 T_2와 T_3간의 관계를 살펴보자. 시간 t_1에서 T_2가 완료되고 서버로부터 무효화 보고서가 방송되어지면 갱신거래 T_3는 t_1 이후 T_2와 비직렬화의 가능성으로 인해 철회된다(그림 2.2).



(그림 2.2) 비직렬화 가능성의 T_2와 T_3의 거래 스케줄

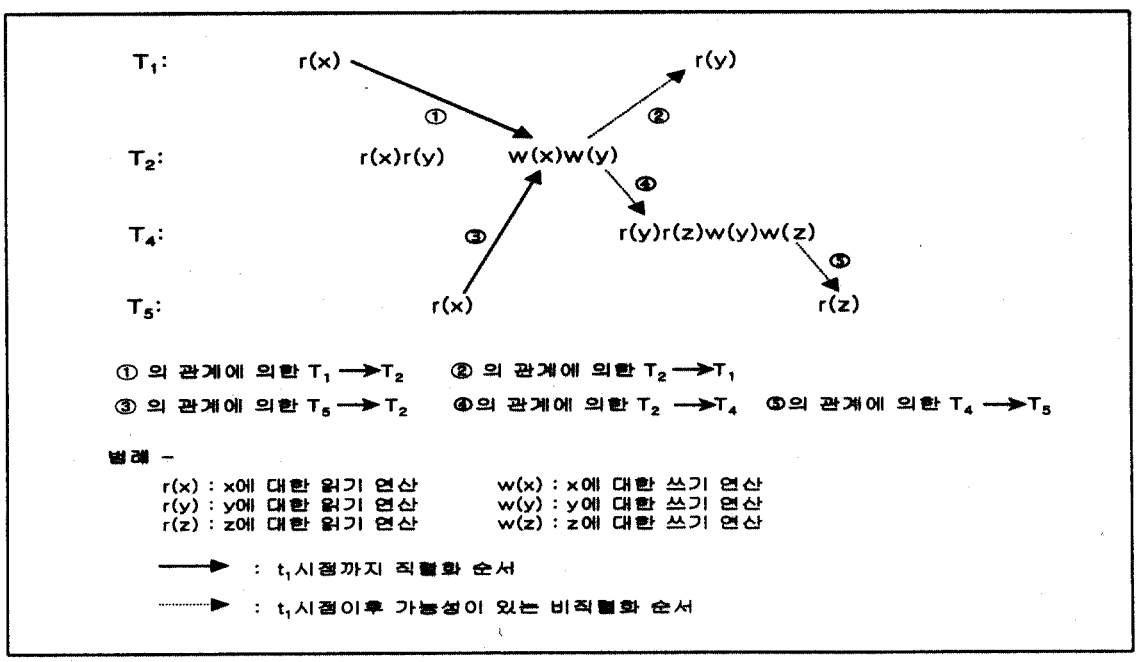
즉, t_1시점에서 방송된 무효화 보고서에 x가 포함되고 그 시점에서 실행 중이며 x를 이미 읽은 T_3는 비직렬화 가능성으로 인해 철회된다. 갱신거래 뿐만 아니라 질의 거래인 T_1과 T_5도 마찬가지이다(그림 2.3).

T_2보다 먼저 거래가 실행된 질의 거래 T_1은 T_2와의 사이에 $T_1 \rightarrow T_2$의 직렬 순서가 보장되어야 하나 완료가 지연됨에 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. 또한, T_5와 같이 그것과 동시에 수행 중인 갱신 거래가 두 개 이상인 경우도 마찬가지다. T_5의 r(x)와 T_2의 w(x)로 인한 $T_5 \rightarrow T_2$의 직렬 순서가 두 거래 완료까지 보장되어야 한다. 그러나, t_1 시점에서 데이타 x를 포함하여 방송된 무효화 보고서에 의해 x를 이미 읽은 T_5는 T_2와 T_4 사이에 일어날 수 있는 직렬 순서 $T_2 \rightarrow T_4$와 T_4와 T_5 사이에서 일어날 수 있는 직렬 순서 $T_4 \rightarrow T_5$에 의해 $T_5 \rightarrow T_2 \rightarrow T_4 \rightarrow T_5$의 비직렬화 가능성 때문에 그 시점에서 실행중인 T_5는 철회된다. □

3. 방송환경에서 갱신 거래 우선 낙관적 동시성 제어 기법

새로이 제안하고자 하는 기법은 방송환경에서 갱신 거래 우선 낙관적 동시성 제어 기법(Optimistic Concurrency Control with Update Transaction First : OCC/UTF)이라 칭한다.

