재배열 기반의 교착상태 없는 혼성 병행실행제어☆

A Hybrid Concurrency Control without Deadlock based on Reordering

조 성 호*
Sung-Ho Cho

요 약

클라이언트-서버 데이터베이스 시스템에서 많은 병행실행제어가 연구되었다. 그러나 일반적으로 알려진 기법들은 데이터베이스에서 데이터의 집중으로 인한 수많은 변화를 수용하기 어렵다. 본 논문에서는 혼성 병행실행제어를 제안한다. 제안하는 기법은 첫 번째 실행에서는 낙관적인 기법을 사용하며 두 번째 실행에서는 비관적인 기법을 사용한다. 제안하는 기법에서는 두 번째 단계에서 잠금을 미리 선언하기 때문에, 비관적인 기법에서 발생할 수 있는 교착상태는 발생하지 않는다. 또한, 낙관적 단계의 검사가 실패하더라도 단 한번만 재실행되는 것을 보장한다. 실험을 통하여 제안하는 기법과 분산 낙관적 병행실행제어를 비교하고, 제안하는 기법이 분산 낙관적 병행실행제어보다 우수하다는 것을 보인다.

Abstract

Many of concurrency control for client-server database systems have been proposed and studied in the literature. However, commonly known schemes do not manage to the case of dramatic changes in data contention because the data contention of database systems is changed for each cases. In this paper, we propose a hybrid concurrency control. The proposed scheme uses a optimistic scheme for the first-run transactions and a pessimistic scheme for the second-run transactions. By pre-claiming locks in the second phase, deadlocks which are possible in pessimistic based concurrency control are prevented in our approach. In addition, the scheme ensures at most one re-execution even if the validation in the optimistic phase fails. By a detailed simulation, this paper examines the behaviors of the Distributed Optimistic Concurrency Control and the proposed scheme. The simulation study shows our scheme outperforms Distributed Optimistic Concurrency Control in our experimental results.

☞ Keyword : Concurrency Control, Hybrid Concurrency Control, Deadlock Free Scheme, Reordering Scheme, 병행실행제어 기법, 재배열 기법, 혼성 병행실행제어, 비관적 병행실행제어, 교착상태

1. 서 론

병행실행제어 기법(concurrency control scheme)은 병렬로 수행되고, 데이터를 공유하며, 또한 서로 간에 간섭이 일어날 수 있는 프로세스들의 행동들을 조정하는 방법으로 트랜잭션 처리 시스템의 성능에 중대한 영향을 미친다[1-3]. 병행실행 제어는 크게 비관적 기법과 낙관적 기법으로 나눌 수 있으며, 클라이언트-서버 데이터베이스 환경을 위한 많은 종류의 병행실 제어가 제안되었다. 또한, 실시간 데이터베이스를 위한 병행실행제어도 제안되었다[4].

일반적으로, 낙관적인 병행실행제어는 매우 높은 데이터 집중(contention) 상태에서 많은 철회(abort)를 만들며, 반대로 비관적인 병행실행제어는 적은 데이터 집중 상태에서 실행되는 트랜잭션의 수를 제안하여 효율성이 떨어진다[5-7].

본 논문에서는 HAD(Hybrid certification protocol Assured Deadlock-free)이라 불리는 혼성(Hybrid) 병행실행제어를 제안한다. 제안하는 기법은 첫 번째 실행에서는 낙관적인 병행실행

* 정회원 : 한신대학교 정보과학통신학과 교수
zoch@hs.ac.kr
[2006/09/18 투고 - 2006/10/10 심사 - 2006/11/06 심사완료]
☆ 본 논문은 2006년 한신대학교 학술연구비 지원에 의하여 연구되었음

제어를 사용하고 두 번째 실행에서는 비관적인 병행실행제어를 사용한다. 두 번째 단계에서 잠금을 미리 선언하기 때문에, 비관적인 기법에서 발생할 수 있는 교착상태는 발생하지 않는다. 또한, 낙관적 단계의 검사가 실패하더라도 단 한번만 재실행되는 것을 보장한다.

