복합 브로드캐스팅 환경에서 이동 트랜잭션 처리
(Mobile Transaction Processing in Hybrid Broadcasting Environment)

김 성 석 † 양 순 옥 ††
(SungSuk Kim) (SunOk Yang)

요 약 최근에 이동 컴퓨팅 환경에서 여러 데이타 전송 모델이 연구되고 있다. 특히 서버가 반복적으로 필요한 정보를 전파해주는 주기적 푸시 모델에 대한 연구가 활발히 진행되고 있다. 그러나 데이타 평균 대기 시간은 브로드캐스트 한 주기의 길이에 상당히 영향을 받으며, 또한 여러 사용자들간의 접근 데이타가 차이가 날 경우 응답시간에 상당히 나빠질 수 있다. 이 경우, 그 사용자들은 차라리 서버에게 명시적으로 데이타를 요청하기를 바랄 것이다. 이러한 두 가지 접근방식을 모두 지원하는 것을 복합 브로드캐스트라고 한다. 이 환경에서, 본 논문에서는 새로운 이동 트랜잭션 처리 알고리즘(*O-PreH*)을 개발하였다. 우선 서버가 관리하는 데이타는 주기적 브로드캐스트 방식으로 처리되는 *Push_Data*와 요구-처리방식으로 처리되는 *Pull_Data*로 나뉘어 진다. 즉, 사용자는 요구하는 데이타의 타입에 따라 접근하는 방식이 차이가 난다. 또한 서버는 이동 트랜잭션 일관성 유지를 돕기 위해 주기적으로 무효화 보고를 전송해준다. 만약 사용자가 무효화 보고에 의해 하나 이상의 충돌을 발견한다면, 일관성을 침해하지 않는 범위 내에서 그 충돌 순서를 결정한 후(*pre-reordering*) 나머지 연산들을 비관적으로 수행시킨다. 자세한 실험 과정을 거쳐 제안한 알고리즘의 성능 향상을 보였다.

키워드 : 데이타베이스, 이동 컴퓨팅, 동시성 제어, 브로드캐스트, 재순서화

Abstract In recent years, different models in data delivery have been explored in mobile computing systems. Particularly, there were a lot of research efforts in the periodic push model where the server repetitively disseminates information without explicit request. However, average waiting time per data operation highly depends on the length of a broadcast cycle and different access pattern among clients may deteriorate the response time considerably. In this case, clients are preferably willing to send a data request to the server explicitly through backchannel in order to obtain optimal response time. We call the broadcast model supporting backchannel as hybrid broadcast. In this paper, we devise a new transaction processing algorithm(*O-PreH*) in hybrid broadcast environments. The data objects which the server maintains are divided into *Push_Data* for periodic broadcasting and *Pull_Data* for on-demand processing. Clients tune in broadcast channel or demand the data of interests according to the data type. Periodic invalidation reports from the server support maintaining transactional consistency. If one or more conflicts are found, conflict orders are determined not to violate the consistency(*pre-reordering*) and then the remaining operations have to be executed pessimistically. Through extensive simulations, we demonstrate the improved throughput of the proposed algorithm.

Key words : distributed database, mobile computing, concurrency control, broadcast, reordering

1. 서 론

브로드캐스팅 기법은 유, 무선 환경에서 정보 전달 방식의 하나로써 오랫동안 연구되어 왔다. Bellcore의 *Data-cycle* 프로젝트의 경우[1], 서버는 데이타베이스를 높은 대역폭의 (140 *Mbps*) 네트워크로 계속 반복하여 전송시켰으며, 사용자는 특별한 병렬 트랜시버를 이용하여 필요한 정보를 얻음으로써 질의를 처리할 수 있게 하였다. 무선 브로드캐스팅 기법 역시 다수의 사용자에게 정보를 전파할 수 있다는 측면에서, 전력이나 대역폭

† 정 회 원 : 서경대학교 전자상거래학과 교수
sskim03@skuniv.ac.kr
†† 비 회 원 : 고려대학교 컴퓨터학과
soyang@korea.ac.kr
논문접수 : 2003년 9월 1일
심사완료 : 2004년 4월 27일

측면에서 효율적으로 방식으로 간주되었다[2,3]. 즉, 무선 비대칭 통신 환경을 고려할 때, 그 기법은 상대적으로 높은 클라이언트-서버 방향의 높은 통신비용을 줄일 수 있고 서버가 지원하는 클라이언트의 수에 상대적으로 영향을 적게 받을 수 있다는 점에서, 효율적인 정보 전송 방식으로 간주되고 있다.

최근에, 이동 컴퓨팅 환경에서 여러 데이타 브로드캐스팅 모델이 연구되었다. 특히 서버가 반복적으로 필요한 정보를 전파해주는 주기적 푸시 모델에 대한 연구가 활발히 진행되고 있다. 이 환경에서 데이타 질의에 대한 평균 응답 시간을 줄일 수 있는 브로드캐스트 프로그램을 개발하는 것이 주요한 연구분야중 하나이다. 이를 위하여 데이타 접근 빈도에 따라 각 데이타의 전송 회수를 결정한 후, 각 데이타를 브로드캐스트 슬롯에 균등하게 배치하여 전파시키도록 한다. 사용자 레벨에서 접근 시간을 줄일 수 있는 다른 방법으로써 지역 저장장치(캐쉬)를 이용하기 위한 방식도 많이 연구되고 있다.

그러나 데이타 연산 당 평균 대기 시간은 브로드캐스트 한 주기의 길이에 상당히 영향을 받으며, 또한 여러 사용자들간의 접근 데이타가 차이가 날 경우 응답시간에 상당히 나빠질 수 있다[2]. 이 경우, 그 사용자들은 차라리 서버에게 명시적으로 데이타를 요청하기를 바랄 것이다. 이와 같이 두 가지 데이타 접근방식을 모두 지원하는 것을 복합 브로드캐스트라고 한다. 이외에도 사용자들은 여러 목적에 의해 역채널을 사용하려고 한다[4] :

- *지역적으로 데이타 수정 허용* : 일단 사용자가 역채널을 사용할 수 있다면, 지역적으로 캐싱되어 있는 데이타에 수정 작업을 한 후 그 결과를 다시 서버에게 보낼 수 있다. 시스템이 트랜잭션을 지원하려고 한다면, 이는 상당히 중요한 의미를 가진다.
- *피드백(feedback) 및 새로운 프로파일 전송* : 만약 브로드캐스트 프로그램이 적절하지 못하거나, 혹은 사용자의 접근 경향이 변하였을 경우, 각 사용자는 피드백이나 새로운 프로파일을 서버에게 전송해주어야 한다.
- *손상된 데이타 재전송* : 만약 전송 채널에서 에러가 발생하기 쉽다면, 사용자는 서버에게 채널의 노이즈로 인하여 손상된 데이타를 재전송 해달라고 요청해야 한다.

이와 같은 다른 목적들은 이후에도 계속 연구가 되어야 할 부분들이다. 하지만 본 논문의 관심 범위를 벗어난다. 즉 본 논문에서는 복합 브로드캐스트 환경에서 이동 컴퓨터에서 읽기-전용 트랜잭션 수행 알고리즘을 개발하고자 한다.

