재배열 기반의 낙관적 캐쉬 일관성 유지 기법의 모델링과 분석

(Modeling and Analysis of a Reordering-based Optimistic Cache Consistency Protocol)

조 성 호 † 황 정 현 ††
(Sung Ho Cho) (Jeong-hyon Hwang)

요 약 낙관적 2단계 잠금(O2PL)은 클라이언트 캐쉬를 적절히 사용하고 상대적으로 적은 네트워크 요구량으로 인하여 다른 기법보다 좋거나 같은 성능을 나타낸다. 그러나, 모든 잠금을 얻을 때까지 해당 트랜잭션이 완료될 수 없기 때문에 O2PL은 필요없는 기다림을 만든다. 또한, 낙관적 병행수행 제어(OCC) 기법은 필요없는 철회를 만든다. 본 논문은 이러한 단점들을 완화시킬 수 있는 효율적인 낙관적 캐쉬 일관성 유지 기법을 제안한다. 제안하는 기법은 트랜잭션의 완료나 철회를 기다림 없이 결정하며 철회율을 최소화하기 위하여 트랜잭션 재배열을 사용한다. 제안하는 기법은 재배열 기법을 사용함에도 불구하고 각 데이타 당 하나의 버전만을 필요로 한다. 마지막으로, 본 논문은 정량적 평가결과를 통하여 제안하는 기법이 O2PL과 OCC보다 좋은 성능을 나타낸다는 것을 보인다.

Abstract Optimistic Two-Phase Locking(O2PL) performs as well as or better than the other approaches because it exploits client caching well and also has relatively lower network bandwidth requirements. However, O2PL leads to unnecessary waits, because, it can not be commit a transaction until the transaction obtains all requested locks. In addition, Optimistic Concurrency Control(OCC) tends to make needless aborts. This paper suggests an efficient optimistic cache consistency protocol that overcomes such shortcomings. Our scheme decides whether to commit or abort a transaction without wait and it adopts transaction re-ordering in order to minimize the abort rate. Our scheme needs only one version for each data item in spite of the re-ordering mechanism used. Finally, this paper presents a simulation-based analysis that shows superiority in performance of our scheme to O2PL and OCC.

1. 서 론

최근 데이타베이스 시스템은 지역적으로 떨어진 사용자들에게 효율적인 데이타 공유를 지원하기 위하여 클라이언트/서버(client/server) 모델을 사용하고 있다. 이 모델의 단점은 클라이언트들에 의해 요청된 많은 양의 데이타로 인하여 네트워크나 서버에 병목현상이 발생하는 것이다. 이러한 문제를 완화시키기 위하여, 각 클라이언트들은 나중에 다시 사용하기 위한 일정양의 데이타를 자신의 캐쉬(cache)에 유지하고 있다[1-7,10-14]. 캐쉬를 사용하게 되면, 트랜잭션의 의미를 이용하여 클라이언트 캐쉬에 있는 데이타들의 일관성을 유지시켜주는 캐쉬 일관성 유지 기법을 필요로 한다[1-5].

현재까지 제안 된 캐쉬 일관성 유지 알고리즘은 크게 비관적인(pessimistic) 방법과 낙관적인(optimistic) 방법으로 나눌 수 있다[2,6,7]. 낙관적인 방법중의 하나인 낙관적 2단계 잠금(Optimistic Two-Phase Locking: O2PL)은 비관적인 방법에 비하여 적은 통신 오버헤드(overhead)를 가지므로 높은 메시지 비용이나 적은 충돌(conflict) 상황에서는 비관적인 방법보다 우수한 성능을 보인다[7-9]. 그러나, 한 트랜잭션이 완료(commit)를 요청하면 모든 잠금(lock)이 얻어질 때까지 해당 트랜잭션은 완료될 수 없기 때문에, O2PL은 필요없는 기

† 정 회 원 : 천안대학교 정보통신학부 교수
zoch@chonan.ac.kr
†† 비 회 원 : 미국 Brown Univ., Dept. of Computer Science
jhwang@disys.korea.ac.kr
논문접수 : 1999년 7월 14일
심사완료 : 2001년 8월 27일

다림을 만든다. 또 다른 낙관적인 방법인 낙관적 병행수행 제어(Optimistic Concurrency Control: OCC)기법의 가장 큰 단점은 철회(abort)에 민감하다는 것이다[16]. 본 논문에서는 이러한 종류의 필요없는 기다림이나 철회를 줄일 수 있는, 선점형 캐쉬 일관성 규약(Preemptive Cache consistency Protocol: PCP)이라 불리는, 효율적인 낙관적 캐쉬 일관성 유지 기법을 제안한다.