이동 거래가 완료되면 거래에 의해 접근된 데이터 항목에 대한 거래 일관성이 보장되어야 한다. 이 논문에서는 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 최소화하기 때문이다. 낙관적 동시성 제어 기법을 사용할 때 서버는 매 거래 완료 시점에서 그 거래를 포함하는 실행이 직렬화를 만족하는지 아닌지를 검사 할 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[13]. 후방향 유효화는 완료하려는 거래가 최근에 완료된 다른 거래에 의해 무효화되어지지



(그림 2.3) 비직렬화 가능성의 T_2와 T_1 그리고 T_2, T_4 및 T_5 거래 스케줄

않는다는 것을 보장하는 것을 검사한다. 선방향 유효화는 완료하려는 거래가 다른 어떤 활동중인 거래와 충돌하지 않는다는 것을 보장하는 것을 검사한다.

이동 클라이언트/서버 컴퓨팅 환경에서 이동 클라이언트측 거래는 서버가 실행중인 이동 거래들에 대해 잘 모르기 때문에 후방향 유효화 방법을 선택한다. 유효화 될 거래의 읽기연산 항목 집합이 이미 완료된 다른 거래의 쓰기연산 항목 집합과 비교되는 순수한 후방향 유효화 방법과는 달리 OCC/UTF에서는 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 그 클라이언트내에서 자체적으로 해결하도록 함으로써 유효화 과정이 진정으로 분산되어 행해질 수 있다.

OCC/UTF는 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 엄격한 일관성(strict consistency)[14]를 유지해야 한다. 즉, 각 질의 거래가 동시에 수행되는 갱신 거래들의 완료 순서와 일치된 갱신 거래들의 직렬 순서를 본다는 것이다. 그러므로, 이동 클라이언트들 측에서 수행중인 모든 질의 거래들이 일치하는 데이타베이스시스템을 보는지를 검사해야 한다. 더욱이 OCC/UTF는 2장의 관련연구에서 살펴보았던 문제점들을 해결하기 위해, 즉 수행 중인 질의 거래가 이미 읽은 데이터와 동일한 데이터를 갱신한 갱신 거래가 먼저 완료되어 해당 질의 거래가 철회되는 질의 거래들의 철회율을 최대한 줄이기 위해 다음과 같은 방법으로 데이터 일관성을 보장하여 기존 기법의 단점을 보완한다. 한 질의 거래가 이미 읽은 데이터가 갱신 거래에 의해 무효화되어지면 그 질의 거래는 무효화된 항목에 해당하는 데이터의 갱신된 최신의 값을 다시 읽어 갱신 거래에서 질의 거래로의 직렬성을 보장함으로써 무사히 완료할 수 있다.

3.1 OCC/UTF 시스템 모델

OCC/UTF의 시스템 모델은 다음과 같다. 이동 클라이언트의 사용자들은 읽기와 쓰기 연산으로 이루어진 이동 거래에 의해 데이타베이스에 자주 접근할 것이다. 한 이동 클라이언트에 의해 어느 시각에 단지 한 거래만 실행할 수 있다고 가정한다. 즉, 클라이언트는 한 거래가 완료된 후에야 다른 거래를 발생시킬 수 있다. 또한, 각 이동 거래는 먼저 읽기 연산에 의해 읽혀진 데이터 항목들의 부분집합만을 갱신할 수 있고 하나의 데이터 항목은 질의 거래에서 단지 한 번 읽힌다고 가정한다. 데이타베이스는 서버에 의해 저장되고 유지되는 데이터 항목들의 집합이다. 갱신은 이동 클라이언트에 의해 발생되지만 궁극적으로 데이타베이스에 반영되어야 한다. 클라이언트는 서버에게 데이터를 요구하고 서버는 주로 클라이언트들에게 해당 데이터와 제어 정보를 방송하기 위해 방송매체를 사용한다. 또한, 거래의 실행은 원자적(atomic)이다. 즉, 거래는 완료되거나 철회된다. 캐쉬 데이터 일관성을 보장하기 위해 서버는 주기적으로 무효화 보고서를 방송하고 모든 실행중인 이동 클라이언트들은 이 보고서를 기다리고 그 내용에 따라 그들의 캐쉬를 무효화시킨다.