브로드캐스트 환경에서 이동 트랜잭션 처리를 위한 알고리즘이 여러 논문에서 제안되었다. [5] 논문에서, 저자는 읽기-전용 트랜잭션 수행을 위한 두 가지 프로토콜, *F-Matrix*와 *R-Matrix*를 제안하였다. 비록 *F-Matrix*가 더 나은 성능을 보이지만, 많은 계산 수행 및 일관성 검사를 위한 정보 전송을 위해 높은 대역폭을 요구한다. 또한 두 프로토콜 모두 트랜잭션 수행을 위한 정확성 기준으로써 *갱신 일관성(update consistency)*을 기반으로 하는데, 이는 직렬화가능성(serializability)을 약화시켰다[6]. [7]과 [8] 논문에서는 읽기-전용 트랜잭션의 일관성을 유지할 수 있는 여러 브로드캐스트 방식이 제안되었다. 특히, *다중 버전 브로드캐스트 방식*의 경우, 트랜잭션의 철회 회수를 줄이기 위해 각 데이타에 대하여 최신 값과 함께 몇 개의 이전 버전 값을 함께 보내도록 하였다. 그러나 이 방식은 브로드캐스트의 한 주기 크기를 상당히 증가시켰으며, 따라서 트랜잭션의 응답시간을 상당히 악화시키게 되었다. 게다가 트랜잭션이 시작하는 시점에서 직렬화 순서가 결정되게 되는데, 이는 너무 제약적인 조건이 될 수 있다. *직렬화-그래프 검사(Serialization-Graph Testing)* 방식의 경우, 사용자와 서버 모두 충돌 검사를 위해 직렬화 그래프의 복사본을 유지하고 있어야 하며, 이 과정에서 상당한 부하가 발생하게 된다. [9] 연구의 경우, 각 데이타마다 가장 최근에 그 데이타를 수정하고 완료한 트랜잭션의 타임스탬프 값을 함께 유지하도록 하였다. 따라서 제안한 알고리즘은 이동 사용자의 자치성을 보장할 수 있고, 트랜잭션의 직렬화 순서를 유연하게 결정할 수 있게 한다. 하지만, 이동 환경에서 타임스탬프를 유지하는 비용이 문제가 될 수 있다.

이러한 연구에서 제안한 알고리즘들은 모두 순수한 푸시전용 브로드캐스트 환경을 가정하고 있다. 즉 사용자는 필요한 데이타를 얻기 위해 단지 브로드캐스트 채널을 튜닝만 하고 있다. 우리의 이전 연구 역시 동일한 환경에서 이동 트랜잭션에 대한 낙관적 처리 알고리즘을 제안하였다[10]. 본 논문에서는 이전 연구를 기반으로 하여, 사용자가 역채널을 사용하여 필요한 데이타를 요청할 수 있는 복합 브로드캐스트 환경에 적합한 새로운 트랜잭션 처리 알고리즘(*O-PreH*)을 개발한다. 이동 트랜잭션은 기본적으로 낙관적으로 수행되며, 주기적인 무효화 보고에 의해 충돌이 발견될 경우 불필요한 철회 회수를 줄이기 위해 *전위-재순서화(pre-reordering)* 기법을 적용한다.

본 논문의 구성은 다음과 같다: 2장에서는 기반으로 하는 시스템 모델을 설명한다. 3장에서는 제안하는 트랜잭션 처리 알고리즘을 설명하며, 4장에서는 시뮬레이션 작업을 이용하여 제안한 기법의 성능 향상을 설명한다. 마지막으로 5장에서 본 논문을 결론짓는다.

2. 시스템 모델

이 절은 제안하는 이동 컴퓨팅 시스템 모델에 대하여 간략하게 설명한다. 시스템은 크게 데이타 서버와, 낮은 대역폭의 무선 네트워크에 의해 서버와 연결되는 다수의 클라이언트로 구성된다. 서버는 데이타베이스의 일관성을 유지하며, 갱신 트랜잭션에 의해 수정되는 최신 값을 유지한다. 또한 클라이언트의 데이타 요구도 처리한다. 서버가 관리하는 데이타베이스의 데이타들은 사용자의 접근 빈도에 따라 *Push_Data*와 *Pull_Data*로 나뉘어있다. *Push_Data*에 포함된 데이타들은 *Pull_Data*에 포함된 데이타보다 빈번하게 접근되며, 따라서 주기적으로 반복해서 브로드캐스트된다. 본 논문에서는 브로드캐스트 프로그램의 구성에 대하여 특별한 가정을 하지 않는다. 단지 한 주기의 브로드캐스트내에 포함된 모든 데이타들은 동일한 회수만큼씩 나타나는 플랫(flat) 구조이며, 한 주기내의 모든 데이타들은 그 주기 시작 시점에 해당하는 일관된 상태 값을 가진다고 가정한다[2]. 즉, 한 주기내의 모든 데이타들은 이전 주기동안 완료한 트랜잭션에 의해 생성된 가장 최신 값을 가진다. 이에 반하여, *Pull_Data*의 데이타들은 사용자가 그 데이타를 요구할 경우에만 서비스해준다.

서버는 사용자의 데이타 요구 처리뿐만 아니라 이동 컴퓨터에서 지역적으로 수행되고 있는 트랜잭션의 일관성 유지를 위하여 유용한 정보도 전송해준다. 즉 매 주기의 *Push_Data*를 전송하기 전에, 최근 이전 $n(\geq 1)$ 주기동안 수정된 데이타 항목에 대한 목록을 무효화보고(invalidation report, *BC.IR*) 형식으로 보내준다.

이동 사용자는 자신의 이동 터미널을 이용하여 작업을 수행한다. 트랜잭션의 연산이 제기되어 필요한 데이타를 얻는 과정은 얻고자 하는 데이타의 종류에 의해 결정된다. 만약 얻고자 하는 데이타가 *Push_Data*의 한 원소라면, 브로드캐스트 채널만 튜닝한다. 이 경우 사용자는 수동적인 수신자로써, 브로드캐스트 매체를 특별한 종류의 메모리처럼 사용한다. 이 때문에 대용량의 정보 전송을 위해서는 브로드캐스팅 기법이 상당한 효율을 보일 수 있다. 그러나, 그 채널은 오직 순차적 접근만이 가능하므로 원하는 데이타에 얻기 위해서는 상당 시간을 기다려야 하는 단점이 있다. 결론적으로, 접근 지연 시간은 *Push_Data*의 크기에 영향을 받을 수 있으므로 *Push_Data*의 크기가 큰 것이 반드시 좋을 수가 없다. 만약 사용자가 원하는 데이타가 브로드캐스트되지 않는 데이타라면, 즉 *Pull_Data*를 접근하고자 한다면, 이것은 일반적인 클라이언트-서버 구조로써, 사용자는 원하는 데이타를 서버에게 요청해야 한다. 이 경우 평균 데이타 접근 시간은 네트워크 부하 및 전체적인 작업량에 영향을 받게 된다.

일반적으로 서버에서 처리할 수 있는 데이타 요청 회수가 어느 범위를 넘어선다면, 정상적으로 그 서비스를 처리하는 것이 불가능해진다[11]. 따라서 *Push_Data*와 *Pull_Data*의 크기를 적절하게 결정하고 그 브로드캐스트 프로그램을 작성하는 것이 중요한 연구분야이지만, 이것은 본 논문의 주요 주제를 벗어난다. 우리는 단지 서버가 사용자의 접근 경향을 미리 알고 있으며 그것이 정적이라고 가정한다.

