방송 디스크 환경에서 클라이언트 트랜잭션을 위한 동시성 제어

정회원 조 행 래*

Concurrency Control for Client Transactions in Broadcast Disk Environments

Haengrae Cho* ***Regular Member***

요 약

방송 디스크는 다수의 이동 클라이언트에게 정보를 전파하는 통신 구조이다. 방송 디스크에서 서버는 데이타베이스에 저장된 모든 데이타를 연속적으로 방송하며, 클라이언트는 방송 채널을 검사하여 자신이 원하는 데이타를 수신한다. 이런 관점에서 방송 채널은 클라이언트가 데이타를 액세스할 수 있는 디스크의 역할을 담당한다. 본 논문에서는 방송 데이타가 서버에서 변경될 경우, 클라이언트에서 실행되는 트랜잭션이 판독하는 데이타의 정확성을 보장하기 위한 캐쉬 기반의 동시성 제어 기법(Cache Conscious Concurrency Control: C^4) 기법을 제안한다. C^4 기법은 서버로부터 동시성 제어를 위한 추가적인 정보의 방송을 최소화하며 클라이언트의 캐쉬를 효율적으로 사용함으로써 트랜잭션의 실행 시간을 단축시킬 수 있다는 장점을 갖는다.

ABSTRACT

Broadcast disks are suited for disseminating information to a large number of clients in mobile computing environments. In broadcast disks, the server continuously and repeatedly broadcasts all data items in the database to clients without specific requests. The clients monitor the broadcast channel and retrieve data items as they arrive on the broadcast channel. The broadcast channel then becomes a *disk* from which clients can retrieve data items. In this paper, we propose a *cache conscious concurrency control* (C^4) scheme to preserve the consistency of client transactions, when the values of broadcast data items are updated at the server. C^4 scheme is novel in the sense that it can reduce the response time of client transactions with minimal control information to be broadcast from the server. This is achieved by the judicious caching strategy of the clients.

Ⅰ. 서 론

방송 디스크는 이동 통신 환경에서 다수의 클라이언트에게 정보를 전파하기 위하여 제안된 구조로서, 서버는 데이타베이스에 저장된 모든 데이타를 연속적으로 반복하여 방송한다[1,3,5]. 클라이언트는 방송 채널을 감시하여, 원하는 데이타가 방송될 경우 방송 채널로부터 데이타를 수신한다. 이런 관점에서 방송 채널은 클라이언트가 데이타를 액세스할 수 있는 저장 장치(디스크)의 역할을 수행한다고 할 수 있다. 접속할 수 있는 클라이언트의 수에 제한이 없다는 장점으로 인해 방송 디스크는 이동 통신망을 이용한 경매나 전자 입찰과 같은 전자 상거래 응용 분야와 주식 거래, 그리고 기상 정보나 교통 정보 방송 분야와 같은 다양한 응용 분야에 활용되고 있다[3].

방송 디스크의 경우, 데이타 방송이 진행되는 동안 서버에서 실행되는 갱신 작업에 의해 데이타의

* 영남대학교 전자정보공학부 (hrcho@yu.ac.kr)
논문번호 : 010137-0601 접수일자 : 2001년 6월 1일

내용이 변경될 수 있으며, 변경된 데이타는 다음 사이클에서 방송된다. 만약 클라이언트에서 실행되는 응용 프로그램이 다른 사이클에서 방송된 여러 개의 데이타를 수신할 경우, 데이타들간의 변경 시점의 차이로 인하여 응용 프로그램의 정확성이 보장되지 않을 수 있다. 따라서 클라이언트에서 실행되는 응용 프로그램의 정확성을 유지하기 위한 동시성 제어 기법이 필요하다[6,7,9]. 이때 서버에서 실행되는 작업을 "서버 트랜잭션"이라 하며, 클라이언트에서 실행되는 응용 프로그램을 "클라이언트 트랜잭션"이라 정의한다. 데이타에 대한 변경은 서버 트랜잭션의 실행에서만 발생한다고 가정하며, 클라이언트 트랜잭션은 방송 채널에서 데이타를 판독만 한다고 가정한다.

본 논문에서는 방송 데이타가 서버에서 변경될 경우, 클라이언트 트랜잭션이 판독하는 방송 데이타의 정확성을 보장하기 위한 "캐쉬 기반의 동시성 제어" (Cache Conscious Concurrency Control: C^4) 기법을 제안한다. C^4 기법은 클라이언트의 캐쉬 메모리를 최대한 이용함으로써 서버로부터 동시성 제어를 위한 추가적인 정보의 방송없이 클라이언트 트랜잭션의 처리 성능을 향상시킬 수 있다는 장점을 갖는다.

본 논문의 구성은 다음과 같다. 먼저 2절에서는 방송 디스크에서 동시성 제어 문제 및 기존 연구들을 살펴보고, 3절에서는 본 논문에서 제안한 C^4 기법의 동작 과정을 자세히 설명한다. 4절에서는 C^4 기법의 성능을 평가하기 위하여 개발한 실험 모형에 대해 설명하고 실험 결과를 분석한다. 마지막으로 5절에서 결론을 맺는다.

Ⅱ. 방송 디스크에서 동시성 제어

본 절에서는 방송 디스크의 데이타 방송 모형과 동시성 제어 문제를 소개하고, 방송 디스크에서 기존에 제안된 동시성 제어 기법들을 살펴본다.