제안하는 혼성기법은 재배열(reordered)기법을 사용한다. 이러한 특징으로 인하여 낙관적인 단계에서 완료(commit)되지 못하고 철회되는 트랜잭션은 재배열 기법에 의하여 트랜잭션의 순서를 변경하여 철회될 수 있는지 확인하게 된다. 이 때, 재배열되지 못한 트랜잭션을 철회된 후 다시 시작하게 되며, 두 번째 실행단계에서는 첫 번째 실행에서 필요했던 모든 데이터에 대하여 잠금을 사용하게 된다. 잠금을 획득할 시에도 재배열기법을 사용하여 잠금을 획득하게 됨으로 트랜잭션의 가용성을 높일 수 있다.

재배열 기법은 직열화 그래프(serialization graphs)[8]나 다중 버전(multiple versions)[9] 혹은 타임스탬프 히스토리(time-stamp histories)[10] 기법에서도 사용된다. 그러나 이러한 방식들은 많은 양의 추가 데이터를 필요로 한다. [11]의 재배열 기법은 적은양의 데이터를 사용하지만 낙관적 기법만을 가정하였다.

본 논문에서는 성능평가를 통하여 혼성 병행실행제어 기법 중 가장 성능이 좋은 것으로 알려진 분산 낙관적 병행실행제어(Distributed Optimistic Concurrency Control; DOC)[12]와 제안하는 기법을 비교한다. 비교를 통하여 제안하는 기법이 DOC보다 높은 성능을 나타낸다는 것을 보인다.

2. 교착상태 없는 혼성 병행실행제어

제안하는 기법에서는 두 종류의 트랜잭션이 존재한다. 첫 번째 트랜잭션(first-run transaction)은 비관적인 병행실행제어를 사용하고 두 번째 트랜잭션(second-run transaction)은 낙관적인 병행실행제어를 사용한다. 첫 번째 트랜잭션의 경우 낙관적 기법을 사용하기 때문에 실행도중에 충돌검사 없이 데이터를 사용하다가 최종 단계에서 완료를 요청하면 서버가 재배열 기법을 사용하여 완료 할 수 있는지를 검사한다. 완료를 할 수 없을 경우 해당 트랜잭션은 철회 된 후, 두 번째 트랜잭션이 된다. 두 번째 트랜잭션은 실행을 하기 전에 재배열 기법을 사용하여 모든 잠금을 획득하여야 하며, 모든 잠금이 획득하지 못했을 경우 기다리게 된다.

2.1 첫 번째 트랜잭션을 위한 재배열 기법

본 논문에서는 표기를 단순하게 하기 위하여 <표 1>과 같은 표기법을 사용한다. 제안하는 기법에서 서버는 각 데이터 D에 대하여 읽기 타임스탬프 $D.T^R$과 쓰기 타임스탬프 $D.T^W$를 유지한다. 두 타임스탬프는 D를 접근했던 트랜잭션 중, 가장 나중에 완료된 트랜잭션의 타임스탬프이다. 트랜잭션이 데이터 D_i를 읽기 원

〈표 1〉 본문에서 사용되는 표기법과 의미

분류	표기법과 의미
일반글자	*X,Y*:트랜잭션, *D*:데이터, *S*:데이터 집합, *T*:타임스탬프, *XL*:배타적인 잠금, *RL*:읽기 잠금, *WL*:잠금 요청 순서,
윗첨자	^{R}:읽기연산, ^{W}:쓰기연산, ^{L}:하한값, ^{U}:상한값, ^{C}:완료 값, ^{I}:무효화 값
아래첨자	$_X$, $_Y$:트랜잭션 구분자, $_i$, $_j$, $_k$:데이터 구분자

할 때, 만약 D_i가 클라이언트 캐시에 없다면, 서버는 데이터와 함께 현재의 $D_i.T^W$를 보내준다. 클라이언트들은 자신의 캐시에 데이터와 타임스탬프를 관리한다. 그러므로 트랜잭션은 데이터를 <데이터, 버전>의 쌍으로 본다.