캐쉬 일관성 유지 기법은 또 다른 분류방법에 의해 회피기반(avoidance-based)과 검출기반(detection-based)으로 나눌 수 있다[2]. [2]에 있는 분류법에 의하면 PCP는 회피 기반 알고리즘 중의 하나이며, O2PL과 마찬가지로 쓰기 연산의 결과를 데이타베이스에 반영하는 시점을 트랜잭션의 종료시점까지 미룬다. PCP와 O2PL의 차이점은 O2PL에서는 한 트랜잭션이 완료를 요청했을 때 모든 잠금을 얻을 때까지 기다려야 하지만, PCP는 잠금을 사용하지 않기 때문에 완료를 요청 한 트랜잭션은 기다림 없이 완료되거나 철회되는 선점형(preemptive) 방법을 사용한다는 것이다. 선점형 방법 중의 하나인 Notify Locks[10]은 진행중인 트랜잭션 중 충돌이 검출된 모든 트랜잭션을 철회시키지만, PCP는 재배열 기법을 사용하기 때문에 진행중인 트랜잭션 중 쓰기-쓰기 충돌(write-write conflict)을 일으키는 트랜잭션만 철회시킨다. OCC와 비교해 보면 PCP는 무효화 된 데이타를 읽은 트랜잭션들을 재배열시킴으로서 철회율을 줄인다.

본 논문은 아래와 같이 구성되어 있다. 2장에서는 캐쉬 일관성 유지를 위한 새로운 선점형 알고리즘을 제안한다. 3장에서 우리의 정량적 평가 모델을 설명하고, 정량적 평가 결과에 대하여 다룬다. 그리고, 4장에서 결론을 맺는다.

2. 재배열 기반의 선점형 캐쉬 프로토콜

제안하는 기법에서 한 트랜잭션이 완료를 요청하면 기다림 없이 완료시키고, 완료된 트랜잭션이 갱신한 데이타를 접근했던 트랜잭션들은 재배열 기법에 의하여 완료되어진다. 제안하는 기법은 무효화(invalidation) 기법이나 전파(propagation) 기법 둘 다 적용 가능하다. 프로토콜을 단순화시키기 위하여, 본 논문에서는 무효화 기법을 기반으로 설명한다. 또한, 실제적으로 데이타는 파일(file), 객체(object) 혹은 페이지(page)일 수 있지만, 본 논문에서는 데이타의 입자성(granularity) 문제는 다루지 않고 단순히 데이타로만 표기한다.

2.1 재배열 기법

제안하는 기법과 다른 낙관적 기법과의 차이점을 설명하기 위하여 다음 예제를 살펴보자.

$R_X(D_1)$, $R_Y(D_1)$, $R_Y(D_2)$, $W_X(D_1)$, RC_X, $W_Y(D_2)$, RC_Y (예제 1)

여기서 R과 W는 각각 읽기 연산과 쓰기 연산을, RC는 완료 요청(request commit)을, 아래첨자는 해당 트랜잭션을 나타낸다. O2PL에서는 트랜잭션 X가 완료를 요청하면, 트랜잭션 Y가 D_1에 대하여 읽기 잠금을 소유하고 있음으로, 트랜잭션 Y가 완료 될 때까지 기다려야만 한다. OCC에서는 트랜잭션 X는 기다림 없이 완료되지만, X에 의해 무효화된 데이타 D_1을 읽었으므로, 트랜잭션 Y는 철회된다. 그러나, 트랜잭션 Y가 먼저 완료를 요청했더라면, O2PL과 OCC는 트랜잭션 X와 Y 둘 다 기다림 없이 완료시킬 수 있다. 이와 같이, 트랜잭션의 의미를 이용하여 트랜잭션 X를 먼저 완료시킨 후, 트랜잭션 Y를 완료시킬 때 직렬화 로그(serial log)에 Y를 먼저 완료시킨 것으로 반영하면, X와 Y를 기다림 없이 완료시킬 수 있다. 이렇게 실제의 완료순서를 변경시키는 방법을 재배열(re-ordering)이라 부른다.

제안하는 기법은 재배열 방식을 기반으로 하여, 한 트랜잭션이 완료를 요청하면, 해당 트랜잭션을 기다림 없이 종료시킨다. PCP는 잠금을 사용하지 않음으로 O2PL과 같이 전역 교착상태 검증 알고리즘을 필요로 하지 않을 뿐 아니라, 필요없는 기다림을 발생시키지 않는다. 이러한 필요없는 기다림은 OCC에서도 발생하지 않지만, OCC는 필요없는 철회를 유발시킨다. PCP는 재배열 방식을 사용하기 때문에 이러한 필요없는 철회를 줄인다.

2.2 재배열 기반의 캐쉬 일관성 유지 기법

본 논문에서는 표기를 단순하게 하기 위하여 타임스탬프(time-stamp)를 T로 집합(set)을 S로 표시하기로 한다. 또한, 아래첨자는 분류자(identifier)를 윗 첨자는 타임 스탬프와 집합의 종류(type)를 나타낸다. 제안하는 기법에서 서버는 각 데이타 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 데이타 D_i를 읽기 원할 때, 만약 D_i가 클라이언트 캐쉬에 없다면, 서버는 데이타와 함께 현재의 $D_i.T^w$를 보내준다. 클라이언트들은 자신의 캐쉬에 데이타와 타임스탬프를 관리한다. 그러므로, 트랜잭션은 데이타를 <데이타, 버전(version)>의 쌍으로 본다.