1. 방송 모형

방송 디스크에서 서버는 주기적으로 다수의 클라이언트에게 데이타들을 방송한다. 방송의 각 주기를 사이클이라 하며, 각 사이클에서 방송되는 내용을 "방송 프로그램"이라 한다. 방송 프로그램은 데이타의 예상 참조 수에 따라 몇 개의 그룹으로 나누어 편성되는데, 기본 개념은 클라이언트에 의해 빈번히 참조되는 그룹들을 상대적으로 자주 방송하여 클라이언트의 평균 대기 시간을 줄이도록 하는 것이다. 이때 각 그룹을 하나의 논리적인 디스크로 간주할 경우 방송 프로그램은 여러 개의 디스크로 구성되며, 방송 프로그램 내에서 디스크의 방송 빈도수가 상이하므로 논리적으로 다양한 액세스 속도를 지원하는 디스크들이 존재하게 되는 것이다. 방송 프로그램의 작성 예가 그림 1에 나타난다.

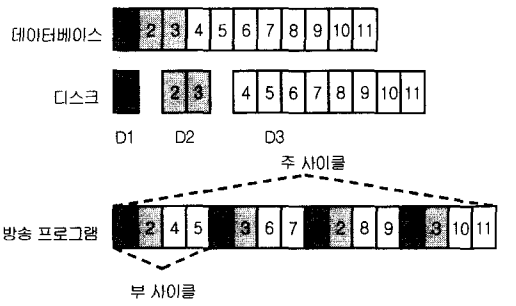


그림 1. 방송 프로그램의 작성 예[1]

그림 1에서 서버의 데이타베이스에는 1부터 11까지 11개의 데이타가 저장되어 있으며, 데이타의 예상 참조 수에 따라 3개의 디스크로 나누어진다. 디스크 D1은 가장 빈번히 참조되며, D3은 참조 수가 가장 낮다고 가정한다. 그 결과로 작성된 방송 프로그램의 경우, D1과 D2, 그리고 D3의 방송 빈도수는 4:2:1이 됨을 알 수 있다. 이때 방송 빈도수가 가장 높은 D1이 방송되는 주기를 "부 사이클"이라 하며, 방송 빈도수가 가장 낮은 D3이 방송되는 주기를 "주 사이클"이라 한다. 본 논문에서 사용하는 사이클은 주 사이클을 의미한다고 가정한다. 그리고, 사이클마다 일련 번호가 주어진다고 가정하며, k번째 사이클의 일련 번호를 B_k로 표기한다. 이때 사이클의 일련 번호는 단조 증가한다고 가정한다. 즉, $B_{k+1} > B_k$ 이다.

데이타가 방송되는 동안에 서버 트랜잭션은 데이타 값을 변경할 수 있으며, 변경된 데이타 값은 다음 사이클에서 방송된다. 따라서, 사이클 k에서 방송되는 데이타 값은 k 이전의 사이클에서 변경된 값들 중에서 가장 최신에 변경된 값이다. 본 논문에서는 데이타 d의 변경 시점을 표현하기 위하여 타임스탬프 TS(d)도 d와 같이 전송된다고 가정한다. d의 현재 값이 사이클 k에서 변경된 결과일 경우, TS(d)는 B_{k+1}로 설정된다.

2. 동시성 제어 문제

각 클라이언트는 판독 전용의 클라이언트 트랜잭

션을 실행하며, 클라이언트 트랜잭션은 방송 채널로부터 방송되는 방송 프로그램 상에서 원하는 데이타를 판독한다. 이때 방송의 특성상 데이타 판독은 순차적으로 진행되며 클라이언트 트랜잭션이 판독할 데이타들의 집합을 트랜잭션의 실행 전에 파악하는 것이 곤란하므로, 클라이언트 트랜잭션은 여러 사이클에 걸쳐 데이타를 판독할 수 있다. 예를 들면, 그림 1의 방송 프로그램에서 클라이언트 트랜잭션이 데이타 5를 판독한 후 데이타 4를 판독할 경우, 현재 사이클에서는 데이타 4가 이미 방송되었으므로 다음 사이클까지 대기하여야 한다. 클라이언트 트랜잭션이 여러 사이클에 걸쳐 실행될 경우 잘못된 실행 결과가 발생할 수 있는데 그림 2에 그 예가 나타난다.

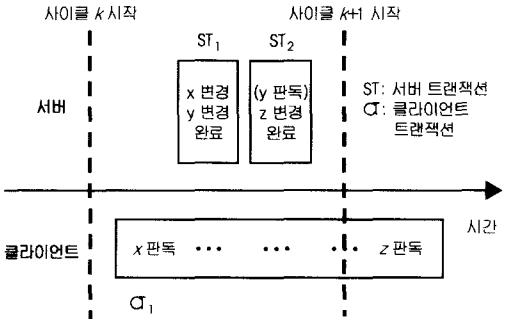


그림 2. 잘못된 트랜잭션 실행의 예

그림 2에서 클라이언트 트랜잭션 CT_1이 데이타 x를 판독한 후 서버 트랜잭션 ST_1이 x와 y를 변경하므로, CT_1의 실행 순서는 ST_1에 우선한다. 즉, $CT_1 \rightarrow ST_1$ 관계가 성립한다. 그리고, 서버 트랜잭션 ST_2는 ST_1이 변경한 y 값을 참조하여 z를 변경하였으므로, $ST_1 \rightarrow ST_2$의 관계가 성립한다. ST_1과 ST_2는 사이클 k+1이 시작되기 전에 완료되었으므로 변경된 x, y, z 값이 사이클 k+1에서 방송되며, CT_1이 판독하는 z 값은 ST_2에 의해 변경된 값이다. 그 결과 $ST_2 \rightarrow CT_1$의 관계가 성립되는데, 전체적으로 $CT_1 \rightarrow ST_1 \rightarrow ST_2 \rightarrow CT_1$의 실행 순서가 성립된다. 트랜잭션 실행 순서에 대해 순환 관계가 존재하므로 CT_1의 실행은 잘못되었으며, CT_1을 철회한 후 재실행해야 한다.