3. 트랜잭션 처리 알고리즘

이 절에서는 제안하는 트랜잭션 처리 알고리즘을 설명한다. 이동 사용자와 서버간의 메시지의 수를 줄일 수 있다는 이유로, 기본적으로 트랜잭션 처리 알고리즘은 주기적인 무효화보고를 이용하는 낙관적 기법을 사용한다. 사용자가 필요로 하는 데이타가 서버에서 계속 갱신되는 환경에서 트랜잭션의 일관성을 유지하는 것이 알고리즘 개발의 중요한 관점이 된다. 이를 위해 우리는 전위-재순서화 기법을 제안한다.

3.1 전위-재순서화 개념

분산 데이타베이스 시스템에서 트랜잭션 수행의 정확성 기준으로써 일반적으로 *직렬화가능성(Serializability)*을 채택한다[6]. 이 환경에서 여러 알고리즘이 제안되었지만, 낙관적 동시성 제어 기법(Optimistic concurrency control)을 기본 방식으로 채택하였다. 이 기법은 트랜잭션의 일관성을 유지하는데 비교적 적은 수의 메시지를 필요로 하며 서버에서 브로드캐스팅 기법과 효율적으로 연동될 수 있다는 점에서 이동 컴퓨팅 환경에 매우 적합하다.

*BC.IR*에서는 시스템 모델에서 언급하였듯이 이전 주기동안 완료된 트랜잭션에 의해 갱신된 데이타 항목에 대한 목록을 가지고 있으므로, 이 정보를 이용한다면 이동 트랜잭션의 지금까지 수행 결과에 대한 부분적인 일관성 검사를 할 수 있다. 그 결과 서버에서 완료한 트랜잭션과 수행중인 트랜잭션간에 하나 이상의 충돌이 발견된다면, 낙관적 수행에서는 무조건 철회시켜야 하며, 이는 응답시간을 상당히 악화시키게 된다. 본 논문에서는 이와 같은 무조건 철회 회수를 줄이기 위해 발견된 충돌에 대하여 충돌 순서를 결정하고자 한다. 다음 그림 1의 예에서, 이동 트랜잭션 T_m은 데이타 x를 읽었다. 하지만 동일한 주기 혹은 그 이후의 주기에서 서버에서 수행되고 완료한 트랜잭션 T_{si} $(i \geq 1)$가 데이타 x를 갱신하였다. 따라서 사용자가 *BC.IR*을 받게되면 읽은 데이타 x에서 읽기-쓰기 충돌을 발견하였음을 알게된다. 이 충돌에 대하여 접근한 데이타의 값을 고려한다면, 당연히 충돌순서는 $T_m \rightarrow T_{si}$으로 결정할 수 있다. 왜냐하면 서버에서 전파되는 모든 데이타 값은 그 주기의 시작 시점의 값이므로, 이후에 갱신 연산보다 읽기 연산

의 수행 순서가 더 빠르다고 결정할 수 있다. 따라서 수행중인 트랜잭션에 대하여 충돌이 발견된다면 무조건 철회하는 대신, 그 충돌 순서를 $T_m \rightarrow T_{si}$으로 결정한다(*전위-재순서화*). 하지만 T_m이 전위-재순서화된 이후에는 나머지 연산들을 그 충돌 순서를 지키기 위해 비관적인 방식으로 수행되어야 한다.

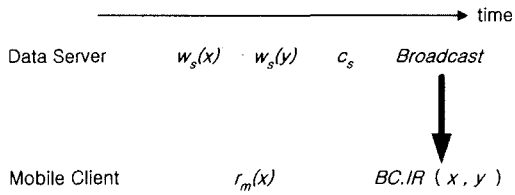


그림 1 전위-재순화의 예

정의 1. 주기적인 무효화보고에 의해 하나 이상의 충돌이 발견된다면, 이동 트랜잭션의 읽기 연산이 서버에서 완료한 트랜잭션의 갱신연산보다 수행 순서가 앞선다고 결정하며, 이를 *전위-재순서화*한다고 한다.

보조정리 1. 이동 읽기-전용 트랜잭션에 대하여 전위-재순서화하더라도 직렬화가능성을 위배하지 않는다.

증명. 읽기-전용 트랜잭션 T_m에서 발생할 수 있는 충돌은 이전 주기동안 서버에서 완료한 갱신 트랜잭션 T_{si} ($i \geq 1$)과 관련된다. T_m을 수행하는 동안 발생한 충돌에 대하여 전위-재순서화한 것이 직렬화가능성을 침해한다는 것은 T_m이 T_{si}가 갱신한 데이타 값을 읽게되는 경우이다. 하지만, 브로드캐스트 프로그램은 주기적으로 전송되며, 따라서 갱신된 데이타 값은 다음 주기에만 보여진다. 따라서 잘못될 수 있는 경우는 발생하지 않는다. □

3.2 순수한 브로드캐스트 환경에서 *O-Pre* 알고리즘

서버가 관리하는 모든 데이타들이 *Push_Data*에 포함된다면 이는 순수한 푸쉬 브로드캐스트 환경이 된다. 이 모델은 복합 브로드캐스트 모델보다 단순하므로 본 논문에서는 먼저 이 환경에 적합한 이동 트랜잭션 처리 알고리즘(*O-Pre : Optimistic algorithm based on Pre-reordering*)을 개발한다. 또한 이 환경에서 개발한 알고리즘은 복합 브로드캐스트 환경으로 적응시키기가 편리한 장점도 있다.

3.2.1 이동 컴퓨터의 알고리즘

순수한 브로드캐스트 환경에서는 사용자가 이동 트랜잭션 T_m을 수행하기 위해 필요한 데이타는 브로드캐스트 채널을 튜닝함으로써 모두 얻을 수 있다. 트랜잭션을 수행할 때는 기본적으로 낙관적으로 수행하며, 읽은 데이타는 *ReadSet*에 저장한다. 수행도중 *BC.IR*을 받게되면 수행중인 트랜잭션에 대한 부분적인 충돌 발생 여부도 발견할 수 있다 ($ReadSet \cap BC.IR \neq$ null). 일단 충돌이 발생하면, 전위-재순서화 개념을 적용한다. 즉, 서버에서 수행하고 완료한 갱신 트랜잭션보다 T_m의 읽기 연산의 수행순서가 앞선다고 결정한다.