클라이언트는 자신의 트랜잭션 X를 위하여 하한(lower-bound) 타임스탬프 T_X^L, 상한(upper-bound) 타임스탬프 T_X^U, 읽기 집합 S_X^R, 쓰기 집합 S_X^W, 무효화 집합 S_X^I를 유지한다. 집합 S_X^R과 S_X^W는 검사

(validation)를 위해 유지되고, 타임스탬프 T_X^L과 T_X^U는 재배열을 위해 유지되며, 집합 S_X^I는 필요없는 연산을 줄이기 위해 유지된다. 타임스탬프 T_X^L과 T_X^U의 초기 값은 시스템 내의 가장 작은 값으로 초기화된다. 각 정보의 역할에 대해서는 다음에 설명된다.

만약 한 트랜잭션이 재배열 되야 한다면, 해당 트랜잭션은 자신이 읽은 데이타의 타임 스탬프들 보다 작은 값으로는 재배열 될 수 없다. 그러므로 하한 타임 스탬프 T^L은 자신의 읽은 데이타의 타임 스탬프 중 가장 큰 값을 가지고 있어야 한다. 이를 위해서, 트랜잭션 X가 데이타 D_i를 읽을 때마다 타임스탬프 T_X^L은 현재 타임스탬프 $D_i.T^w$와 비교된다. 만약 T_X^L이 $D_i.T^w$보다 작으면, T_X^L은 $D_i.T^w$로 설정된다. 그 후, 트랜잭션 X의 클라이언트는 D_i를 집합 S_X^R에 넣는다. (알고리듬 1 참조)

트랜잭션 X가 완료를 요청할 때, 만약 X가 읽기 전용(read-only) 트랜잭션이라면 서버와의 접속 없이 완료된다. 그렇지 않다면, 트랜잭션 X의 클라이언트는 집합 S_X^R, S_X^W, 타임스탬프 T_X^L, T_X^U를 서버에게 보낸다. 서버가 이 메시지를 받으면, 유일한 완료 타임스탬프 T_X^C를 부여한다. 그 후, 서버는 집합 S_X^W에 있는 갱신된 데이타의 복사본을 가진 모든 원격지 클라이언트(remote client)에게 T_X^C, S_X^R, S_X^W를 포함한 무효화 메시지를 보낸다. 원격지 클라이언트가 이 무효화 메시지를 받으면, 트랜잭션 X에 의해 갱신된 데이타를 캐쉬에서 지워버리고, 응답메시지를 서버에게 보낸다. 모든 응답메시지를 받으면, 서버는 집합 S_X^R에 있는 각각의 데이타 D_j에 대하여 타임스탬프 $D_j.T^r$을 완료 타임스탬프 T_X^C로, 집합 S_X^W에 있는 각각의 데이타 D_k에 대하여 타임스탬프 $D_k.T^w$를 완료 타임스탬프 T_X^C로 설정한다. 그 후, 서버는 트랜잭션 X의 클라이언트에게 완료 메시지를 보낸다.

한 트랜잭션이 완료가 되면, 완료된 트랜잭션이 쓴 데이타 값이 변했으므로, 이러한 값을 읽은 트랜잭션들은 재배열되어야만 한다. 한 트랜잭션이 재배열되기 위한 타임 스탬프는 최소한 먼저 완료되고 자신이 읽은 값을 변경한 트랜잭션들의 값들보다는 작아야만 한다. 그러므로, T^U는 먼저 완료되고 자신이 읽은 값을 변경한 트랜잭션의 타임 스탬프들 중에서 가장 작은 값을 유지해야만 한다. 이를 위해서 한 원격지 클라이언트가 트랜잭션 X에 의해 발생된 무효화 메시지를 받았을 때, 트랜잭션 Y가 무효화된 데이타를 접근했다면, 클라이언트는 다음과 같은 규칙에 의해 타임스탬프 T_Y^U를 갱신한다.

◎ 집합 S_X^R에 있는 각 데이타 D_i에 대하여, D_i가 집합 S_Y^W에 존재한다면 트랜잭션 Y는 재배열 될 수 없으므로 철회된다.

◎ 트랜잭션 Y가 철회되지 않았다면 타임스탬프 T_Y^U는 완료 타임스탬프 T_X^C와 비교되어진다. 만약, T_Y^U가 초기 값이면 T_Y^U는 T_X^C로 설정된다. 그 외의 경우에는 T_Y^U가 T_X^C보다 클 경우에만 T_Y^U는 T_X^C로 설정된다.