클라이언트는 자신의 트랜잭션 X를 위하여 하한(lower-bound) 타임스탬프 T_X^L, 상한(upper-bound) 타임스탬프 T_X^U, 읽기집합 S_X^R, 쓰기집합 S_X^W, 무효화 집합 S_X^I를 유지한다. S_X^R과 S_X^W는 검사(validation)를 위해 유지되고, T_X^L과 T_X^U는 재배열을 위해 유지되며, S_X^I는 필요 없는 연산을 줄이기 위해 유지된다. T_X^L과 T_X^U의 초기 값은 시스템 내의 가장 작은 값으로 초기화된다.

만약 한 트랜잭션이 재배열 돼야 한다면, 해당 트랜잭션은 자신이 읽은 데이터의 타임스탬프들 보다 작은 값으로는 재배열 될 수 없다. 그러므로 T^L은 자신의 읽은 데이터의 타임스탬프 중 가장 큰 값을 가지고 있어야 한다. 이를 위해서, 트랜잭션 X가 D_i를 읽을 때마다 T_X^L은 $D_i.T^w$와 비교된다. 만약 T_X^L이 $D_i.T^w$보다 작으면, T_X^L은 $D_i.T^w$로 설정된다. 그 후, X의 클라이언트는 D_i를 집합 S_X^R에 넣는다.

트랜잭션 X가 완료를 요청할 때, 만약 X가 읽기전용 트랜잭션이라면 서버와의 접속 없이 완료된다. 그렇지 않다면, X의 클라이언트는 S_X^R, S_X^W, T_X^L, T_X^U를 서버에게 보낸다. 서버가 이 메시지를 받으면, 유일한 완료 타임스탬프 T_X^C를 부여한다. 그 후, 서버는 S_X^W에 있는 갱신된 데이터의 복사본을 가진 모든 원격지 클라이언트에게 T_X^C, S_X^R, S_X^W를 포함한 무효화 메시지를 보낸다. 원격지 클라이언트가 이 무효화 메시지를 받으면, X에 의해 갱신된 데이터를 캐시에서 지워버리고, 응답메시지를 서버에게 보낸다. 모든 응답메시지를 받으면, 서버는 S_X^R에 있는 각각의 데이터 D_j에 대하여 $D_j.T^R$을 완료 타임스탬프 T_X^C로, S_X^W에 있는 각각의 데이터 D_k에 대하여 $D_k.T^W$를 T_X^C로 설정한다. 그 후, 서버는 X의 클라이언트에게 완료 메시지를 보낸다.

한 트랜잭션이 완료가 되면, 완료된 트랜잭션이 쓴 데이터 값이 변했으므로, 이러한 값을 읽은 트랜잭션들은 백쉬프팅(back-shifting)된다. 한 트랜잭션이 백쉬프팅되기 위한 타임스탬프는 최소한 먼저 완료되고 자신이 읽은 값을 변경한 트랜잭션들의 값들보다는 작아야만 한다. 그러므로 T^U는 먼저 완료되고 자신이 읽은 값을 변경한 트랜잭션의 타임스탬프들 중에서 가장 작은 값을 유지해야만 한다. 이를 위해서 한 원격지 클라이언트가 X에 의해 발생된 무효화 메시지를 받았을 때, 트랜잭션 Y가 무효화된 데이터를 접근했다면, 클라이언트는 표2와 같은 알고리즘에 의해 T를 갱신한다.

〈표 2〉 무효와 메시지를 받았을 경우

```
If (S_X^W ∩ S_Y^R ≠ ∅) {
  If (S_X^R ∩ S_Y^W ≠ ∅) Abort Y;
  If (T_Y^U = 초기 값) T_Y^U = T_X^C;
  Else If (T_Y^U > T_X^C) T_Y^U = T_X^C;
  If (T_Y^L ≠ 초기 값 and T_Y^L ≥ T_Y^U) Abort Y; }
```

제안하는 기법에서 백쉬프팅을 위한 타임스탬프는 T^L과 T^U사이에 존재하게 된다. 그러므로 T_Y^L 혹은 T_Y^U가 변경될 때마다, 만약 T_Y^L이 T_Y^U보다 크거나 같다면, 서버는 재배열 할 수 있는 타임스탬프를 찾을 수 없으므로 Y는 철회된다. 또한, Y가 S_Y^I에 있는 데이터를 쓰기 연산을 하려고 한다면, 쓰기-쓰기 충돌을 방지하기 위해, Y는 철회된다.