일단 T_m이 전위-재순서화가 되면, 나머지 연산들은 비관적인 방식으로 수행된다. 먼저, 사용자는 전위-재순서화를 발생시킨 *BC.IR*을 *UpdateList*에 저장하며, 이후 트랜잭션이 종료할 때까지 받게되는 모든 주기적인 *BC.IR*을 계속 *UpdateList*에 저장한다. T_m의 나머지 연산 $r(x)$를 수행하려고 하면 먼저 데이타 x가 *UpdateList*의 원소인지를 검사한다. 만약 x가 *UpdateList*의 한 원소라면 T_m은 철회되어야 한다. 그렇지 않은 경우에는 그 연산을 수행하면 된다. *O-Pre* 알고리즘에 따라, 각 클라이언트는 다음과 같은 정보를 유지하고 있어야 한다:

MobileData = { *ReadSet, UpdateList* }

ReadSet : T_m이 읽기 연산을 수행한 데이타 집합

UpdateList : *pre-reordering*된 이후 *BC.IR*에 포함된 데이타

단 *UpdateList*는 수행중인 T_m이 충돌에 의해 재순서화가 적용되는 그 주기의 *BC.IR*부터 종료될 때까지의 *BC.IR*을 저장하고 있다. 다음 알고리즘 1은 이동 트랜잭션에 대한 *O-Pre* 알고리즘을 서술하고 있다.

```
▷ 읽기 연산 r(x)를 받았을 때
  IF (충돌이 발견되지 않은 경우)
  /* Tm이 아직 재순서화되지 않은 경우 */
     - 브로드캐스트 데이타를 이용하여 연산 수행
     - ReadSet ← ReadSet ∪ { x }
  ELSE IF (x ∈ UpdateList) abort Tm
  ELSE
     - 연산 수행
     - ReadSet ← ReadSet ∪ { x }
▷ 완료 연산을 받았을 때
  - 조건검사 없이 완료 가능
▷ 주기적인 BC.IR을 받았을 때
  IF (충돌이 발견되지 않은 경우)
     - Temp ← ReadSet ∩ BC.IR
     IF (Temp ≠ NULL) /* 재순서화 될 경우 */
        UpdateList ← BC.IR
  ELSE UpdateList ← UpdateList ∪ BC.IR
```

알고리즘 1 클라이언트를 위한 *O-Pre* 알고리즘

위 알고리즘에서 알 수 있듯이, 일단 T_m이 재순서화가 되면 더이상 w-r 충돌 검사를 수행하지 않는다. 그 이유를 설명하기 위해 재순서화된 이후 다른 충돌이 더 발생하였다고 가정하자. 새로운 충돌이 발생하더라도 실제 충돌 순서는 처음 재순서화된 충돌 순서와 동일하게 결정되며, 이는 *pre-reordering* 개념을 위배하지 않는

다. 따라서 그 이후의 충돌 검사과정은 생략해도 된다. 물론 재순서화된 T_m이 *UpdateList*의 한 원소를 접근하려고 한다면 이는 재순서화에 의해 결정된 충돌 순서를 위배할 수 있으므로 트랜잭션이 철회된다.

O-Pre 알고리즘에 의해 읽기-전용 트랜잭션을 수행할 경우, 모든 읽기 연산이 수행되면 서버와 메시지 교환이 없이 트랜잭션이 정상 완료될 수 있다. 왜냐하면 수행 도중 충돌이 발생하더라도 재순서화에 의해 나머지 연산을 수행할 수 있으며(보조정리 1), 이후의 연산들도 재순서화 개념을 위배하지 않도록 수행되어지기 때문이다(정리 1 참조).

3.2.2 서버의 알고리즘

서버는 데이타베이스의 일관성을 유지하며, 사용자의 데이타 요구를 처리한다. 즉, 완료한 트랜잭션에 의해 수정된 데이타를 새로 갱신된 값으로 반영하며, 주기적인 브로드캐스트 프로그램을 작성하여 전송한다. 이 논문에서는 비록 갱신 트랜잭션과 관련된 특별한 가정을 하고 있지 않지만, 단지 그 트랜잭션들의 수행은 ACID 속성을 만족시킨다고 가정한다[3]. 물론 브로드캐스트되는 데이타들은 그 주기의 시작 시점의 값을 가진다.

정리 1. *O-Pre* 알고리즘은 직렬화가능하다.

증명. 트랜잭션이 수행도중 w-r 충돌이 발견된다면, 충돌 순서는 전위-재순서화에 의해 $T_m \rightarrow T_{si}$ (T_m : 이동 트랜잭션, T_{si} : 서버에서 완료한 갱신 트랜잭션($i \geq 1$))와 같이 결정되어 히스트리 H에 추가된다. 그 이후의 연산 수행으로 인하여 새로운 충돌, $T_{sj} \rightarrow T_m$ 이 발생하여 H에 추가된다고 가정하자. T_{sj}은 이미 서버에서 완료된 갱신 트랜잭션이며, 따라서 위의 새로운 충돌은, T_m이 T_{sj}가 갱신한 데이타 값을 읽었음을 의미한다. 이와 같은 경우가 발생하기 위해서는 다음 두 경우만이 가능하다. 첫째, T_{sj}가 T_{si}보다 직렬화 순서에서 먼저 완료한 경우이다. 이 경우는 T_m에서 충돌이 발견되지 않았을 때처럼 브로드캐스트 데이타 값을 읽은 것이므로, 아무런 문제가 발생하지 않는다. 그러므로 이 충돌관계는 무시할 수 있다. 두번째는 T_{sj}가 T_{si}와 동일한 주기 혹은 T_{si}보다 이후에 완료한 경우이다. 하지만 주기적인 *UpdateList*를 받게되면 T_m은 충돌 발생으로 인하여 철회된다. 즉, 이동 트랜잭션이 전위-재순서화되면, 이후의 연산을 비관적으로 수행함으로써 직렬화가능성을 침해할 수 있는 경우를 방지할 수 있다. 결과적으로, *O-Pre* 알고리즘에 의하여 수행될 경우에는 직렬화그래프에서 결코 순환(cycle)이 발생하지 않으므로, 항상 직렬화가능한 결과를 얻을 수 있다. □

3.3 복합 브로드캐스트 환경에서 *O-PreH* 알고리즘

이동 사용자가 단지 브로드캐스트되는 데이타를 기다리는 것이 아니라 서버에게 직접 데이타를 요구할 수 있도록 하기 위해서는 다음의 사항들을 고려해야 한다.

(1) 요구한 데이타 값을 어떻게 전송할 것인가?

(2) 데이타에 대한 어떤 버전 값을 전송할 것인가?

이후 위 사항에 대한 결론이 곧 우리의 알고리즘이 될 것이다. 데이타 전송 방식의 경우(첫번째 질문), 서버는 3가지 방법을 고려할 수 있다 : 유니캐스트(unicast), 동일 채널을 이용한 브로드캐스트, 다중 채널을 이용한 브로드캐스트. 각 방식이 장단점이 있지만, 본 논문에서는 두번째 방식, 즉 동일 채널을 이용한 브로드캐스트 방식에 의하여 서버는 클라이언트가 명시적으로 요구한 데이타 값을 전송하도록 한다. 왜냐하면, 3.2 절에서 제안한 재순서화 개념을 이용하는 알고리즘과 두번째 방식이 자연스럽게 적용될 수 있기 때문이다. 이 경우, 사용자가 명시적으로 요구한 데이타들은 *Push_Data*가 전송된 이후에 동일한 채널을 통하여 전송된다고 가정한다. 매 주기마다 요구하는 데이타의 양이 차이가 나지만, 요구한 데이타를 위해 할당하는 대역폭의 크기를 고정적으로 한다. 만약 어느 주기에 할당된 대역폭크기보다 더 많은 데이타 요구가 발생한다면 나머지 데이타는 다음 주기에 전송하도록 한다. 그 반대의 경우, 즉 할당된 대역폭이 남을 경우에는 널(null) 데이타를 전송한다. 이 경우, 일부 대역폭을 낭비할 수 있지만, 브로드캐스트 주기를 고정적으로 할 수 있으므로 여러 장점을 가질 수 있다.