방송 디스크에서 동시성 제어의 문제점은 그림 2와 같은 잘못된 트랜잭션 실행의 경우와 정확한 트랜잭션 실행을 구분할 수 없다는 것이다. 예를 들면, 그림 2에서 ST_2가 y 값을 판독하지 않고 z 값을 변경한다면 $ST_1 \rightarrow ST_2$의 관계가 성립하지 않는다. 이 경우 순환 관계가 존재하지 않으므로 CT_1을 철회할 필요없이 계속 실행할 수 있다. 그러나, 클라이언트는 서버와의 추가적인 통신없이는 ST_2에 대한 실행 내용을 알 수 없으며 항상 최악의 경우를 고려해야 하므로, CT_1을 철회한 후 재실행해야 한다.

3. 기존 연구

방송 디스크에서 클라이언트 트랜잭션의 정확성을 유지하기 위한 동시성 제어 기법들이 최근 들어 제안되고 있다[6,7,9]. [6]의 경우, 서버는 트랜잭션 실행 그래프와 각 데이타를 가장 최근에 변경한 트랜잭션 식별자를 방송 프로그램과 함께 방송한다. 예를 들면, 그림 2의 사이클 k+1 시작 시점에서 서버는 $[ST_1 \rightarrow ST_2, (x, ST_1), (y, ST_1), (z, ST_2)]$의 정보를 방송한다. 이를 이용하여 CT_1의 실행 결과 순환 관계가 성립할 경우에만 CT_1을 철회할 수 있다. [7]의 경우에는 데이타의 현재 값 외에 이전 값도 같이 방송함으로써 불필요한 클라이언트 트랜잭션의 철회율을 줄였다. 예를 들면, 그림 2에서 CT_1은 사이클 k+1 이전에 변경된 z 값이 방송될 경우 이를 판독함으로써 철회되지 않고 계속 실행될 수 있다. [9]에서는 서버가 방송 프로그램 외에 제어 행렬 $\mathbf{C}$를 같이 방송한다. 임의의 데이타 d_1과 d_2에 대해 $\mathbf{C}(d_1, d_2)$는 d_2를 가장 최근에 갱신한 서버 트랜잭션보다 실행 순서가 앞서는 서버 트랜잭션 중에서 d_1을 가장 최근에 갱신한 트랜잭션이 완료한 사이클의 일련 번호를 나타낸다. 예를 들면, 그림 2에서 $\mathbf{C}(x, z)$는 B_k인데, 그 이유는 z를 가장 최근에 갱신한 트랜잭션은 ST_2이며 x를 최근에 갱신한 ST_1 트랜잭션에 대해 $ST_1 \rightarrow ST_2$의 관계가 성립하고 ST_1은 사이클 k에서 완료되었기 때문이다. CT_1이 판독한 x의 타임스탬프 $TS(x)$가 $\mathbf{C}(x, z)$보다 크지 않으므로 z를 판독할 때 CT_1은 철회한 후 재실행되어야 한다.

기존에 제안된 동시성 제어 기법들의 공통점은 서버가 동시성 제어 정보를 추가로 전송하여 클라이언트 트랜잭션의 철회율을 줄인다는 것이다. 그러나, 동시성 제어 정보는 데이타의 수나 서버 트랜잭션의 수에 비례하여 크기가 증가하므로, 대규모 데이타베이스 응용 분야의 경우 방송 대역폭의 상당 부분을 차지하게 된다. 그 결과 클라이언트 트랜잭션이 일반적인 방송 데이타를 판독하기 위한 대기 시간이 길어지며 전체적으로 트랜잭션 작업을 완료할 때까지의 응답 시간이 길어진다.

Ⅲ. 캐쉬 기반의 동시성 제어

본 절에서는 캐쉬 기반의 동시성 제어(Cache Conscious Concurrency Control: C^4) 기법을 제안한다. C^4 기법은 동시성 제어를 위해 최소한의 정보만을 방송함으로써 일반 데이타의 방송에 대역폭의 대부분을 사용한다. 뿐만 아니라, 클라이언트의 캐쉬 메모리를 이용함으로써 트랜잭션의 철회율을 줄이며, 트랜잭션이 철회될 경우에도 빠른 재실행을 보장한다. 본 절에서는 C^4 기법의 동시성 제어 과정과 캐쉬의 활용 정책에 대해 설명하도록 한다.

1. 동시성 제어

C^4 기법의 동시성 제어 과정은 [6]에서 제안한 다중버전 무효화 기법을 단일 버전으로 수정하여 데이타의 이전 값은 방송하지 않고 최신 값만 방송한다고 가정한다. 그리고, 동시성 제어를 위해 각 사이클의 시작 시점마다 무효화 보고서(INV)만 방송하는데, 사이클 i+1에서 방송되는 INV_{i+1}은 사이클 i가 진행되는 동안 서버에서 갱신된 데이타들의 "식별자"를 포함한다. 데이타의 최신 값만 방송하며 INV에 포함되는 식별자는 실제 데이타보다 크기가 매우 작으므로, 동시성 제어를 위해 추가로 사용하는 방송 대역폭을 최소화할 수 있다.