무효화 된 데이터를 접근했던 Y가 완료를 요청하면, 클라이언트는 S_Y^R, S_Y^W, T_Y^L, T_Y^U를 서버에게 보낸다. 서버가 이 메시지를 받으면 T_Y^U가 초기 값이 아니므로 다른 트랜잭션에 의해 갱신된 값을 읽었다는 것을 알 수 있다. 이러한 경우에 있어서, 유일한 완료 타임스탬프를 부여하는 것 대신, 서버는 재배열을 위해서

완료 타임스탬프 T_Y^C를 T_Y^U-δ로 설정한다(δ는 극소 값(infinitesimal quantity)이다). 본 논문에서는 δ을 어떤 특정한 값이 아닌 극소 값으로만 명시한다. 특정한 값이 아닌 극소 값을 사용하게 되면 완료 된 트랜잭션들 사이에 재배열 될 수 있는 충분한 공간을 가지게 되는 장점이 생긴다.

재배열을 시키려는 트랜잭션 Y가 쓴 데이터를 다른 트랜잭션이 읽었다면 간접 충돌에 의하여 캐시 데이터의 일관성이 깨지는 현상이 발생 할 수 있다. 이를 방지하기 위하여 간접 충돌검사를 해야만 한다. 간접 충돌검사는 다음과 같다. T_Y^C를 T_Y^U-δ로 설정한 후, S_Y^W에 속한 모든 D_i에 대하여 서버는 재배열 타임스탬프 T_Y^C가 $D_i.T$보다 항상 큰가를 검사한다. 만약 T_Y^C가 이를 만족하지 못한다면 트랜잭션 Y는 철회된다. 그렇지 않다면, 트랜잭션 Y는 T_Y^C로 완료된다.

완료단계에서, 서버는 Y가 갱신한 데이터의 복사본을 가진 모든 클라이언트에게 T_Y^C, S_Y^R, S_Y^W를 보낸다. 모든 응답 메시지를 받으면, 서버는 S_Y^R에 있는 D_j에 대하여 $D_j.T^R$이 T_Y^C보다 작은 경우에 만 $D_j.T$을 T_Y^C로 설정한다. 또한, 서버는 S_Y^W에 있는 D_k에 대하여 타임스탬프 $D_k.T^W$가 T_Y^C보다 작은 경우에만 $D_k.T^W$을 T_Y^C로 설정한다.

2.2 두 번째 트랜잭션을 위한 재배열 기법

두 번째 트랜잭션을 위한 잠금을 획득하는 방법은 다음과 같다. 제안하는 기법에서는 다른 비관적인 방식과 마찬가지로 잠금 테이블을 사용한다. 잠금 테이블에는 다음과 같은 정보가 저장된다. XL은 배타적인 잠금(exclusive lock)을 나타내며 완료 시점에서 변경이 된 데이터를 나타내게 된다. RL은 읽기는 했으나 변경이 안 된 데이터들을 나타낸다. 모든 호환 불가능한 잠금을 요구한 트랜잭션들은 WL에 요구한 순서대로 저장된다.

만약 두 번째 트랜잭션인 Y가 요구한 모든 잠금이 획득되게 되면, 서버는 Y를 일반적인 완료 트랜잭션으로 간주하게 된다. 즉, 서버는 Y에게 유일한 완료 타임스탬프를 부여하게 되고 트랜잭션 Y의 쓰기 집합 S_Y^W의 모든 복사본을 가진 클라이언트에게 T_Y^C, S_Y^R, S_Y^W를 보낸다. 이러한 데이터를 받은 클라이언트들은 자신의 T^U를 2.1절에서 설명한 방식과 같이 업데이트를 한다. 업데이트 후 각 클라이언트들은 확인메시지를 서버에게 보낸다.

모든 확인 메시지를 받으면, 서버는 집합 S_Y^R에 있는 데이터 D_i에 대하여 $D_i.T$를 T_Y^C로 변경한다. 이 이후 트랜잭션 Y는 새로운 데이터와 함께 두 번째 실행을 시작한다. Y가 완료를 요청하면, 서버는 S_Y^W에 있는 데이터를 데이터베이스에 반영하고 잠금을 해제한다.