서버는 완료된 거래들에 의해 최근 갱신된 데이터 항목들을 추적하고 매 L초마다 $ts_i = iL$인 시간에 무효화 보고서를 방송한다고 가정하자. 타임스탬프 ts_i 시점에서 무효화 보고서는 현재의 타임스탬프 ts_i와 (j, t_j, v_j)의 리스트로 구성된다. 여기서 j는 데이터 항목, t_j는 $ts_i - \omega L \le t_j \le ts_i$ (ω는 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] | j는 $ts_i - \omega L$과 ts_i 사이에 완료된 거래들에 의해 갱신된 데이터 항목이고 t_j 는 $ts_i - \omega L \le t_j \le ts_i$인 j가 마지막 갱신된 타임스탬프}

무효화 보고서를 받자마자 이동 클라이언트는 캐쉬 내에 j를 유지해야할지 말지를 결정하기 위해 캐쉬 내에 있는 (j, t^c_j) (여기서 t^c_j는 캐쉬 내 j에 대한 타임스탬프임) 리스트와 비교하여 캐쉬 내 데이터 타임스탬프 값이 작다면 즉, t^c_j가 t_j보다 작다면 캐쉬 내 j 데이터 항목의 원래의 값을 제거하고 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.

3.2 이동 클라이언트에서의 거래 처리

이동 거래들의 직렬화를 성취하기 위한 이동 클라이언트측 알고리즘은 다음과 같다. 각 이동 클라이언트들은 각 거래가 갱신 거래인지 질의 거래인지를 거래 수행 시작시점에서 안다고 가정한다. 각 이동 클라이언트들은 거래를 위해 다음과 같은 두 가지 집합을 유지한다.

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

$NewValues(T_{id})$ = {[j, v_j] | j는 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 표3.1과 같은 알고리즘을 수행한다.

〈표 3.1〉 이동 거래를 처리하기 위한 클라이언트측 알고리즘

```
1. On receiving r_T(j) {
        if j is in the cache {
          answer by using the value in its cache ;
        }
        else { go uplink to the server and load j into its
        cache ;
          answer by using the value in its cache ;
        }
    }
2. On receiving w_T(j) {
        update the value in its cache ;
    }
3. On receiving commit_T request {
        if T is update transaction {
          send a RequestCommit message, along with
          Tid, ReadSet and
    NewValues_i ;
        }
        else { commit T ; }
    }
4. On receiving the invalidation report IR(ts_i) {
        if Tid appears in CommitList { commit T ; }
        if Tid identifier appears in AbortList { abort T ; }
        if (ts_i - ts_lb > wL) { drop the entire cache ; }
        else {
          for every item j in its cache {
            if there is a pair (j, t_j) in IR(ts_i) {
              if t^c_j < t_j { throw j out of the cache ; }
              else { t^c_j = ts_i ; }
            }
          }
        }
        ts_lb = ts_i ;
        if any data item was dropped from its cache {
            if there is an active T such that its ReadSet
            contains any dropped
        data item {
                    if T is update transaction { abort T ; }
                    else { read d ;
                        (∀ d ∈ all the data in ReadSet )
                    }
                }
        }
    }
```

각 이동 클라이언트는 읽기, 쓰기, 완료 및 무효화 보고서가 방송될 때 각각 다음과 같은 수행을 한다. 읽기 연산시에 이동 클라이언트는 필요로 하는 데이터 항목이 자체 캐쉬 내에 있으면 즉시 캐싱된 데이터를 사용하고, 없다면 서버에 요구하여 다음 방송주기에 서버측 데이타베이스 내의 해당 데이터 값과 타임스탬프 값을 얻는다. 쓰기 연산시에는 해당 데이터 항목의 값을 갱신하고 타임스탬프 값도 갱신한다. 갱신 거래 T에 대한 완료를 해야할 때 거래 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 캐쉬 내 데이터 항목의 값과 타임스탬프는 의미가 없으므로 제거한다. 무효화 보고서 내용에 따라 캐쉬 내 해당 데이터 항목들을 무효화하는 과정은 앞서 시스템 모델에서 언급된 바와 같다. 또한, 각 클라이언트의 캐쉬 내 무효화된 데이터 항목을 $ReadSet(T_{id})$에 포함하고 있는 실행 중인 이동 거래가 있다면 다음과 같은 과정으로 처리한다. 그 이동 거래가 갱신 거래일 경우 그 거래는 철회된다. 질의 거래일 경우에는 예 2.1에서처럼 그 질의 거래가 읽기 연산한 대상 데이터 항목을 갱신하고 먼저 완료된 갱신 거래에 의해 갱신된 또다른 데이터 항목에 대한 연산 충돌이 일어날 수 있다. 이러한 비직렬화 가능성이 발생함으로써 그 질의 거래가 철회되는 것을 막기 위해 다음과 같은 과정을 밟는다. 즉, 이러한 경우에 질의 거래 → 갱신 거래 → 질의 거래의 비직렬 순서가 아니라 갱신 거래 → 질의 거래로의 직렬 순서가 반드시 유지되도록 $ReadSet(T_{id})$의 데이터 항목 중 무효화된 항목에 대한 새로운 값을 무효화 보고서 리스트 내용을 통해 다시 읽는다. 그러므로 마지막 읽기 연산까지 그 질의 거래는 동시에 수행중인 거래들과 직렬 순서를 유지하므로 무사히 완료된다. 이와 같이 서버에게 어떤 완료요구 없이 각 클라이언트가 직렬성을 위배하지 않으며 질의 거래를 독립적으로 처리할 수 있도록 한다.