클라이언트 트랜잭션 R에 대해 클라이언트는 판독 집합 $RS(R)$과 무효화 타임스탬프 $TS(R)$을 유지한다. $RS(R)$은 R이 현재까지 판독한 데이타들의 식별자 집합이며, $TS(R)$은 R이 처음으로 무효화된 시점의 사이클 번호를 나타낸다. R이 시작될 때, $RS(R)$은 공집합으로 설정되며 $TS(R)$은 최대 값으로 초기화된다. 동시성 제어를 위한 기본적인 동작 과정은 다음과 같다.

- R이 방송 데이타 d를 판독하려고 할 경우, $TS(R) > TS(d)$이면 판독 가능하다. d를 판독한 후, $RS(R)$은 $RS(R) \cup \{d\}$로 설정된다. 만약 $TS(R) \leq TS(d)$이면 R은 철회한 후 재실행되어야 한다.
- 사이클 i의 시작 시점에서 $TS(R)$이 최대 값이며 $RS(R) \cap INV_i \neq \emptyset$이면 $TS(R)$은 i의 일련 번호 B_i로 설정된다.

위 알고리즘의 단점은 트랜잭션 철회율이 높다는 것이다. 예를 들면, 그림 2의 사이클 k+1 시작 시점에서 $x \in INV_{k+1}$이며 $TS(CT_1)$이 최대 값이므로, CT_1은 무효화되고 $TS(CT_1)$은 B_{k+1}로 설정된다. 이후, CT_1이 z를 판독할 경우 $TS(CT_1) = TS(z)$이므로, CT_1은 철회한 후 재실행되어야 한다. 이때 주의할 것은, ST_2의 y 판독 여부에 관계없이 CT_1은 철회된다는 것이다.

2. 캐싱 정책

클라이언트는 최근에 판독한 데이타들을 지역 메모리에 캐싱한다. 만약 다음에 판독할 데이타가 캐싱되어 있다면 해당 데이타가 방송될 때까지 대기하지 않고 캐쉬에서 바로 판독할 수 있으므로 클라이언트 트랜잭션의 실행 시간을 줄일 수 있다[1,2].

본 논문에서는 방송 데이타를 캐쉬에 저장할 때 해당 데이타의 타임스탬프도 같이 저장한다고 가정한다. 그리고, 캐쉬에 저장된 데이타가 INV에 포함될 경우 클라이언트는 해당 데이타의 "old" 플래그를 설정한다. "old" 플래그가 설정된 데이타에 대해 클라이언트는 해당 데이타의 현재 값을 방송 채널에서 자동적으로 판독하여 캐쉬에 저장한 후, "old" 플래그를 해제한다. 캐쉬에 저장된 임의의 데이타 d에 대해 $d_{current}$는 "old" 플래그가 설정되지 않을 경우의 d 값을 의미하며, d_{old}는 "old" 플래그가 설정되었을 경우의 d 값을 의미한다고 가정한다. 자동 판독이 완료된 후 d_{old}는 $d_{current}$로 대체되므로, d에 대해서는 $d_{current}$나 d_{old}의 값 중에서 하나만 캐쉬에 저장된다.

캐쉬에 저장된 데이타를 이용하여 3.1절에서 설명한 동시성 제어의 문제점을 해결하기 위한 기본 정책을 아래에 정리한다.

1. 클라이언트 트랜잭션 R에 대해 클라이언트는 $RS(R)$에 포함된 모든 데이타들을 캐쉬에 저장한다. 만약 $RS(R)$에 포함된 d가 무효화될 경우, 즉 $d \in INV$일 경우, d의 현재 값을 방송 채널에서 자동적으로 판독한다.
2. R이 무효화되었을 경우, $TS(R) > TS(d_{old})$인 d_{old}가 캐쉬에 저장되어 있다면 R은 방송 데이타를 기다리지 않고 d_{old}를 즉시 판독한다.

첫 번째 정책의 기본 개념은 R이 철회되어 재실행할 경우, 철회될 당시의 $RS(R)$에 포함된 데이타의 최신 값이 캐쉬에 저장되어 있다면 철회 시점까지는 방송 채널에서 데이타를 판독할 필요가 없으므로 빠른 재실행이 가능해진다는 것이다. 따라서 트랜잭션의 재실행은 전체 트랜잭션의 실행 시간에

큰 영향을 미치지 않을 수 있다. 예를 들면, 그림 2에서 CT_1이 z를 판독할 때, x의 현재 값($x_{current}$)이 이미 캐쉬에 저장되었다고 가정하자. z를 판독하여 $z_{current}$ 값을 캐쉬에 저장한 후, CT_1은 $x_{current}$와 $z_{current}$를 이용하여 즉시 재실행할 수 있다.

두 번째 정책의 기본 개념은 R이 판독하는 데이타의 현재성을 다소 희생하여 R의 철회율을 줄인다는 것이다. 3.1절의 동시성 제어의 경우, R이 사이클 i에서 무효화되었으며 $j \geq i$인 사이클 j에서 d가 무효화되었다면, R이 d를 판독할 경우 $TS(R) \leq TS(d)$이므로 R은 철회되어야 한다. 그러나, d의 현재 값이 방송되기 전까지는 캐쉬에 d_{old}가 저장되어 있으며, 이 기간 중에는 R에게 d_{old}를 판독하도록 한다면 $TS(R) > TS(d_{old})$일 경우 R은 철회될 필요가 없다. 예를 들면, 그림 2에서 CT_1이 z를 판독할 때 z_{old}가 캐쉬에 저장되어 있으며 $TS(z_{old})$는 B_k라고 가정하자. 이는 서버 트랜잭션 ST_0가 존재하여 z를 갱신한 후 사이클 k이전에 완료하였다는 것을 의미한다. $TS(CT_1)$은 B_{k+1}이므로 z_{old}를 판독함으로써 CT_1은 철회되지 않고 계속 실행할 수 있다. 이때 주의할 것은 ST_0는 사이클 k이전에 완료하였으므로 $ST_1 \rightarrow ST_0$나 $ST_2 \rightarrow ST_0$의 관계가 성립하지 않고, 따라서 실행 순서에 대한 순환 관계가 성립하지 않으며 CT_1의 실행 결과는 정확하다는 것이다.