만약 Y가 데이터 D_i의 잠금 XL_i을 요구했을 때 이미 다른 트랜잭션이 해당 데이터의 잠금을 걸고 있다면 Y는 잠금이 풀릴 때 까지 기다려야만 한다. 그러나 제안하는 기법에서는, Y가 RL_j를 요청했을 때 이미 다른 트랜잭션이 XL_j를 가지고 있더라도 지연 시키지 않고 재배열 기법을 사용하여 잠금을 줄 수 있는지 검사한다. 검사방식은 2.1절에서 설명한 방법과 유사하다.

〈표 3〉 재배열 기반의 충돌검사

집합 S_Y^W에 있는 각 데이터 D_j에 대하여 완료 타임스탬프 T_Y^C는 언제나 $D_j.T$보다 크다.

우선, 트랜잭션 Y가 재배열 할 수 있는지 확인하기 위하여 다음과 같은 조건을 만족하는지 확인한다. "트랜잭션 Y가 쓸려고 하는 데이터에 대하여 트랜잭션 X가 RL을 가지고 있지 않으며, 또한 해당 RL을 가진 어떤 트랜잭션도 트랜잭션 X가 실행된 이후에 재실행되지 않았

다." 이러한 조건을 만족한다면 제안하는 기법에서는 충돌검사를 <표 3>과 같은 기준으로 실행한다. 이러한 조건을 만족한다면 트랜잭션 *Y*는 완료 타임스탬프를 δ만큼 백쉬프트시켜 재시작하게 된다.

3. 성능평가

이 장에서는 정량적 성능평가를 통하여 제안한 방법(HAD)과 분산 낙관적 병행실행제어(DOC)를 비교한다. 정량적 평가는 큐잉(queuing)모델을 기반으로 한다. 평가모델에는 매개변수 *NO_Client*가 클라이언트의 개수를 나타내며 각 클라이언트들은 *No_Tr*에 정해진 수만큼의 트랜잭션을 실행한다. 이번 평가에서는 1000개의 트랜잭션을 사용하였다. 만약 트랜잭션이 철회되면 매개변수 *Restart_Delay* 만큼 기다린 후, 다시 시작하게 된다. 재시작하는 트랜잭션은 이전과 같은 데이터를 접근하게 되면, 모든 트랜잭션은 결국에 완료된다. 매개 변수 *Tr_Size*는 트랜잭션들의 평균 연산수를 의미한다. 한 트랜잭션이 접근하는 데이터들은 중복됨이 없이 임의(random)로 선택된다. 읽혀진 데이터는 *Write_Prob*를 기준으로 하여 다시 쓰기 연산을 수행하게 된다.

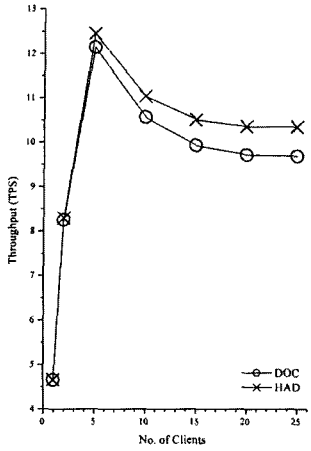
〈표 9〉 매개 변수 및 그 값들

매개변수	값
Page_Size	4K byte
Data_Size	1250
No_Client	1-25
No_Tr	1000
Tr_Size	20
Write_Prob	20%
Ex_IT	200
Ex_IO	200
Restart_Delay	100
Time_Read	30
Time_Write	30
Time_Netdelay	100