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.

3.3 서버에서의 거래 처리

서버가 처리해야 할 자세한 알고리즘은 <표 3.2>에 나타나 있다.

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 데

이터 항목들이 $ts_{i-1} < t_j < ts_i$인 같은 타임스탬프 t_j를 갖도록 $U(ts_i)$를 갱신한다.

〈표 3.2〉 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) ;
   }
```

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.

3.4 OCC/UTF의 적용

OCC/UTF에서는 질의 거래가 동시에 수행 중인 갱신 거래의 완료와 상관없이 무사히 완료된다. 이러한 사항이 예 3.1에서 자세히 설명되어 있다.

예 3.1 OCC/UTF를 질의 거래가 많은 시스템에 적용했을 경우의 완료 :

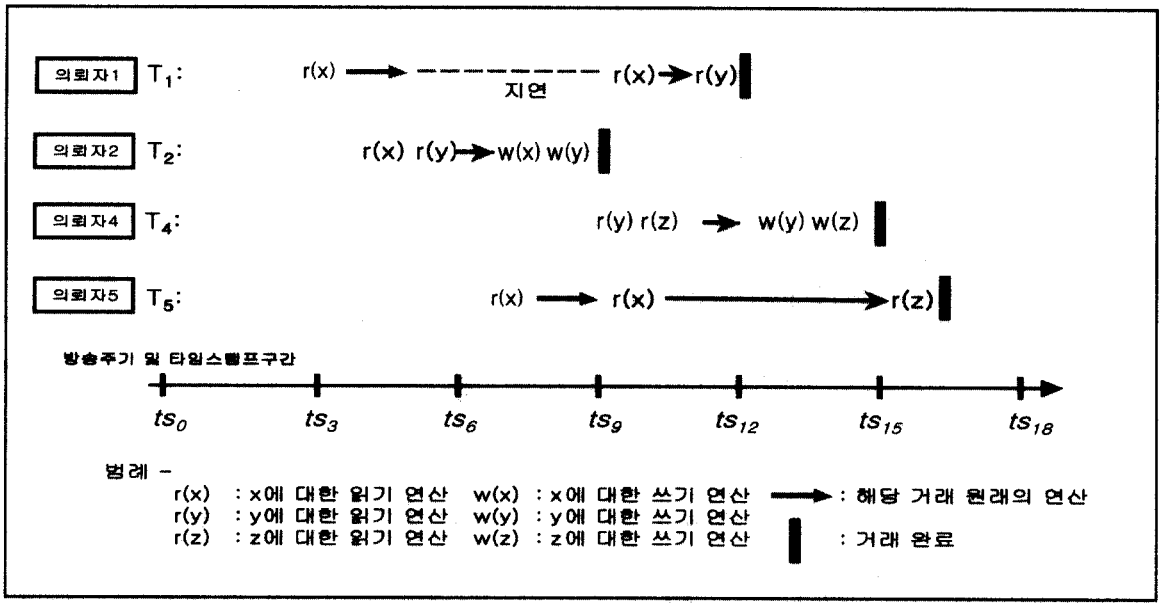
질의 거래 T_1과 T_5 그리고 갱신 거래 T_2와 T_4가 이동 클라이언트측에서 각각 실행하고 있다고 가정한다. 또한, 매 3초마다 무효화 보고서를 방송한다고 가정한다. T_1은 T_2보다 거래 시작이 먼저 된 후 무선환경의 단절 특성에 따른 실행 지연으로 완료가 늦어진 경우라 가정한다(그림 3.1).

시간 ts_9에서 T_2가 완료된다. ts_9 시점에서 무효화 방송에 의해 각 클라이언트의 캐쉬 내 x와 y의 값이 새로운 값으로 채워진다. 즉, 먼저 수행을 시작한 T_1은 x를 읽은 후 ts_9 시점에서 방송된 무효화 보고서에 의해 x가 무효화된다. 그러므로, T_1은 이 시점에서 새로운 값 x를 다시 읽고 완료 때까지 $T_2 \rightarrow T_1$의 직렬 순서가 보장되므로 무사히 완료된다. 거래 T_5도 마찬가지다. ts_9 시점 전에 읽기 연산을 시작한 T_5는 ts_9 시점에서 데이터 x가 무효화되었으므로 새로운 값 x를 다시 읽은 다음, 완료 때까지 $T_2 \rightarrow T_4 \rightarrow T_5$의 직렬 순서가 보장되므로 무사히 완료된다. □

4. 정확성 검증

우리는 OCC/UTF 알고리즘이 비교적 적은 비용과 기존의 방송 체계 시스템에 처리 부하로 인한 영향을 거의 끼치지 않음을 보인다. 지금 우리는 OCC/UTF 알고리즘을 사용할 때 질의 거래(Q)와 갱신 거래(U)들로 이루어지는 직렬화 그래프(SG)가 항상 순환이 아님을 증명한다.

아래 증명에서 Q와 U사이의 데이터 충돌에 관한 모든 경우를 다 고려한다. 각 경우의 정확성 검증에서 직렬화 그래프가 이용된다. N을 거래들을 나타내는 노드들의 집합이라 할



(그림 3.1) OCC/UTF를 적용했을 경우 거래들의 완료

때 SG = (N, E)로 표현한다. 모든 경우에 대한 유일한 거래들은 Q와 U이다. 그러므로 N = {Q, U}이다. E는 거래 T_i와 T_j사이에 적어도 한 쌍의 충돌 연산이 존재하고, 충돌되는 T_i의 연산이 T_j의 연산을 앞서는 순서쌍 (T_i, T_j)들의 집합이다. 우리의 표기법에서 $T_i \rightarrow T_j$는 순서쌍(T_i, T_j)를 나타내기 위해 사용될 것이다.