그림 3에 동시성 제어 과정과 캐싱 정책을 통합한 C^4 기법의 동작 과정을 정리한다.

기존의 동시성 제어 기법과 비교할 때 C^4 기법의 장점은 다음과 같다. 먼저 동시성 제어를 위해 INV만 추가로 전송하면 된다. 따라서 C^4 기법의 경우 순수한 데이타를 전송하는데 대부분의 방송 대역폭을 사용할 수 있고, 그 결과 클라이언트의 방송 데이타에 대한 대기 시간을 줄일 수 있다. 다음으로, 데이타에 대한 여러 개의 값을 저장하는 다중버전 캐싱 알고리즘[6] 과는 달리 C^4 기법은 하나의 데이타 값만 캐쉬에 저장한다. 따라서, 주어진 캐쉬 공간에 보다 많은 수의 데이타들이 저장될 수 있고, 캐쉬 적중율도 높아질 수 있다. 마지막으로, 클라이언트 트랜잭션의 철회율을 줄이고자 하는 기존 기법들과는 달리 C^4 기법은 트랜잭션의 전체 실행 시간을 단축하는 것이 주요 목적이다. 이러한 관점에서 C^4 기법과 기존 기법들과의 혼합 기법을 설계하는 것이 가능하다. 예를 들면, [6]에서 제안한 다중버전 동시성 제어 기법에서 다중버전의 수를 다소 줄이고 그에 따른 철회율의 증가를 C^4 기법의 캐싱 정책으로 보상하는 방법 등을 들 수 있다.

- 사이클 i의 시작 시점에서 $TS(R)$이 최대 값이며 $RS(R) \cap INV_i \neq \emptyset$이면 R은 무효화되고 $TS(R)$은 i의 일련 번호 B_i로 설정된다.
- 클라이언트 트랜잭션 R이 데이타 d를 판독하려고 할 경우, 아래 조건을 차례대로 검사한다.
  1. 만약 캐쉬에 $d_{current}$가 저장되어 있으며 $TS(R) > TS(d_{current})$이면 R은 $d_{current}$를 판독한다. $TS(R) \leq TS(d_{current})$일 경우, R은 철회되어 재실행한다.
  2. 만약 R이 무효화되었으며 $TS(R) > TS(d_{old})$인 d_{old}가 캐쉬에 저장되어 있다면 R은 d_{old}를 판독한다. $TS(R) \leq TS(d_{old})$일 경우, R은 철회되어 재실행한다.
  3. R은 d가 방송될 때까지 대기한다. 만약 $TS(R) > TS(d)$이면 R은 d를 판독한다. 그렇지 않을 경우 R은 철회되어 재실행한다.
- 클라이언트는 $RS(R)$에 속하는 모든 데이타들을 캐쉬에 저장한다. 만약 $RS(R)$에 속하는 데이타 d가 무효화될 경우 클라이언트는 d의 현재 값을 방송 채널에서 자동적으로 판독한다. d의 현재 값이 아직 방송되지 않은 시점에서 R이 철회된다면, d를 판독할 때까지 R의 재실행을 지연시킨다.

그림 3. C^4 기법의 동작 과정

Ⅳ. 성능 평가

본 논문에서는 시뮬레이션을 이용하여 C^4 기법의 성능을 평가한다. 성능 평가 대상으로는 방송 디스크 환경에서 가장 좋은 성능을 갖는 것으로 알려진 다중버전 동시성 제어 기법[6]을 선택하였다. 구체적으로 실험을 위하여 4가지 동시성 제어 기법(C^4P, C^4C, MV2, MV4)을 구현하였다. C^4P는 그림 3에 나타난 C^4 기법의 동작 과정을 완전히 구현한 기법이며, C^4C는 캐쉬에 저장된 데이타의 이전 값은 사용하지 않도록 구현하였다. 따라서, C^4C에서는 클라이언트 트랜잭션이 가장 최신의 값만을 판독할 수 있지만 빈번한 트랜잭션 철회가 발생할 수 있다. MV2와 MV4는 다중버전 동시성 제어 기법을 구현한 것으로 MV2는 이전 두 개의 사이클에서 변경된 데이타 값을 추가로 방송하는 기법이며, MV4는 이전 네 개의 사이클에서 변경된 값을 방송한다. 이전 사이클에서 변경된 값들은 방송 빈도수가 가장 낮은 별도의 방송 디스크에서 방송되도록 구현하였다.

본 절에서는 먼저 실험 모형을 설명한 후, 4가지 동시성 제어 기법에 대한 실험 결과를 분석하도록 한다.

1. 실험 모형

시뮬레이션은 미국의 MCC에서 개발한 CSIM 언어[8]를 이용하여 수행하였으며, 실험에 사용된 입력 매개 변수를 표 1에 정리한다. 각 매개 변수의 구체적인 값들은 [2]에서 주로 참조하였다.