◎ 집합 S_Y^W와 S_Y^R에 속한 데이타 D_k는 필요없는 연산을 줄이기 위하여 집합 S_Y^I에 넣는다.

클라이언트가 무효화 메시지를 받았을 때 사용하는 알고리듬을 알고리듬 2에 표시하였다.제안하는 기법에서 재배열을 위한 타임스탬프는 T^L과 T^U사이에 존재하게 된다. 그러므로, 타임스탬프 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 트랜잭션 Y는 철회된다. 또한, 트랜잭션 Y가 집합 S_Y^I에 있는 데이타를 쓰기 연산을 하려고 한다면, 쓰기-쓰기 충돌을 방지하기 위해, 트랜잭션 Y는 철회된다(알고리듬 3 참조).

무효화 된 데이타를 접근했던 트랜잭션 Y가 완료를 요청하면, 클라이언트는 S_Y^R, S_Y^W, T_Y^L, T_Y^U를 서버에게 보낸다. 서버가 이 메시지를 받으면 타임스탬프 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 $T_Y^U - \delta$로 설정한다(δ는 극소 값(infinitesimal quantity)이다). 본 논문에서는 δ을 어떤 특정한 값이 아닌 극소 값으로만 명시한다. 재배열에 있어서 극소 값은 [15]에서 처음 사용하였는데, 특정한 값이 아닌 극소 값을 사용하게되면 완료 된 트랜잭션들 사이에 재배열 될 수 있는 충분한 공간을 가지게 되는 장점이 생긴다.

재배열을 시키려는 트랜잭션 Y가 쓴 데이타를 다른 트랜잭션이 읽었다면 간접 충돌에 의하여 캐쉬 데이타의 일관성이 깨지는 현상이 발생 할 수 있다. 이를 방지하기 위하여 간접 충돌 검사를 해야만 한다. 간접 충돌 검사는 다음과 같다. T_Y^C를 $T_Y^U - \delta$로 설정한 후, 집합 S_Y^W에 속한 모든 데이타 D_i에 대하여 서버는 재배열 타임스탬프 T_Y^C가 $D_i.T^r$보다 항상 큰가를 검사한다. 만약 재배열 타임스탬프 T_Y^C가 이를 만족하지 못한다면 트랜잭션 Y는 철회된다. 그렇지 않다면, 트랜잭션 Y는 타임스탬프 T_Y^C로 완료된다.

완료단계에서, 서버는 트랜잭션 Y가 갱신한 데이타의 복사본을 가진 모든 클라이언트에게 T_Y^C, S_Y^R, S_Y^W를 보낸다. 모든 응답 메시지를 받으면, 서버는 집합 S_Y^R에

있는 데이타 D_j에 대하여 타임스탬프 $D_i.T^r$이 타임스탬프 $T_Y{}^C$보다 작은 경우에 만 $D_i.T^r$을 $T_Y{}^C$로 설정한다. 또한, 서버는 집합 $S_Y{}^W$에 있는 데이타 D_k에 대하여 타임스탬프 $D_k.T^w$가 타임스탬프 $T_Y{}^C$보다 작은 경우에만 $D_k.T^w$을 $T_Y{}^C$로 설정한다. 트랜잭션이 완료를 요청했을 때 사용하는 알고리듬을 알고리듬 4에 표시하였다.

```
/* 한 트랜잭션(트랜잭션 Y)이 데이타 Di를 읽을 때 */
If (TY^L < Di.T^w) TY^L = Di.T^w;
If (TY^L ≠ 초기 값 and TY^L ≥ TY^U)
    Y를 철회시킴; /* 상한 값이 하한 값보다 작은 경우 */
Else SY^R = SY^R + {Di};
```

알고리듬 1 클라이언트가 데이타를 읽을 때

```
/* (트랜잭션 X로부터) 무효화 메시지가 도착 할 때 */
If (SX^W ∩ SY^R ≠ ∅) {
    If (SX^R ∩ SY^W ≠ ∅)
        Y를 철회시킴; /* 쓰기-쓰기 충돌이 발생한 경우 */
    If (TY^U = 초기 값) TY^U = TX^C;
    Else If (TY^U > TX^C) TY^U = TX^C;
    If (TY^L ≠ 초기 값 and TY^L ≥ TY^U)
        Y를 철회시킴; /* 상한 값이 하한 값보다 작은 경우 */
    SY^I = SY^I + (SY^R ∩ SX^W); }
    응답메시지를 서버에게 보냄;
```

알고리듬 2 클라이언트가 무효화 메시지를 받을 때

```
/* 한 트랜잭션(트랜잭션 Y)이 데이타 Dj에 쓰려 할 때 */
If (Dj ∈ SY^I)
    Y를 철회시킴; /* Y가 무효화된 데이타를 쓰려는 경우 */
Else SY^W = SY^W + {Di};
```