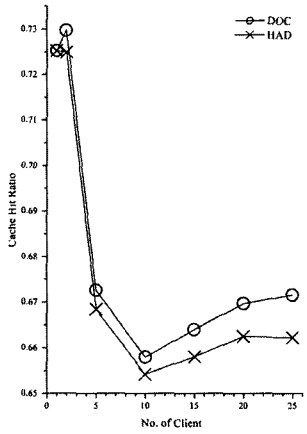
전체 데이터의 크기는 *Data_Size*이다. 시스템 내에서 트랜잭션이 도착하는 시간 사이의 간격은 *Ex_IT*를 기본으로 하는 지수분포를 따른다. 한 트랜잭션에 있어서 연산 사이의 간격도 *Ex_IO*를 기본으로 하는 지수분포를 따른다. 한 트랜잭션이 데이터를 읽거나 쓸 때에 *Time_Read*와 *Time-Write* 시간만큼 소비한다고 가정한다. 이번 실험에서 특정한 네트워크를 구현하지 않았으며 네트워크에서는 *Time_Net delay*만큼의 시간을 소비한다고 가정한다. <표 4>에 매개 변수들과 설정 값을 요약해 놓았다.

<그림 1>은 두 기법의 처리량을 나타낸다. 클라이언트 수가 4보다 작은 경우에는 데이터의 집중도가 낮아 둘 다 거의 비슷한 성능을 나타낸다. 두 기법 모두 낙관적인 기법을 사용하여 처음 트랜잭션을 실행시키기 때문에 클라이언트 수가 1에서부터 5까지는 성능이 급격히 증가하는 현상을 보인다. 그러나 클라이언트 수가 5를 넘어가면서 충돌이 발생하게 되고 철회되는 트랜잭션의 개수가 증가하면서 처리량이 감소하게 된다. 감소되는 처리량을 보면 DOC가 제안하는 HAD보다 큰다. 그 이유는 두 기법이 유사한 접근방식을 사용하지만, HAD가 재배열기법을 사용하여 트랜잭션의 가용성을 높였기 때문이다. HAD는 재배열 기법을 이용하여 잠금이 공유되는 정도가 DOC 보다 높다. 잠금이 공유되는 정도가 높아지게 되면 캐시에 있는 데이터의 일관성이 높게 유지되게 되고 이는 곧 높은 캐시 적중률을 나타내게 된다.

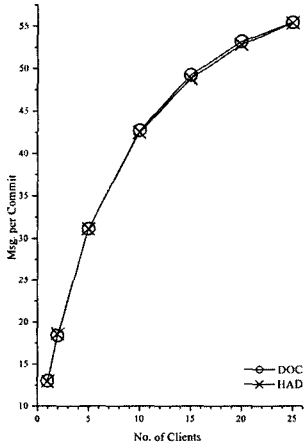
<그림 2>는 두 기법의 캐시 적중률을 나타낸다. 이미 설명한바와 같이 제안하는 HAD 기법이 재배열 기반의 잠금 공유기법을 사용하게 됨에 따라 동시에 시작되는 두 번째 실행 트랜잭션의 개수가 증가하게 되고, 이러한 현상으로 인하여 캐시의 일관성이 높게 유지 되게 됨으로 DOC보다 높은 캐시 적중률을 보이게 된다. 또한, 두 기법 모두 클라이언트 수가 1개에서 10개까지 캐시 적중률이 감소하다가 10개



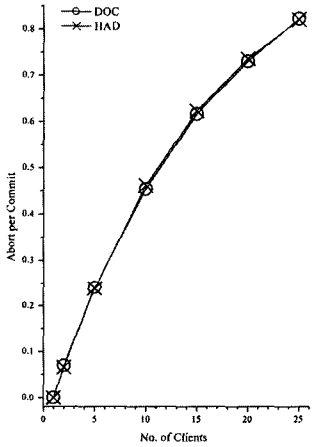
〈그림 1〉 처리량



〈그림 2〉 캐시 적중률



〈그림 3〉 메시지 수



〈그림 4〉 철회율

이후에 안정화되는 모습을 보였다. 그 이유는 클라이언트 수가 10개 이전까지는 첫 번째 실행에서 철회되는 트랜잭션들이 증가하여 해당 트랜잭션이 철회되면서 캐시의 데이터를 지워버리지만, 10개 이후에는 모든 트랜잭션이 비관적인 기법을 사용하게 되어 캐시의 데이터가 안정되는 현상을 보였다.