일단 여러 데이타 요구 중에서 이번 주기에 처리될 데이타가 결정되면, 서버는 각 데이타에 대하여 어느 버전을 전송할 것인지가 결정되어야 한다. 먼저, 서버 입장에서 볼 때, 가장 최신 버전을 보내는 것이 가장 간단하다. 하지만 서버의 버전과 관련된 부하뿐만 아니라 사용자가 트랜잭션 일관성을 유지하기 위한 부하를 함께 고려한다면, 최신의 데이타를 보내는 것이 항상 최고의 고려사항이 되는 것은 아니다. 즉, 한 주기동안 *Push_Data*에 포함된 모든 데이타들은 일관된 상태였으므로, 이동 트랜잭션의 일관성 유지가 훨씬 수월하였었다. 따라서 서버는 *Push_Data*를 전파하기 시작하는 시점의 데이타베이스 상태에 해당하는 값을 전송하도록 한다. 이와 같은 동작을 지원하기 위해, 서버는 한 주기가 시작한 이후 사용자가 요구한 데이타가 완료된 트랜잭션에 의해 갱신되었다면, 주기의 시작 시점의 버전 값을 저장해두어야 한다. 이 버전은 오직 한 주기동안만 유지하면 된다.

한가지 언급할 점은, *Push_Data* 데이타들보다 훨씬 적은 빈도로 접근되거나 갱신되는 데이타들이 *Pull_Data*에 포함되기 때문에 실제 서버가 버전을 관리하기 위한 부하는 그리 크지 않을 수 있다는 것이다.

3.3.1 알고리즘

복합 브로드캐스트 모델과 관련된 주제들이 이전 절과 같이 결정되었다면, *O-PreH* 알고리즘은 아래에서 설명되는 일부분만 제외하고는 *O-Pre* 알고리즘과 동일하다.

차이가 나는 첫 번째는 읽기 연산, $r(x)$가 제출되는 경우이다. 사용자는 먼저 데이타 x의 종류를 결정해야 한다. 만약 *Push_Data*에 포함된다면, 단순히 브로드캐스트 채널에서 원하는 데이타 값을 얻으면 된다. 그렇지 않은 경우, 즉 *Pull_Data*의 한 원소라면, 서버에게 명시적으로 그 데이타 값을 요청하는 메시지를 보내야 한다.

O-PreH 알고리즘에서 차이가 나는 두번째 부분은, 트랜잭션 T_m이 데이타 x를 요청한 후 값을 받기 전에 주기적인 *BC.IR*로부터 데이타 x가 갱신되었다고 알려진 경우이다(그림 2). 이 경우, 트랜잭션의 상태, 즉 재순서화 여부에 따라 다음 연산이 결정된다. 먼저 T_m이 벌써 재순서화되어진 경우에는 철회시킨다. 왜냐하면 이후 데이타 x에 대한 읽기 연산을 수행하게 되면 재순서화 개념에 반하므로 직렬화가능성을 위배할 수 있기 때문이다. 그렇지 않고 어떤 충돌도 발견되지 않은 상황이라면, 트랜잭션은 재순서화되었다고 결정한 후, $r(x)$ 이후의 연산들은 비관적으로 수행시킨다.

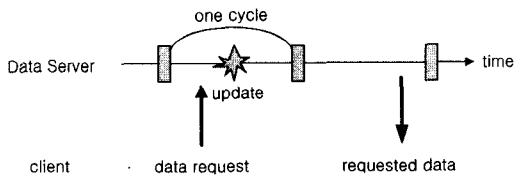


그림 2 문제가 발생할 수 있는 예

3.4 캐쉬

지역 캐쉬를 사용하게 되면 트랜잭션이 필요로 하는 데이타를 지역적으로 얻을 수 있으므로, 트랜잭션의 수행시간을 향상시킬 수 있을 뿐만 에너지를 효율적으로 사용할 수도 있다. 그리고 브로드캐스트 환경에서 *BC.IR*을 전송해준다면 캐쉬 데이타의 일관성을 유지하는데 도움이 된다. 그리고 상당수의 빈번하게 사용되는 데이타가 브로드캐스트 채널에서 얻을 수 있다면 선반입(auto-prefetching) 기법에 의해 캐쉬 데이타를 갱신할 수도 있다[12]. 이 논문에서 캐쉬와 관련하여 다음과 같은 가정을 그대로 적용한다: 서버에서의 갱신은 다음 주기 시작시점까지 보여지지 않으며, 따라서 한 주기동안의 모든 데이타들은 일관된 상태이다. 따라서 캐쉬 교체 알고리즘은 일반적인 *LRU(Least Recently Used)*을 채택하며, 이에 대하여서는 더 이상 자세히 다루지 않는다.

트랜잭션 T_m이 캐쉬 데이타 x를 접근하고자 할 경우에는 T_m이 이미 재순서화된 경우에만 조건 검사가 필요하다. 즉 캐쉬 데이타 x에 대하여 그 데이타가 *UpdateList*의 한 원소인 경우에는 T_m은 철회된다. 그외의 경우에는 아무런 조건 검사없이 캐쉬 데이타를 이용할 수 있다. 그 이유는 캐쉬 데이타를 이용하는 것은 브로드캐스트 채널에서 주기적인 데이타 혹은 명시적으로 요구한 데이타를 접근하는 것과 동일하기 때문이다.

4. 성능 평가

4.1 실험 모델

이 절에서부터 실험을 통하여 제안한 알고리즘의 성능을 평가하고자 한다. 그림 3은 제안하는 실험 모델을 보여주고 있다. 서버는 무효화 보고(*BC.IR*), *Push_Data*에 포함된 모든 데이타, 그리고 이전 주기동안 요청한 *Pull_Data*의 일부 데이타 순서대로 계속 주기적으로 브로드캐스트를 수행하고 있다 (물론 순수한 브로드캐스트 환경에서는 모든 데이타가 *Push_Data*의 원소이므로, *Pull_Data*가 없다). *Push_Data*에 대한 브로드캐스트 모델은 균등 (uniform)하다. 즉 서버는 한 주기동안 *Push_Data*에 포함된 모든 데이타를 한번씩 전송한다. 2절에서 언급하였듯이, 한 주기내의 모든 데이타들은 일관된 상태 -한 주기의 시작 시점의 데이타베이스의 상태 -를 가진다.

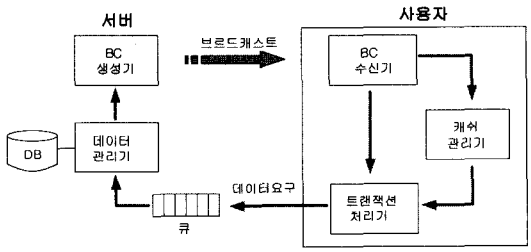


그림 3 실험 모델

실험에서 서버에서 데이타 접근(읽기 및 갱신)은 세타(θ) 인자를 가진 zipf 분포에 따라 발생한다. 즉, 서버가 관리하는 모든 데이타에 대하여 첫번째 데이타가 가장 높은 접근빈도를 가지며, 두번째 데이타가 다음으로 높은 접근 빈도를 가진다. 그리고 가장 마지막 데이타는 접근할 확률이 가장 낮다고 가정한다. *UpdateRate*는 서버가 관리하는 모든 데이타(*NumberOfData*)가 전송되는 동안 갱신되는 데이타의 개수를 나타낸다. 서버에서 갱신의 발생 분포는 균등하다. 이는 두 개의 연속된 갱신간의 시간 간격은 동일함을 의미한다. 이 가정이 의미를 가지는 것은 갱신은 서버에서만 발생하며, 사용자

가 (읽기 연산을 위해) 필요로 하는 데이타는 그 다음 주기 이후에 보여지기 때문이다.