알고리듬 3 클라이언트가 데이타를 갱신하려 할 때

```
/* 한 트랜잭션(트랜잭션 Y)이 완료를 요청했을 때*/
If (TY^L ≠ 초기 값) {
    TY^C = TY^U - δ;
    For (each Di ∈ SY^W) If (Di.T^r > TY^C)
        Y를 철회시킴; } /* 간접 충돌 검사 */
Else 새로운 TY^C 값을 부여 함;
TY^C, SY^W, SY^R를 트랜잭션 Y가 갱신한 데이타를 가진 모든
클라이언트에게 보냄;
If (모든 응답 메시지를 받으면) {
    For (each Dj ∈ SY^W) If (Dj.T^w > TY^C) Dj.T^w = TY^C;
    For (each Dk ∈ SY^R) If (Dk.T^r > TY^C) Dk.T^r = TY^C; }
```

알고리듬 4 완료를 요청했을 때

2.3 예제를 통한 알고리즘 설명

예제 1의 실행 스케줄 $R_X(D_1)$, $R_Y(D_1)$, $R_Y(D_2)$, $W_X(D_1)$, RC_X, $W_Y(D_2)$, RC_Y로서 제안하는 기법을 설명한다. 시스템의 기본 값으로 각 데이타는 <읽기 타임스탬프, 쓰기 타임스탬프>를 가지며, 각 타임스탬프의 초기 값은 ts_1이다. 그리고, 타임스탬프 T^L과 T^U의 초기 값은 ts_0 ($< ts_1$)이며 트랜잭션 X는 클라이언트 1에서, 트랜잭션 Y는 클라이언트 2에 실행된다고 가정한다. 표 1은 초기 값들을 보여준다. 트랜잭션 X가 완료를 요청하면, 클라이언트 1은 $T_X{}^L$ (= ts_1), $T_X{}^U$ (= ts_0), $S_X{}^R$ (= $\{D_1\}$), $S_X{}^W$ (= $\{D_1\}$)을 서버에게 보낸다 (표 2 참조). 타임스탬프 $T_X{}^U$가 초기 값이므로, 서버는 완료 타임스탬프 $T_X{}^C$를 ts_5로 설정하고 $T_X{}^U$, $S_X{}^R$, $S_X{}^W$를 클라이언트 2에게 보낸다. 클라이언트 2가 메시지를 받으면, 트랜잭션 Y가 데이타 D_1을 읽기만 했으므로, 타임스탬프 $T_Y{}^U$를 ts_5로 설정하고 응답 메시지를 서버에게 보낸다. 서버가 응답 메시지를 받으면 타임스탬프 $D_i.T^R$과 $D_i.T^w$를 ts_5로 설정하고, 트랜잭션 X를 완료시킨다.

표 1 초기값

클라이언트 1	서버	클라이언트 2
$T_X{}^L = ts_0$	D_1 (ts_1, ts_1)	$T_Y{}^L = ts_0$
$T_X{}^U = ts_0$	D_2 (ts_1, ts_1)	$T_Y{}^U = ts_0$
	D_3 (ts_1 ts_1)	

표 2 X가 완료를 요청할 때

클라이언트 1	서버	클라이언트 2
$T_X{}^L = ts_1$	D_1 (ts_1, ts_1)	$T_Y{}^L = ts_1$
$T_X{}^U = ts_0$	D_2 (ts_1, ts_1)	$T_Y{}^U = ts_0$
	D_3 (ts_1 ts_1)	

표 3 Y가 완료를 요청할 때

클라이언트 1	서버	클라이언트 2
	D_1 (ts_5, ts_5)	$T_Y{}^L = ts_1$
	D_2 (ts_1, ts_1)	$T_Y{}^U = ts_5$
	D_3 (ts_1 ts_1)	

표 4 Z가 완료를 요청할 때

클라이언트 3	서버	클라이언트 2
$T_Z{}^L = ts_1$	D_1 (ts_5, ts_1)	
$T_Z{}^U = ts_4$	D_2 (ts_4, ts_4)	
	D_3 (ts_5 ts_5)	

트랜잭션 Y가 ts_6에 완료를 요청하면 타임스탬프 T_Y^U가 초기 값이 아니므로 재배열 타임스탬프 T_Y^C를 ts_4(= $T_Y^U - \delta$, $\delta = ts_1$)로 설정한다 (표 3 참조). 타임스탬프 $D_2.T^r$이 재배열 타임스탬프보다 작으므로 트랜잭션 Y는 완료된다. 쓰기 타임스탬프 $D_2.T^w$가 ts_4로 설정되지만, 재배열 타임스탬프 T_Y^C보다 크므로, 읽기 타임스탬프 $D_1.T^r$은 변하지 않는다.