<그림 3>과 <그림 4>는 두 기법의 메시지 수와 철회율을 보인다. 두 그림이 나타내듯이 DOC와 HAD의 메시지 수와 철회율은 거의 유사하게 나타났다. 이는 제안하는 HAD가 DOC에 비하여 오버헤드가 거의 없음을 나타낸다. 즉, HAD는 거의 유사한 메시지 오버헤드를 사용하여 더 높은 처리량을 나타내고 있다.

4. 결 론

지금까지 제안된 병행실행제어는 크게 비관적인 방법이나 낙관적인 방법을 사용하고 있다. 이러한 기법의 차이가 분명하여 비관적인 기법의 경우 데이터의 집중이 적은 경우 동시에 실행되는 트랜잭션의 수를 제한하고, 낙관

적인 기법의 경우 데이터의 집중이 높은 경우 철회율이 증가하는 단점을 가지고 있다. 이러한 문제를 해결하기 위하여 본 논문에서는 혼성 병행실행제어 기법을 제안하였다. 제안하는 기법은 잠금을 미리 선언하여 교착상태를 방지하고, 낮은 철회율과 높은 잠금 사용율을 위하여 재배열 기법을 사용하였다. 본 논문에서는 성능평가를 통하여 제안하는 기법이 분산 낙관적 병행실행제어와 거의 비슷한 오버헤드를 가지면서도 더 높은 처리량을 가진다는 것을 보였다. 이러한 높은 처리량은 재배열 기법을 사용하게 됨에 따라 동시에 실행되는 트랜잭션의 수가 증가하고 동시에 캐시 적중률이 증가하기 때문이다.

참 고 문 헌

[1] E. Pitoura and P. K. Chrysanthis, "Multiversion Data Broadcast," *IEEE Transacions on Computers*, Vol. 51, No. 10, pp. 1224-1230, 2002.

[2] P. S. Yu and D. M. Dias, "Impact of Large Memeory on the Performance of Optimistic Concurrency Control Schemes," *Proc. Int. Conf. Databases, Parallel Architectures, and Their Appl.*, pp. 86-90, 1990.

[3] A. Adya, R. Gruber, B. Liskov and U. Maheshwari, "Efficient Optimistic Concurrency Control Using Loosely Synchronized Clock", *ACM SIGMOD*, 1995.

[4] 이석재. 박새미, 강태호, 유재수, "실시간 데이터베이스 시스템을 위한 효율적인 병행실행제어 알고리즘 설계," *한국인터넷정보학회 논문지*, 5권 1호, pp. 67-83, 2004.

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

[6] V. Gottemukkala, E. Omiecinski and U. Ramachandran, "Relaxed Consistency for a Client-Server Database," *Proc. of International Conference on Data Engineering*, 1996.

[7] M. J. Franklin, M. J. Carey, and M. Livny, "Local disk caching in client-server database systems," *Proc. of the Conf. on Very Large Data Bases*, pp. 543-554, 1993.

[8] Daniel Barbara, "Mobile Computing and Database - a Survey," *IEEE Transactions on Knowledge and Data Engineering*, Vol. 11, No. 1, pp. 108-117, 1999.

[9] Voruganti, "An Adaptive Hybrid Server Architecture for Client-Cashing ODBMSs," *Ph.D. thesis*, Univ. of Alberta, 2001.

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

[11] 조성호, 강민구, "브로드케스트 무효화 기법을 이용한 인증 프로토콜," *한국인터넷정보학회 논문지*, 3권 6호, pp. 79-89, 2002.

[12] A Thomasian, "Distributed Optimistic Concurrency Control Methods for High-performance Transaction Processing," *IEEE Transactions on Knowledge and Data Engineering*, vol. 10, pp. 173-189, 1998.

◐ 저 자 소 개 ◑

조 성 호 (Sung-Ho Cho)
1994년 한국외국어대학교 컴퓨터공학과 졸업(학사)
1997년 고려대학교 대학원 컴퓨터학과 졸업(석사)
2000년 고려대학교 대학원 컴퓨터학과 졸업(박사)
2002~현재 한신대학교 정보통신학과 교수
관심분야 : 분산시스템, 가상교육, 모바일컴퓨팅, 검사도구 etc.
E-mail : zoch@hs.ac.kr