서버에는 사용자의 명시적 데이타 요구를 저장하기 위해 하나의 큐를 두고 있으며, 이 큐내의 요구들은 first-in-first-out(FIFO) 방식으로 처리된다. 이 논문에서는 접속단절과 관련된 주제는 다루지 않는다.

이동 트랜잭션은 *AccessRange* 범위내의 데이타들(1-*AccessRange* 내의 데이타들)을 접근한다. 이 범위내에서 접근 확률은 역시 zipf 분포를 따른다. 사용자에서 빈번하게 접근되는 데이타와 서버에서 빈번하게 갱신되는 데이타들간의 차이를 나타내기 위해 *offset*이 사용된다. 예를 들면, *offset*이 50인 경우는, 사용자는 50번째 데이타를 가장 빈번하게 접근하고, 51번째 데이타를 그 다음 빈번하게 접근함을 의미한다. 그리고 49번째 데이타를 가장 덜 접근하게 된다. 이에 반하여 서버에서는 1번째 데이타가 가장 빈번하게 갱신된다.

각 사용자들은 *CacheSize* 크기의 지역 캐쉬를 관리하고 있다. 캐쉬 교체 알고리즘은 선반입을 이용한 LRU 방식을 따르며, 캐쉬 데이타의 일관성은 주기적인 *BC.IR*을 이용하여 수행한다.

실험에서 단위 시간은 서버에서 하나의 데이타를 전송하는데 걸리는 시간으로 설정하였으며, 이는 사용자가에서 하나의 읽기 연산을 수행하는데 걸리는 시간과 동일하다고 가정하였다. 단위 시간을 기준으로 수행하여야 할 모든 연산들의 시간을 상대적으로 결정하였다. 표 1은 실험에서 사용된 인자 및 기본 값을 보여준다.

제안한 *O-PreH* 알고리즘의 성능을 비교 평가하기 위해 *Invalidation-Only (IO)* 기법, *Multiversion with Invalidation (MI)* 기법, 그리고 *O-Pre* 기법을 함께 실험하였다[7,8,11]. 비교되는 기법들은 비록 순수한 브로드캐스트 환경을 가정하고 있지만, 거의 대부분의 환경이 제안한 알고리즘과 유사하며 또한 트랜잭션 수행의 정확성 기준으로써 직렬화가능성을 채택하고 있기 때문이다.

IO 기법에서 사용자는 주기적인 무효화 보고를 기반으로 하여 낙관적으로 수행중인 트랜잭션의 일관성을 검사한다. 만약 충돌이 발견되면 그 트랜잭션은 철회하고 재시작한다. *MI* 기법의 경우, 데이타의 갱신 비율에 따라 한 주기마다 각 데이타별로 1-4개의 이전 버전을 함께 브로드캐스트 하는 기법을 제안하였다. 그리고 수행중인 트랜잭션은 자신에게 적합한 버전 값을 이용하면 된다. 이 기법에서 한 주기의 길이는 다른 기법들보다 훨씬 길게 된다.

IO 및 *MI* 기법들은 역시 일반적인 캐쉬 기법을 사용한다.

4.2 실험 결과

제안한 *O-PreH* 알고리즘의 성능을 평가하기 위해, 주로 트랜잭션 당 수행시간에 초점을 맞추었다. 첫 번째 실험에서 트랜잭션 데이타 연산의 개수가 미치는 영향을 보이고자 한다. 이때 사용자의 수는 2000이며 *UpdateRate*는 1000으로 설정되었다. 그림 4에서, *offset* 인자는 0과 200으로 각각 설정되었다. 이전 절에서 언급하였듯이, *offset*이 양수 값을 갖는다는 것은 서버에서 빈번하게 갱신되는 데이타와 실험 대상인 사용자측에서 빈번하게 접근되는 데이타가 차이가 남을 의미한다.

결과에서 트랜잭션의 수행시간은 주로 브로드캐스트 주기에 영향을 받는데, 이는 대다수의 (혹은 모든) 연산들이 브로드캐스트되는 데이타를 이용하기 때문이다. 물

표 1 실험 인자

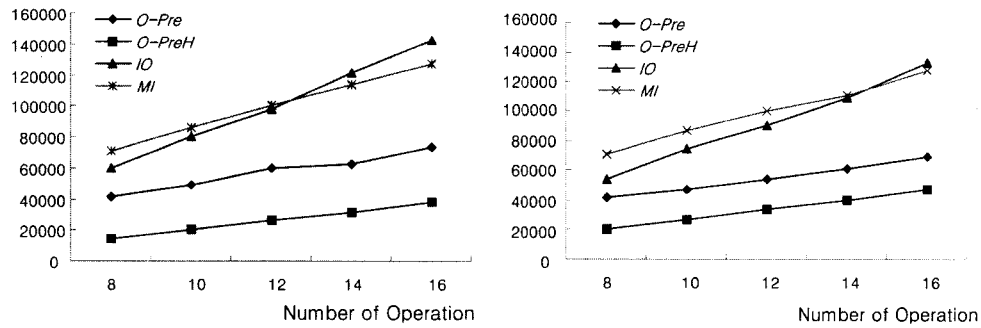
인자	기본값	설명
NumberOfData	10000	서버가 관리하는 데이타의 총개수
Push_Data	10000 (순수) 2000 (복합)	순수 및 복합 브로드캐스트에서 *Push_Data*의 크기
Pull_bandwidth	500	복합 브로드캐스트에서 한 주기에 전송되는 데이타 요구의 개수
UpdateRate	가변적	한 주기동안 서버에서 갱신되는 데이타의 개수
세타(θ)	0.95	zipf 분포의 인자
NumOfClient	가변적	이동 사용자의 수
CacheSize	500	지역 캐쉬 크기
AccessRange	평균 7000	이동 트랜잭션의 평균 접근 범위
Offset	0, 200	서버와 사용자간의 접근 차이
ReadTime	1	읽기 연산 시간 - 단위 시간
WriteTime	3	쓰기 연산 시간
BC_CheckTime	3	BC.IR을 처리하는 데 걸리는 시간
msgTransferTime	50	데이타 요청 메시지를 전송하는 시간
RestartTime	10	철회 후 재시작 전까지의 시간

론 데이타 갱신으로 인한 철회 회수에도 상당히 영향을 받는다. 그림에서 알 수 있듯이, 트랜잭션당 연산의 개수가 많아질수록 수행시간은 길어진다. 그러나 그 원인은 기법들마다 차이가 난다. *MI* 기법의 경우, 한 주기의 길이가 다른 기법들보다 훨씬 길기 때문에, 연산당 평균 접근 시간 역시 훨씬 길어지기 때문이다. 하지만 이 기법에서는 트랜잭션이 수행도중 서버에서 갱신이 발생하더라도 필요한 데이타의 버전을 브로드캐스트 채널에서 거의 다 얻을 수 있으므로, 철회되는 회수는 거의 없다. 순수하게 트랜잭션을 수행시키는 *IO* 기법의 경우, 연산의 개수가 증가하면, 여러 주기에 걸쳐서 트랜잭션을 수행해야 하며, 따라서 서버에서 갱신된 정보인 무효화 보고에 의해 철회되는 회수가 증가하게 된다. 따라서 평균 수행시간이 악화된다. 이 기법들과 비교해볼 때, *O-Pre* 및 *O-PreH* 기법들은 한 주기의 길이가 *IO*와 같거나 훨씬 짧으며, 재순서화 개념을 적용함으로써 철회가 발생하더라도 신속하게 이전 상태로 복귀할 수 있어서 상당히 향상된 수행시간을 보여주고 있다. 특히 *O-PreH* 기법에서는 트랜잭션이 *Push_Data*에 포함된 데이타에 접근할 경우, 대기 시간이 상당히 짧아진다.