거래 실행 중 일어날 수 있는 경우들이 다음의 세 가지로 요약되어진다. 먼저 <표 3.1> 알고리즘에서 if there is an active T such that its *ReadSet* contains any dropped data item 문장이 참이 아닌 그 밖의 경우이다. 이 경우는 다음의 두 가지 경우로 다시 나누어진다.

Case 1 : 완료된 갱신 거래에 의해 갱신된 데이터 항목과 동시에 수행 중인 질의 거래의 연산 대상 데이터 항목 사이에 공통된 것이 존재하지 않는 경우이다.

이 경우에는 Q와 U의 데이터 항목들로 이루어진 집합의 교집합이 공집합인 경우이다. 그러므로, Q와 U사이에 충돌된 연산이 없는 경우 즉, E = {}인 경우이다. 결과적으로 직렬화 그래프 SG는 Q와 U 두 개의 노드가 연결되지 않은 그래프이므로 순환이 아니다.

Case 2 : 완료된 갱신 거래에 의해 갱신된 데이터 항목과 동시에 수행 중인 질의 거래의 연산 대상 데이터 항목 사이에 공통된 것이 존재하나 그 갱신된 데이터 항목을 갱신 거래가 완료되어 무효화 방송된 후에야 비로소 그 새로운 값을 질의 거래가 읽는 경우이다.

이 경우에는 Q와 U의 데이터 항목들로 이루어진 집합의 교집합이 공집합이 아닌 경우이다. 그러나, 그 공통된 데이터 항목들을 해당 갱신 거래가 갱신 후 완료되기 전에 질의 거래가 아직 읽기 연산하지 않은 경우이다. 그러므로, 공통된 항목이 x라 할 때 $w_u(x) \rightarrow r_Q(x)$와 E = {U → Q}가 성립된다. 결과적으로 직렬화 그래프 SG는 순환이 아니다.

다음은 나머지 경우로 <표 3.1> 알고리즘에서 if there is an active T such that its *ReadSet* contains any dropped data item 에 대해 참인 경우이다.

Case 3 : 완료된 갱신 거래에 의해 갱신된 데이터 항목과 동시에 수행 중인 질의 거래의 연산 대상 데이터 항목 사이에 공통된 것이 존재하고 그 갱신된 데이터 항목을 갱신 거래 완료되어 무효화 방송되기 전에 질의 거래가 이미 읽은 경우이다.

이 경우는 Q와 U의 데이터 항목들로 이루어진 집합의 교집합이 공집합이 아니고 그 공통된 데이터 항목들이 무효화되기 전에 질의 거래가 이미 읽은 경우이다. 이 시점에서 공통된 항목이 x라 할 때 $r_Q(x) \rightarrow w_u(x)$이므로 E = {Q → U}가 성립된다. 그러므로 질의 거래가 갱신 거래 완료 전에 먼저 완료한다면 직렬화 그래프 SG는 순환이 아니다. 그러나, 그 질의 거래의 수행이 해당 갱신 거래가 완료된 후에도 계속되어진다면 다른 공통된 항목 y가 존재하여 $w_u(y) \rightarrow r_Q(y)$로 인해 U → Q가 E에 더해질 수 있다. 즉, E = {Q → U, U → Q}가 성립되어 직렬화 그래프 SG가 순환이다. 그러나, <표 3.1> 알고리즘에 나타난 바와 같이 read d ; 에 의해서 x가 무효화된 뒤 새로운 값으로 채워지면 해당 질의 거래가 다시 읽음으로써 직렬 순서 Q → U가 없어지게 된다. 그러므로, E = {U → Q}가 되고 직렬화 그래프 SG는 결국 순환이 아니다.

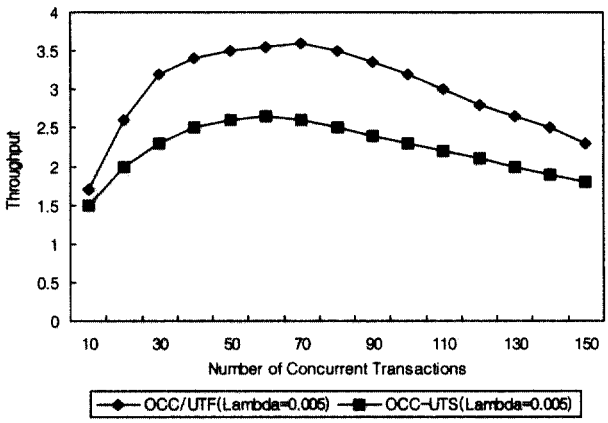