서버는 1부터 ServerDBSize까지의 데이타들을 주기적으로 방송하며, 동시성 제어 기법에 따라 두 가지의 방송 프로그램이 존재한다. C^4P와 C^4C는 3개의 방송 디스크를 가정하며, 디스크들간의 방송 빈도수는 5:3:1로 결정하였다. MV2와 MV4의 경우에는 이전 사이클에서 변경된 값들을 방송하기 위하여 추가적으로 하나의 디스크를 더 사용하며, 디스크들간의 방송 빈도수는 10:6:2:1로 결정하였다. 그 결과, MV2와 MV4에서의 방송 프로그램은 C^4P와 C^4C에 비해 약 두배 정도 길어진다. 방송 프로그램의 길이 차이에 따라 무효화 메시지의 전송 빈도수가 다를 수 있으므로 이를 보상하기 위해 C^4P와 C^4C에서는 두 번의 방송 사이클마다 무효화 메시지가 방송되도록 구현하였다. 무효화 메시지에는 해당 사이클동안 갱신된 Nupdate 개의 데이타 식별자가 저장된다고 가정한다. 각 방송 디스크에 포함된 데이타 항목들이 갱신될 비율은 동일하다고 가정하였으며, 갱신할 데이타를 저장하는 방송 디스크는 θ를 인자로 하는 Zipf 분포[4]에 따라 선택하였다. 그 결과 상대적으로 크기가 작은 디스크 1의 갱신 비율이 가장 높으며, 디스크 2와 3으로 갈수록 갱신 비율이 낮아진다.

클라이언트는 1부터 ReadRange 사이의 방송 데이타를 판독한다. 판독할 데이타는 θ를 인자로 하는 Zipf 분포에 따라 결정된 방송 디스크 내에서 무작위로 선택된다. 데이타를 판독한 클라이언트 트랜잭션은 ThinkTime만큼 지연된 후 다음 데이타를 판독하며, 트랜잭션이 판독하는 데이타 수는 TranSize ± TranSize × TRSizeDev 사이의 일양 분포를 따른다. UpdateOffset 변수는 클라이언트와 서버 사이의 액세스 분포 차이를 표현하기 위해 사용하는데, UpdateOffset 값이 K일 경우 서버에서의 갱신 데이타 분포는 클라이언트가 주로 판독하는 데이타와 K만큼 차이를 갖는다. 따라서 UpdateOffset 값이 0일 경우 클라이언트가 주로 판독하는 데이타들이 서버에서 갱신될 가능성이 가장 높으며, 트랜잭션 철회가 빈번히 발생한다. UpdateOffset 값이 증가함에 따라 클라이언트에서 판독하는 데이타들이 서버에서 갱신될 가능성이 줄어든다. 각 클라이언트는 CacheSize 만큼의 데이타를 저장할 수 있는 캐쉬를 가지며, 캐쉬는 LRU 정책에 따라 관리된다.

모의 실험에서 사용된 주요 성능 지수는 트랜잭션 응답 시간이다. 트랜잭션 응답 시간은 트랜잭션이 생성된 후 완료될 때까지의 시간을 의미하며, 큐에서의 대기 시간 및 트랜잭션이 철회되어 재실행되는 시간도 모두 포함된다. 이때 시간은 방송 단위의 수로 측정하였는데, 방송 단위는 하나의 데이타

표 1. 입력 매개 변수

서버 변수		
ServerDBSize	방송될 데이타의 수	1000
NumDisks	방송 프로그램을 구성하는 디스크의 수	C4P, C4C - 3, MV2, MV4 - 4
DiskSize	방송 디스크의 크기(Disk1, Disk2, Disk3)	80, 170, 750
DiskFreq	방송 프로그램에서 디스크의 방송 빈도수	C4P, C4C - 5 : 3 : 1
	(Disk1 : Disk2 : Disk3 : Disk4)	MV2, MV4 - 10 : 6 : 2 : 1
Nupdate	무효화 메시지에 포함된 갱신 데이타의 수	50, 100
θ	Zipf 분포 변수	0.95
클라이언트 변수		
ReadRange	클라이언트가 판독하는 데이타 범위	500
θ	Zipf 분포 변수	0.95
ThinkTime	판독한 데이타에 대한 처리 시간 (방송 단위)	2
TranSize	트랜잭션이 판독하는 데이타 수	20
TRSizeDev	트랜잭션 길이의 편차	0.1
UpdateOffset	클라이언트와 서버 사이의 액세스 분포 차이	0, 40, 85, 170, 320
CacheSize	클라이언트의 캐쉬에 저장 가능한 데이타 수	125

를 방송하는데 소요되는 실제 시간을 의미한다. 그리고 보조 성능 지수로 트랜잭션 철회율과 캐쉬 적중율 등을 사용하도록 한다.

2. 실험 결과

본 절에서는 UpdateOffset과 Nupdate 값을 변화시키면서 실험한 4가지 동시성 제어 기법의 성능 평가 결과를 분석하도록 한다. Nupdate의 값이 각각 50과 100일 경우의 트랜잭션 응답 시간이 그림 4에 나타난다.

UpdateOffset의 값이 증가할수록 모든 기법들의 성능이 증가하는데, 그 이유는 클라이언트 트랜잭션이 판독한 데이타가 무효화될 가능성이 줄어들기 때문이다. C^4P와 C^4C의 경우 그림 5에 나타난 트랜잭션 철회율은 MV2와 MV4보다 높지만 트랜잭션 응답 시간은 빠른 것으로 나타났다. 그 이유는 철회되어 재실행하는 트랜잭션이 액세스할 데이타가 캐쉬에 이미 저장되어 있으므로 직전의 철회 시점까지는 방송 데이타에 대한 대기 시간이 필요하지 않고, 그 결과 빠른 재실행이 가능하기 때문이다. 이는 그림 6에 나타나듯이 C^4P와 C^4C에서의 캐쉬 적중율이 MV2와 MV4보다 높은 결과에서 판단할 수 있다.