그림 4(b)에서는 *offset* 값이 200으로 설정되었다. 따라서 사용자 시스템에서 빈번하게 접근하는 데이타가 *Push_Data*에 포함되지 않을 확률이 높으므로, *offset*이 0으로 설정된 경우(그림 4(a))보다 서버에게 메시지를 더 빈번하게 보내야 한다. 이러한 접근빈도의 차이를 좀 더 구체적으로 모델링하기 위해, 실험에서 사용자 총 수의 30%는 *offset* 200으로, 나머지 70%는 *offset* 0으로 설정하였다. 수행시간에서의 전반적인 증가 경향은 그림 4(a)와 유사하지만, 다소 차이가 난다. 먼저 *IO* 기법이나 *O-Pre* 기법은 서버에서 자주 갱신되는 데이타를 읽을 확률이 낮아지므로 충돌이 줄어들고, 따라서 전체적으로 철회 회수가 줄어서 수행시간이 다소 향상되었다. 이에 반해 *MI* 기법은 필요한 데이타의 모든 버전 값을 모두 얻을 수 있으므로, 성능에는 거의 영향이 없다. 하지만 *O-PreH* 기법은 *offset* 값이 양수로 설정되었다는 것은, 필요한 데이타가 *Pull_Data*에 포함된 경우가 많아지므로 서버에게 메시지를 보내고 기다려야 하는 회수가 증가한다. 따라서 전체적인 수행시간이 다소 나빠졌음을 알 수 있다. 그렇더라도 브로드캐스트의 한 주기 길이가 가장 짧기 때문에 전체적으로는 여전히 가장 향상된 수행시간을 보여준다.

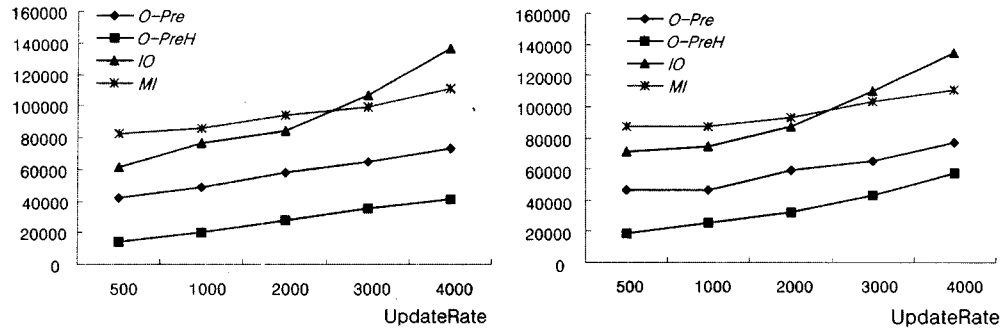
두번째 실험에서는 서버에서 데이타 갱신의 의미를 나타내는 *UpdateRate*가 미치는 영향을 실험하였다. 이 인자는 서버가 관리하는 모든 데이타 ([1..*NumberOfData*])를 전송하는 동안 갱신되는 데이타의 개수를 나타낸다. 따라서 *UpdateRate*를 동일하게 설정하였더라도, 복합 브로드캐스트에서 한주기당 갱신되는 데이타의 개수는, 순수한 브로드캐스트의 한주기 당 갱신되는 데이타의 개수보다 훨씬 적게 된다. 이 실험에서도 역시 *offset* 값에 따라 두가지로 나뉘어 진행하였다.

그림 5에서, *MI* 기법의 경우 *UpdateRate*가 증가하더라도 수행시간에 미치는 영향이 크지 않음을 알 수 있다. 또한 그림 4와 동일하게 *offset* 인자에도 큰 영향을 받지 않는다. 하지만 수행시간이라는 측면에서 볼 때, 그렇게 우수한 결과를 보이지는 않는다. 이는 브로드캐스트 채널에서 필요로 하는 거의 모든 데이타를 얻을 수 있기 때문에 철회 회수는 줄일 수 있지만, 서버가 전송하는 정보의 양이 많아지므로 단위 연산당 수행시간이 길어지기 때문이다. 그 외의 기법들은 정반대의 결과를 보이게 되는데, 한 주기당 *UpdateRate*의 값이 커질 수록 수행시간이 증가함을 알 수 있다. 특히 순수한 낙관적 수행을 기반으로 하는 *IO* 기법에 미치는 영향이 상당히 크다. *O-Pre* 및 *O-PreH* 기법들도 기본적으로 낙관적으로 수행하지만, 재순서화 개념에 의해 충돌이 발견될 때마다 철회하는 것이 아니므로 어느 정도 철회 회수를 줄일 수 있기 때문이다. 그림에서 *O-PreH* 기법



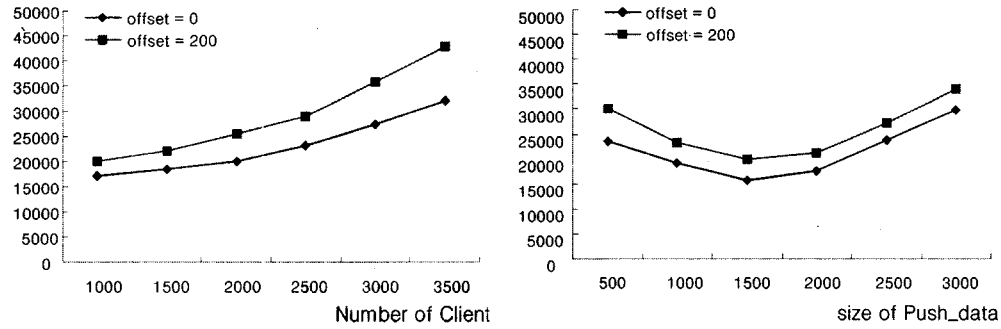
(a) offset = 0 (b) offset = 200

그림 4 트랜잭션당 연산의 개수에 대한 실험



(a) offset = 0 (b) offset = 200

그림 5 서버에서 데이타 갱신 비율에 따른 실험



(a) 10 operations, UpdateRate = 1500 (b) 10 operations, UpdateRate = 1500

그림 6 이동 사용자의 수 및 Push_Data의 크기에 대한 실험

은 서버와 비교적 동일한 접근 경향(*offset* = 0)과 낮은 갱신 비율 환경에서 높은 성능을 보여줌을 알 수 있다.

위 두 실험으로부터 다음과 같은 결론을 얻을 수 있다.

- 서버로부터 전송되는 정보의 양(무효화보고, 데이타 개수, 데이타 당 여러 버전 등)이 많아질수록, 트랜잭션의 수행에 필요한 많은 정보를 얻을 수 있다.
- 서버에서 브로드캐스트할 데이타(*Push_Data*)와 그렇지 않은 데이타(*Pull_Data*)를 적절하게 선택함으로써, 응답시간을 향상시킬 수 있다.