5. 성능 평가

이 장에서는 모의 실험을 통해 OCC-UTS와 OCC/UTF의 처리율을 비교 분석한다. 기본 환경은 다음의 <표 5.1>에서 보여주는 모델 매개변수들과 일치한다.

〈표 5.1〉 모델 매개변수

매개변수	설 명	값 / 표현
a	이동 거래에 의해 접근된 데이터 항목의 수	10
M	지역 캐쉬 없는 이동 거래의 실행 시간	20 sec
l	이동 클라이언트에서 한 데이터 항목에 대한 순수한 실행 시간과 비교되어진 상대적인 네트워크 지연 값	4
E	지역 캐쉬 사용 이동 거래의 실행 시간	$\frac{M}{1+I} + \frac{IM}{1+I}(1-h)$ sec
μ	서버에서 데이터 항목 당 갱신 비율	0.0001/sec
λ	이동 클라이언트에서 데이터 항목 당 읽기 비율	0.01, 0.005 and 0.001/sec
L	무효화 보고서를 위한 방송 주기	3 ~ 27 sec.
C	동시에 수행하는 이동 거래들의 수	10 ~ 150
ω	무효화 방송 윈도우	1 and 2
s	이동 클라이언트에서 주기당 단절 가능성	0.0 ~ 0.9

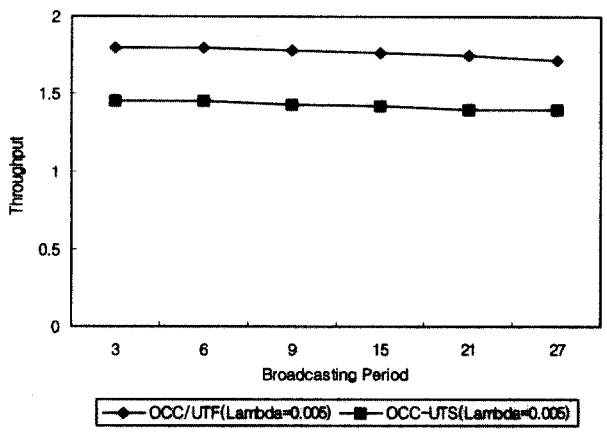
(그림 5.1)은 μ가 $0.0001 \times C$일 때 OCC-UTS와 OCC/UTF의 처리율을 나타낸다.



(그림 5.1) 이동 거래 처리율 (s = 0.0)

OCC/UTF와 OCC-UTS의 처리율은 C가 각각 70과 60일 때 최대값을 보인다. 모든 C에 대해 OCC/UTF의 처리율이 OCC-UTS이 처리율보다 더 크다. C가 150일 때 약 80%까지 처리율이 향상되었다. 이것은 OCC/UTF가 OCC-UTS와 달리 질의 거래 처리를 하기 때문이다. 즉, 임의의 질의 거래에 의해 이미 읽혀진 데이터가 갱신 거래에 의해 무효화 될지라도 그 질의 거래는 철회되지 않고 무효화된 데이터를 재읽기 연산하고 그 이후의 거래 연산을 다시 하기 때문이다. 그 결과 전체 거래 철회율이 감소하고 처리율 향상을 가져왔다.

(그림 5.2)는 C가 10일 때 방송 주기 크기에 대한 OCC/UTF와 OCC-UTS의 처리율을 보인다.

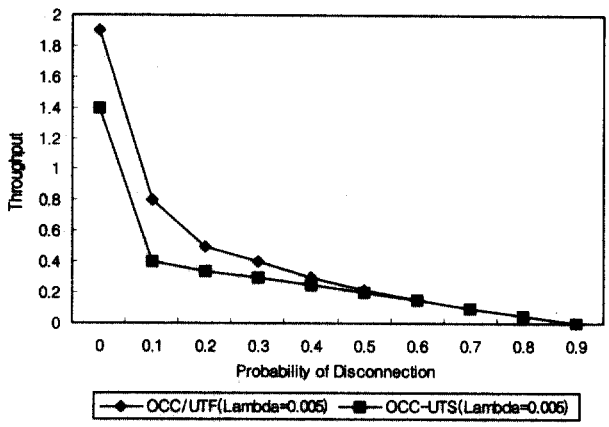


(그림 5.2) 이동 거래 처리율 - 방송 주기에 대한 민감도 (s = 0.0)

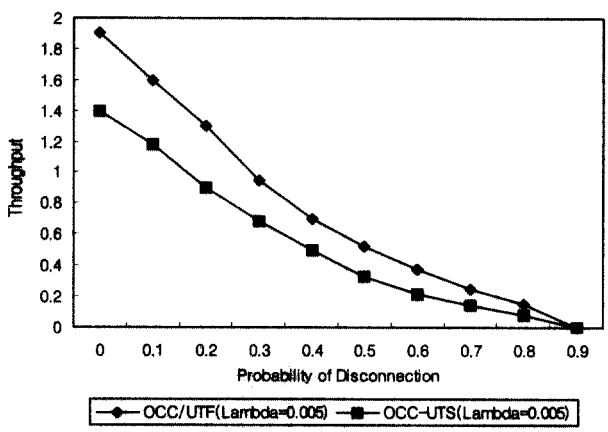
이동 클라이언트 거래 처리율이 방송 주기 크기에 무관한 결과를 보이고 있다. 방송 주기 크기가 작다는 것은 물론 데이터 충돌이 더 일찍 감지되어 이동 갱신 거래일 경우는 철회가 빨리 일어날 것이다. 그러나, 질의 거래일 경우 무효화 데이터를 읽은 거래가 일찍 감지되고 무효화된 해당 데이터 읽기 연산 이후의 거래 처리를 다시 하도록 하여 재스케줄(reschedule)되어질 수 있다. 그러므로, 방송 주기가 작을수록 데이터 충돌을 일찍 감지하여 이동 질의 거래가 철회됨을 막고 일찍 재처리될 수 있는 잠재된 이익을 가진다. 그러나, 우리의 실험 모델에서는 철회된 이동 거래들이 재스케줄되어지는 상황을 고려하지 않는다. 즉, 잠재된 이익이 (그림 5.2)에는 반영되지 않았다.

(그림 5.3)과 (그림 5.4)는 C가 10일 때 통신 단절 가능성에 대한 OCC/UTF와 OCC-UTS의 처리율을 보인다.