C^4P에 비해 C^4C의 성능이 다소 나쁜 것으로 나타났으며, UpdateOffset이 작을수록 성능 차이는 큰 것으로 나타났다. 그 이유는 C^4C의 경우 항상 최신의 데이타만 판독하므로 트랜잭션 철회율이 C^4P에 비해 높기 때문이다. 특히 UpdateOffset이 0일 경우 판독한 데이타가 무효화될 가능성이 매우 높고, 그 결과 C^4C에서의 트랜잭션 철회율이 Nupdate에 따라 각각 300%와 100% 정도로 매우 큰 값으로 나타났다. 이에 비해 C^4P는 캐쉬에 저장된 데이타의 이전 값을 이용함으로써 트랜잭션 철회율을 상당히 낮출 수 있었다.

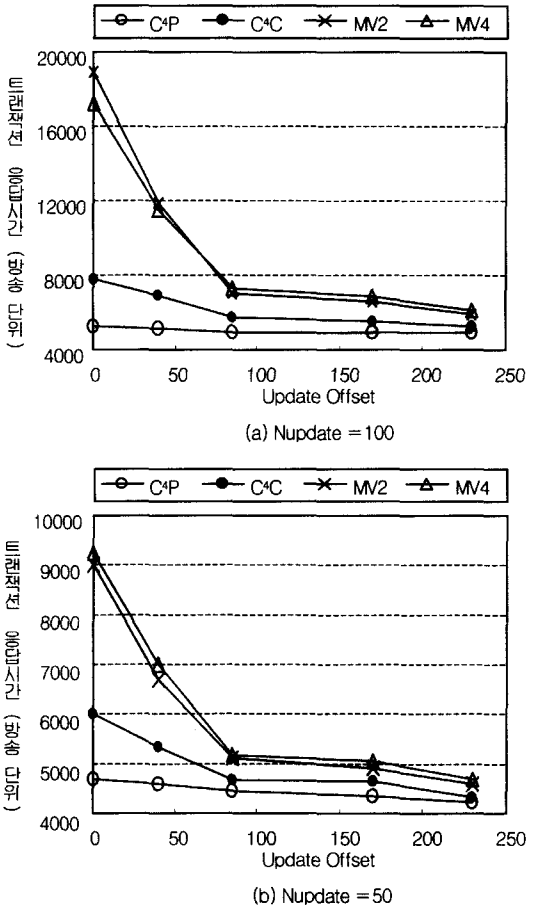


그림 4. 트랜잭션 응답 시간

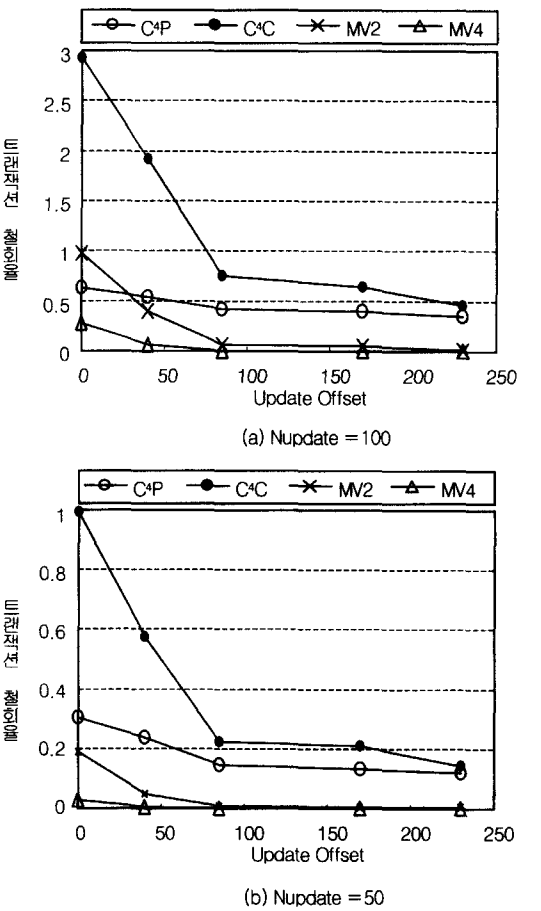


그림 5. 트랜잭션 철회율

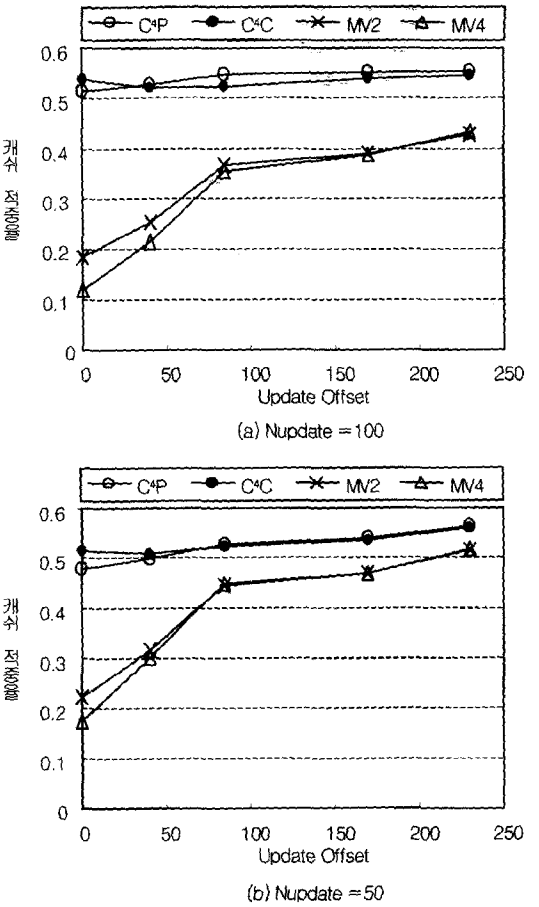


그림 6. 캐쉬 적중률

MV2와 MV4는 UpdateOffset이 작을 경우 성능이 매우 저하되는 것으로 나타났는데, 그 이유는 다음과 같다. 첫째, UpdateOffset이 작을 경우 트랜잭션이 무효화될 가능성이 높고, 무효화된 트랜잭션은 이전 버전의 데이타를 판독하여야 한다. 이전 버전의 데이타는 방송 주기가 가장 낮은 디스크에 저장되어 있으므로 이에 대한 대기 시간이 매우 길어질 수 있다. 둘째, 캐쉬에는 최신 데이타가 저장되어 있지만 트랜잭션은 이전 버전의 데이타들을 판독할 가능성이 높으므로, 그림 6에 나타나듯이 캐쉬 적중율이 낮아진다. 마지막으로, 방송 프로그램에 이전 버전의 데이타도 포함되므로 최신 버전의 데이타에 대한 평균 대기 시간이 길어진다. 따라서 보다 많은 버전을 방송할수록 평균 대기 시간은 더욱 증가하며, 그 결과 UpdateOffset이 크거나 Nupdate가 작을 경우 MV4는 MV2보다 성능이 떨어지는 것으로 나타났다.

V. 결 론

본 논문에서는 방송 디스크 환경에서 클라이언트 트랜잭션의 정확성을 보장하기 위한 동시성 제어 기법으로 C^4 기법을 제안하였다. 방송 디스크 환경에서 기존에 제안된 대부분의 동시성 제어 기법들은 추가적인 동시성 제어 정보들을 방송함으로써 트랜잭션의 철회율을 줄이는 것을 목적으로 하고 있다. 이와는 달리 C^4 기법은 동시성 제어를 위한 최소한의 정보만을 방송하며, 그 결과 발생할 수 있는 빈번한 트랜잭션 철회에 대해 철회된 트랜잭션의 실행 결과를 캐쉬에 저장함으로써 트랜잭션이 신속히 재실행되도록 한다. 즉, 서버로부터의 방송 대역폭을 최소한 사용하며 트랜잭션의 빠른 재실행을 가능하게 함으로써 트랜잭션이 최종적으로 완료되는 시점을 단축할 수 있다는 것이 C^4 기법의 기본 개념이다.

시뮬레이션을 이용하여 C^4 기법과 기존에 제안된 대표적인 동시성 제어 기법인 다중버전 기법의 성능을 비교하였다. 실험 결과 C^4 기법의 성능이 전반적으로 우수한 것으로 나타났으며, 클라이언트에서 판독하는 데이타들이 서버에서 갱신될 가능성이 클수록 성능 차이가 확대되었다. 뿐만 아니라, 트랜잭션이 판독하는 데이타의 현재성을 희생함으로써 C^4 기법의 성능이 현저하게 증가하였다. 특정 데이타가 빈번히 참조되는 비균일(non-uniform) 액세스 패턴이 일반적인 데이타베이스 응용 분야의 속성에 비추어 볼 때 C^4 기법의 이러한 특징은 매우 바람직하다고 할 수 있다.