트랜잭션 Z가 추가된 다음의 예 $R_X(D_1)$, $R_Z(D_1)$, $R_Z(D_3)$, $R_Y(D_1)$, $R_Y(D_2)$, $R_X(D_3)$, $W_X(D_3)$, RC_X, $W_Y(D_2)$, RC_Y, $W_Z(D_1)$, RC_Z를 살펴보자. 트랜잭션 Y가 완료되면, 트랜잭션 Z가 데이타 D_2를 읽었으므로, 타임스탬프 T_Z^U는 ts_4로 설정된다 (표 4 참조). 실제적으로 직렬화 그래프에서 트랜잭션 X, Y, Z가 순환(cycle)을 형성하기 때문에 트랜잭션 Z는 철회되어야 한다. 트랜잭션 Z가 ts_7에 완료를 요청하면, 재배열 타임스탬프 값 ts_3가 타임스탬프 $D_1.T^r$보다 작으므로, 트랜잭션 Z는 철회된다.

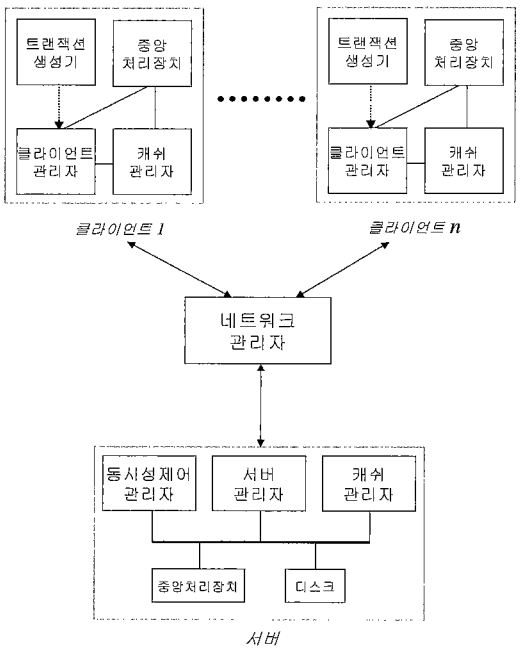

그림 1 정량적 평가 모델

3. 정량적 평가 및 결과 분석

3.1 정량적 평가 모델

이 절에서는 OCC와 O2PL을 제안하는 기법(PCP)과 비교한다. 그림 1은 우리의 정량적 평가 모델을 도식화한 것이고, 표 6은 정량적 평가에 사용된 매개변수 및 그 값을 나타낸다. 우리의 정량적 평가 모델은 디스크가 없는 워크스테이션과 서버가 간단한 네트워크로 연결되어 있는 구조를 가정하였다. *No_Client*는 시스템의 클라이언트 수를 나타낸다. 클라이언트 및 서버에 있는 *캐쉬 관리자*는 LRU 페이지 대체 알고리즘을 사용한다. 각 클라이언트는 *트랜잭션 생성기*로부터 받은 트랜잭션을 한번에 한 개씩 실행한다. 클라이언트와 비교해보면, 서버는 다음과 같은 차이점을 갖는다. 서버는 *중앙처리장치*(CPU)뿐만 아니라 *디스크*를 관리하며, 페이지의 복사본을 가진 클라이언트의 정보를 저장하며, O2PL을 위해서 잠금을 유지한다.

표 5 매개변수 및 값

매개변수	값
Page_Size	4K byte
DB_Size	1250
No_Client	1 to 25
No_Tr	1000
Tr_Size	20 page
Write_Prob	20%
Ex_Tr	0 sec.
Ex_Op	0 sec.
Client_CPU	15 MIPS
Server_CPU	30 MIPS
Client_Buf	25% of DB
Server_Buf	50% of DB
Ave_Disk	20 millisecond
Net_Bandwidth	8Mbps
Page_Inst	30K inst.
Fix_Msg_Inst	20K inst.
Add_Msg_Inst	10K inst. per 4K byte
Control_Msg	256 byte
Lock_Inst	0.3K inst.
Disk_Overhead	5K inst.
DL_Detect_Freq	1 sec.

만약, 트랜잭션이 철회되면 그 트랜잭션은 다시 실행되며, 첫 번째 실행 때와 같은 데이타에 접근하게된다. 그러므로, 모든 트랜잭션은 결국에는 종료되게 된다. 한 클라이언트내의 트랜잭션들의 수는 *No_Tr*이라 가정한다. 정확한 결과를 얻기 위해서 우리는 각 클라이언트들이 1000개의 트랜잭션들을 실행하게 하였다. 그러므로, 이번 정량적 평가는 최대 25000개의 트랜잭션들을 실험한 결과이다. 각 트랜잭션은 *Tr_Size*만큼의 읽기나 쓰기 연산을 수행하여 데이타에 접근하게된다. *Ex_Tr*은 트랜잭션들이 도착하는 시간을 나타내며, *Ex_Op*는 연산사이의 평균시간을 나타낸다. 데이타 집중도를 높이기 위해, 이 실험에서는 두 매개변수를 0으로 정하였다. 데

이타들은 다음에 설명하는 작업부하(workload)를 기본으로 하여 중복됨이 없이 불규칙하게 선택된다. 데이타베이스 내의 페이지 수는 *DB_Size*로 표시된다. *Page_Size*는 각 페이지의 크기를 나타낸다. 페이지를 접근하는데 드는 비용(*Page_Inst*)은 고정된 수의 연산으로 표시된다. 읽은 데이타를 다시 쓸 확률은 *Write_Prob*에 의해 결정된다.