(그림 5.3)과 (그림 5.4)의 결과는 통신 단절에 대한 처리율 비교에서 전반적으로 OCC/UTF가 OCC-UTS보다 향상되었음을 보여준다. (그림 5.3)에서는 무효화 방송 윈도우 크기 ω가 1일 때 통신 단절에 대한 OCC/UTF와 OCC-UTS의 처리율을 보여준다. 갱신 발생 빈도가 최소일 때 통신 단절에 대한 영향을 측정하기 위해 우리는 C를 10으로 했다. 이동 클라이언트의 통신 단절은 그들 거래들의 실행 시간을 연장하고 클라이언트 내 캐쉬 적중율을 떨어뜨린다. OCC/UTF와 OCC-UTS의 처리율은 작은 통신 단절 가능성에도 민감하다. OCC/UTF와 OCC-UTS의 처리율 둘 다 s가 0.6보다 크다면 거의 일치점에 도달한다. 이동 거래 처리율이 단절의 영향을 덜 받도록 하기 위해서는 윈도우 크기를 크게 하면 된다. (그림 5.4)는 ω가 2일 때 통신 단절에 대한 OCC/UTF와 OCC-UTS의 처리율을 보여준다. 예상대로 처리율은 단절 가능성이 증감함에 따라 ω가 1일 때보다 상대적으로 덜 민감하다.



(그림 5.3) 이동 거래 처리율 - 통신 단절에 대한 민감도 (C = 10과 ω = 1)



(그림 5.4) 이동 거래 처리율 - 통신 단절에 대한 민감도 (C = 10과 ω = 2)

6. 결론 및 향후 연구

본 논문에서는 방송환경에서 이동 클라이언트/서버 시스템을 위한 효율적인 동시성 제어 기법인 OCC/UTF를 제안했다. 일관성과 현재성을 동시에 요구하는[15] 주식 거래 또는 교통 정보 시스템과 같은 방송 기반 데이터베이스시스템의 여러 응용들이 있다. 본 논문은 이러한 요구사항에 부응하기 위한 효율적인 동시성 제어 기법을 제시하였다. 방송환경의 특수성을 고려한 OCC/UTF는 기존의 이동 컴퓨팅 환경을 위한 동시성 제어 기법 OCC-UTS 또는 그 밖의 다른 기법과 비교될 때 다음과 같은 장점을 가지고 있다.

첫째, OCC/UTF는 질의 거래가 이미 읽은 데이터 항목들이 동시에 수행된 갱신 거래에 의해 무효화될 때 무효화 보고서에 포함된 해당 데이터 항목에 대한 새로운 값을 해당 질의 거래가 다시 읽음으로써 갱신 거래 → 질의 거래로의 직렬 순서를 유지하여 철회 없이 완료할 수 있도록 하였다. 그럼으로써, 서버에게 질의 거래에 대한 완료 요구 없이 클라이언트가 자체 해결을 할 수 있도록 하였으며 클라이언트 내 모든 질의 거래들이 갱신 거래들의 완료 순서와 일치하는 동일한 갱신 거래들의 직렬 순서를 볼 수 있도록 하여 직렬화를 만족시킨다.

둘째, OCC/UTF는 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 무효화될 데이터의 새로운 값을 포함시켜 방송하도록 함으로써 클라이언트측 질의 거래가 해당 데이터 항목을 재 읽기 연산을 해야 할 때 서버가 다시 방송해야 하는 필요성을 줄임으로써 비싼 무선 대역폭도 효율적으로 사용할 수 있도록 하였다.

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

앞으로의 연구에서는 질의 거래의 읽기 연산 순서가 중요한 환경에서 OCC/UTF를 어떻게 개선하여 적용할 지를 연구해 보고, 또한 방송 채널의 데이터 항목에 대한 구조 즉, Broadcast Disk를 고려하여 클라이언트들에게 더 효율적으로 데이터를 방송함으로써 거래 실행 시간을 단축시키는 향상된 기법을 연구할 것이다.