참 고 문 헌

[1] S. Acharya, R. Alonso, M. Franklin, and S. Zdonik, "Broadcast Disks: Data Management for Asymmetric Communication Environment," *Proc. of ACM SIGMOD*, pp.199-210, 1995.

[2] S. Acharya, M. Franklin, and S. Zdonik, "Disseminating Updates on Broadcast Disks," *Proc. of 22nd VLDB Conf.*, pp.354-365, 1996.

[3] J. Jing, A. Heral, and A. Elmagarmid, "Client-Server Computing in Mobile Environments," *ACM Comp. Surveys*, 31(2), pp.117-157, 1999.

[4] D. Knuth, *The Art of Computer Programming*

(3) - Sorting and Searching, Addition Wesley 1998.

[5] K. Lam, M. Au, and E. Chan, "Broadcast of Consistent Data to Read-Only Transactions from Mobile Clients," *Proc. of 2nd IEEE Workshop on Mobile Comp. Syst. and Applications*, pp.80-88, 1999.

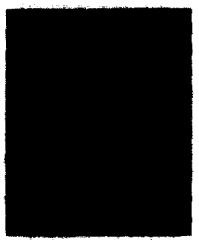
[6] E. Pitoura and P. Chrysanthis, "Exploting Versions for Handling Updates in Broadcast Disks," *Proc. of 25th VLDB Conf.*, pp.114-125, 1999.

[7] E. Pitoura and P. Chrysanthis, "Scalable Processing of Read-Only Transactions in Broadcast Push," *Proc. of 19th ICDCS*, pp.432-439, 1999.

[8] H. Schwetman, *CSIM Users Guide for use with CSIM Revision 16*, MCC, 1992.

[9] J. Shanmugasundaram, et al., "Efficient Concurrency Control for Broadcast Environments," *Proc. of ACM SIGMOD*, pp.85-96, 1999.

조 행 래(Haengrae Cho) 정회원



1988년 : 서울대학교 컴퓨터공학과(공학사)
1990년 : 한국과학기술원 전산학과(공학석사)
1995년 : 한국과학기술원 전산학과 (공학박사)

1995년~현재 : 영남대학교 전자정보공학부 부교수
<주관심 분야> Mobile Computing, 분산 데이터베이스, 고성능 트랜잭션 처리, 성능 평가