*Client_Buf*와 *Server_Buf*는 각각 클라이언트와 서버의 버퍼 크기를 나타낸다. 이 실험에서는 각 클라이언트가 비교적 큰 크기의 버퍼 (전체 페이지의 25%)를 가졌다고 가정한다. CPU의 성능은 CPU MIPS에 의존하며 표 6에 명시된 연산을 수행하게 된다. 평가에 사용된 CPU는 두 개의 우선 순위(priority)를 사용하며 메시지와 디스크에 대한 요구를 처리하는데 사용된다. CPU와 디스크는 FIFO를 사용한다. 디스크를 접근하는데 걸리는 평균시간은 *Ave_Disk*에 명시되어 있다. *Disk_Overhead*는 디스크를 접근하기 위해 CPU에서 처리되는 연산수를 나타낸다. *Lock_Inst*는 잠금을 얻거나 풀기 위해 필요한 연산의 수를 나타내며 *DL_Detect_Freq*은 교착상태 검출 빈도를 나타낸다. 교착상태를 검출하기 위하여 서버는 그래프를 유지한다.

이번 실험에서는 단순한 네트워크 모델을 사용하였다. 네트워크는 *Net_Bandwidth*의 대역폭을 가진 단순한 FIFO 서버라고 가정하였다. 대역폭은 충돌로 인한 감소분을 고려하여 이더넷(Ethenet)의 대역폭으로서 8Mbps/sec으로 정했다. 메시지를 주고받기 위해 이용되는 CPU의 양은 각 메시지마다 고정된 연산의 수(*Fix_Msg_Inst*)에 각 byte마다 추가되는 메시지 수(*Add_Msg_Inst*)를 더하여서 계산하였다. *Control_Msg*는 페이지를 포함하지 않는 조정 메시지의 크기를 나타낸다.

표 6 클라이언트 n에 대한 작업부하 매개변수 및 설정

매개변수	HICON	HOTCOLD	UNIFORM
Hot_Bound	5% to 35% of DB	p to p+49, p=50(n-1)+1	-
Cold_Bound	Rest of DB	Rest of DB	All of DB
Hot_Acc_Prob	0.8, 0.6	0.8	-
Cold_Acc_Prob	0.2, 0.4	0.2	1.0

우리의 정량적 평가 모델은 클라이언트 작업부하를 변화시켜가며 실험하였다. 각 클라이언트의 접근 패턴(pattern)은 표 6에 나타난 매개변수에 의해 결정된다. 데이타베이스는 *HOT*과 *COLD*의 두 개의 영역으로 구분되어 진다. 표 7에 HOT 부분을 접근하는 확률을 명시하였고 나머지는 COLD 부분을 접근하게 된다. *HICON*은 높은 지역성(locality)을 가지며 클라이언트 사이에 데이타 집중도가 높은 상황이다. HICON과 비교하여 *HOTCOLD*는 높은 지역성을 가지지만, 클라이언트사이의 데이타 공유정도는 작다. *UNIFORM*은 HICON의 특별한 경우로서 낮은 지역성(locality)을 가지며, 데이타 공유정도가 낮다. 각 매개변수 값은 [2,7]을 기본으로 하여 선택되었다.

3.2 성능평가 분석

이 절에서는 우리의 정량적 성능 평가의 결과들을 기술한다. 성능평가 결과는 1초당 완료되는 트랜잭션의 수인 처리율(throughput)을 기본으로 하였다. 처음의 실험결과는 HICON의 실험 결과이다. HICON은 특정한 데이타를 영역을 (HOT 영역) 많은 클라이언트가 읽고 쓰는 환경을 가정한 모델이다. 그러므로 HICON은 클라이언트의 높은 지역성과 많은 양의 데이타를 공유하게 된다.

그림 2와 그림 3은 *Hot_Bound* = 5%, *Hot_Acc_Prob* = 0.8로 고정시키고, 클라이언트 수를 1에서 25로 증가시켰을 때의 HICON에서의 전체 시스템 철회율과 처리율을 보여준다. 이 실험에서*Hot_Bound*가 DB전체의 5%이기 때문에 많은 트랜잭션들이 공용의 데이타를 억세스할 확률이 커지기 때문에 높은 데이타 집중도(contention)가 생긴다. 이러한 높은 데이타 집중도로 인하여, 그림 2가 보여 주듯이, 클라이언트 수가 늘어날수록, OCC의 철회율이 급격히 증가하였다. 이러한 높은 철회율로 인하여, 클라이언트 수가 5 이상인 경우, OCC는 가장 나쁜 성능을 보였다 (그림 3 참조).