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

[3] Pitoura, E. and Bhargava, B. "Dealing with Mobility : Issues and Research Challenges," *Technical Report*, Purdue Univ., Nov., 1993.

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

[5] Pitoura, E. and Bhargava, B. "Maintaing Consistency of Data in Mobile Distributed Environment," in *Proceeding of the 15th International Conference on Distributed Computing Systems*, pp.404-413, 1995.

[6] Xuan, P., O. Gonzalez, J. Fernandez & Ramamritham, K., "Broadcast on Demand : Efficient and Timely Dissemination of Data in Mobile Environments," in *Proceedings of 3rd IEEE Real-Time Technology Application Symposium*, 1997.

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

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

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

[10] 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.

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

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

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

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

[15] P. Xuan, et. al, "Broadcast on Demand-Efficient and Timely Dissemination of Data in Mobile Environments," *IEEE Real-Time Technology and Applications Symposium*, pp. 38-48, June, 1997.

이 욱 현

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

1992년 이화여자대학교 전자계산학과 (이학사)

1997년 한국과학기술원 정보및통신공학과 (공학석사)

2002년 전남대학교 전산학과 박사 수료

1992년~1997년 포스데이타 근무

1997년~2000년 광주은행 근무

1999년~2001년 동강대학 초빙교수

2002년~현재 인하대학교 컴퓨터공학부 강의전담교수

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