다음 두 실험은 O-PreH 기법의 속성을 이해하기 위하여 수행하였다. 그림 6(a)에서, 사용자의 수를 변화시켰다. 사용자의 수가 증가한다는 것은 복합 브로드캐스트 환경에서 *Pull_Data*에 접근하기 위해 서버로 메시지를 보내야 하므로, 무선 대역폭에 대한 경쟁이 발생한다. 또한 서버가 한 주기마다 처리해줄 수 있는 메시지의 수가 고정되었다고 가정하였으므로, 연산 수행시간이 길어지게 된다. 결과적으로 사용자의 수가 어느 수준 이하이며, 사용자와 서버간의 데이타 접근 경향이 유사할 경우에 *O-PreH* 기법의 성능이 최적으로 나타날 수 있다.

그림 6(b)에서는 전체 데이타 중에서 브로드캐스트 되는 데이타(*Push_Data*)의 크기 θ가 트랜잭션의 수행시간에 미치는 영향을 보여주고 있다. 만약 θ가 매우 작은 크기라면, 사용자들의 트랜잭션 수행에 필요한 대부분을 서버에게 요청해야 하므로 이전 6(a)와 동일한 이유에 의해 성능이 나빠지게 된다. 그 반대의 경우, 즉 θ가 아주 큰 경우(극단적인 경우는 순수한 브로드캐스트 환경)에는 사용자는 단지 무선 채널만을 기다리면서 필요한 데이타를 얻으면 되지만, 수행시간은 전송되는 데이타의 개수만큼 나빠지게 된다. 이 실험에서 1번째 데이타를 가장 빈번하게 접근하며, *NumberOfData*번째 데이타를 가장 덜 접근하며, 전체적인 접근 경향은 zipf 분포를 따른다고 가정하였다. 그림에서 보여지듯이, θ가 1500일 때 트랜잭션의 수행시간은 최적이 됨을 나타내고 있다. 최적의 θ 값은 실험 환경에 따라 차이가 날 수 있지만, 이 값을 적절하게 유지하는 것이 전체적인 처리량(throughput)을 향상시키는데 큰 영향을 미칠 수 있다.

5. 결론 및 향후 연구

이동 환경에서 제한된 자원이라는 측면에서 볼 때, 데

이타 브로드캐스트 방식은 큰 의미를 가진다고 할 수 있다. 그러나 서버가 이동 트랜잭션을 지원하기 위해 전송해야 할 정보의 양이 상당히 많거나 혹은 너무 트랜잭션의 수행을 제약한다면, 결국 전체적인 성능을 저하시키는 결과가 된다. 이 논문에서, 복합 브로드캐스트 환경에서 새로운 트랜잭션 수행 기법을 제안하였다. 이동 트랜잭션은 기본적으로 낙관적으로 수행되며, 트랜잭션의 일관성 유지에 대한 책임은 전적으로 사용자 측에서 담당하도록 한다. 특히, 재순서화 개념 및 복합 브로드캐스트 환경을 이용함으로써, 수행시간을 상당히 단축시킬 수 있었다.

하지만 아직 해결해야할 과제도 많다. 먼저 마지막 실험에서도 알 수 있듯이, 서버가 주어진 상황에 적합하게 브로드캐스트 프로그램을 생성할 수 있는 알고리즘을 개발하고자 한다. 특히 단순(flat) 브로드캐스트 모델 외에도 다양한 모델 개발이 병행하고자 한다[2]. 그리고 앞으로 실제 이동 컴퓨팅 환경에서 구체적으로 적용가능한 브로드캐스트 프로토콜을 개발하여 적용해보고자 한다.

참 고 문 헌

[1] T. Bowen, G. Gopal, G. Herman, T. Hickey, K. Lee, W. Mansfield, J. Raitz, and A. Weinrib, "The Datacycle Architecture," *Communications of the ACM*, vol. 35, no. 12, pp. 71-81, 1992.

[2] S. Acharya, R. Alonso, M. Franklin, and S. Zdonik, "Broadcast Disks: Data Management for Asymmetric Communications Environments," *in Proceedings of the ACM SIGMOD Conference*, pp. 199-210, 1995.

[3] T. Imielinski, S. Viswanathan, and B. Badrinath, "Energy Efficient Indexing on Air," *in Proceedings of the ACM SIGMOD Conference on Management of Data*, pp. 25-36, 1994.

[4] S. Acharya, M. Franklin, and S. Zdonik, "Balancing Push and Pull for data Broadcast," *in Proceedings of the ACM SIGMOD Conference*, pp. 183-194, 1997.

[5] J. Shanmugasundaram, A. Nithrakashyap, R. Sivasankaran, and K. Ramamritham, "Efficient Concurrency Control for Broadcast Environments," *in Proceedings of the ACM SIGMOD Conference on Management of Data*, pp. 85-96, 1999.

[6] P.A. Bernstein, V. Hadzilacos and N. Goodman, *Concurrency Control and Recovery in Database Systems*, Addison-Wesley, 1987.

[7] E. Pitoura and P. Chrysanthis, "Exploiting Versions for Handling Updates in Broadcast Disks," *in Proceedings of the International Conference on Very Large Data Bases*, pp. 114-125, 1999.

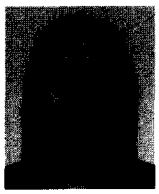
[8] E. Pitoura and P. Chrysanthis, "Multiversion Data Broadcast," *IEEE Transactions on Computers*, vol. 51, no. 10, pp. 1224-1230, 2002.

[9] V.C.S. Lee, S.H. Son and K.W. Lam, "On the Performance of Transaction Processing in Broadcast Environments," *in Proceedings of the Conference on Mobile Data Access*, pp. 61-70, 1999.

[10] S.S. Kim, S.K. Lee, and C. Hwang, "Using reordering technique for mobile transaction management in broadcast enviroments," *International Journal on Data & Knowledge Engineering*, vol. 45, no. 1, pp.79-100, 2003.

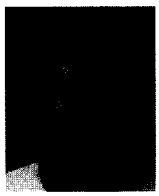
[11] K. Stathatos, N. Roussopoulos, and J.S. Baras, "Adaptive Data Broadcast in Hybrid Networks," *in Proceedings of the International Conference on Very Large Data Bases*, pp. 326-335, 1997.

[12] S. Acharya, M. Franklin, and S. Zdonik, "Prefetching from a Broadcast Disk," *in Proceedings of International Conference on Data Engineering*, pp. 276-285, 1996.



양 순 옥

1995년 고려대학교 자연자원대학 원예과학과 졸업. 2002년 고려대학교 대학원 컴퓨터학과(이학석사). 2002년부터 고려대학교 대학원 컴퓨터학과 박사과정 재학 관심분야는 이동 컴퓨팅 시스템에서의 사용자의 이동성 관리 및 고성능 컴퓨팅 등



김 성 석

1997년 고려대학교 전산과학과 졸업 1999년 및 2003년 고려대학교 컴퓨터학과 석사 및 박사 졸업. 2003년부터 서경대학교 전자상거래학과 교수로 재직. 관심분야는 분산시스템에서 캐싱 및 동시성 제어 기법, 이동 컴퓨팅 시스템에서 정보 서비스 등