O2PL은 트랜잭션이 완료를 요청하면 모든 요청된 잠

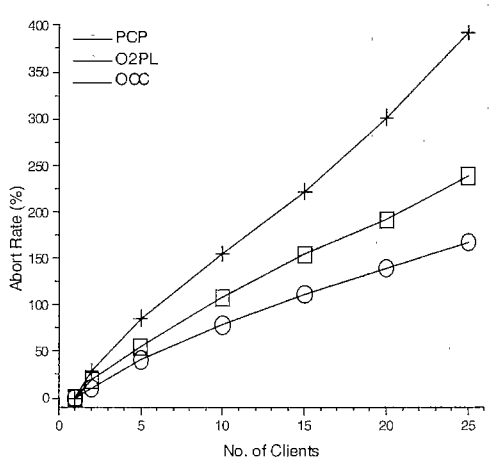

그림 2 철회율(*Hot_Bound*=5%)

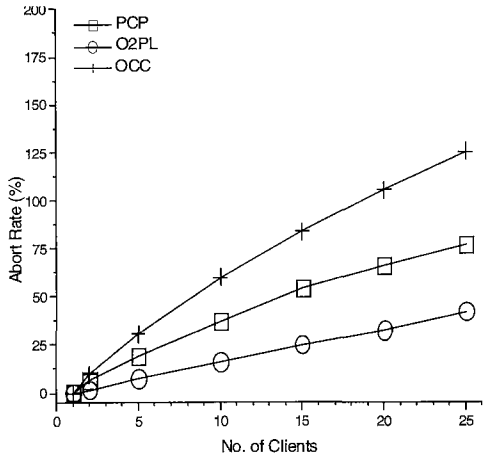

그림 3 철회율(*Hot_Bound*=20%)

금을 얻을 때까지 기다리게 함으로 철회율을 줄이기 때문에, O2PL은 가장 낮은 철회율을 보인다. 그러나, 이러한 트랜잭션의 기다림은 실행중인 트랜잭션의 수를 감소시킨다. 이와 같은 트랜잭션의 기다림은 잠금이 걸린 데이타를 요청한 다른 트랜잭션들도 기다리게 함으로서 성능을 저하시킨다. 대부분이 클라이언트들이 *Hot_Bound*에 있는 데이타들을 접근하기 때문에, O2PL에서 데이타 가용성(availability)은 급격히 감소된다.

그림 2와 그림 3이 보여 주는 또 다른 특징은, 클라이언트 수가 15이상인 경우에 있어서, OCC의 철회율이 O2PL보다 2배 이상 높지만 OCC의 성능이 O2PL의 대하여 80%에 가까운 처리율을 보여 준다는 것이다. 캐쉬를 사용하게 되면, 트랜잭션에 의해 접근되었던 모든 데이타들이 클라이언트에서 사용 가능하게 되므로, 트랜잭션의 재실행은 첫 번째 실행보다 빠르게 진행된다. 이러한 이유로 높은 철회율은 성능에 많은 영향을 미치지 않는다.

PCP의 경우에 있어서, OCC와 마찬가지로 잠금을 사용하지 않기 때문에, O2PL보다는 높은 철회율을 나타낸다. 그러나, 재배열 기법을 사용하기 때문에 PCP는 OCC보다는 낮은 철회율을 보였다. 또한, PCP는 잠금을 사용하지 않기 때문에 O2PL에서 발생 할 수 있는 필요없는 기다림이나 잠금을 유지하기 위한 오버헤드가 없다. 이러한 이유로 인하여 PCP는 가장 좋은 성능을 나타내었다.

그림 4와 그림 5는 *Hot_Bound* = 20%일 때 철회율과 처리율을 나타낸다. *Hot_Bound*를 5%에서 20%로 증가 시켰기 때문에, 첫 번째 실험에 비하여, Hot 영역에 포함된 데이타에 대한 집중도가 줄어든다. 그림 4가 보여 주듯이, 데이타의 집중도가 감소됨에 따라 모든 프로토콜의 철회율이 급격히 감소되었다. 이러한 이유로 인하여, 클라이언트 수가 5보다 큰 경우에 있어서, 전체 프로토콜의 성능 감소 폭이 줄어들었다. 또 다른 특징으로는, 비록 다른 프로토콜보다는 우수한 성능을 나타내었지만, PCP의 성능이 다른 프로토콜들과 비교하여 급격히 증가하지 않았다. 이러한 이유는, 위에 언급하였듯이, 재실행되는 트랜잭션이 성능에 미치는 영향이 크지 않기 때문이다. 그러나 *Hot_Bound*가 변함에 따라, 잠금을 유지하는 시간이 감소함으로서 O2PL이 PCP보다 더 많은 영향을 받았다. 잠금을 유지하는 시간을 줄이는

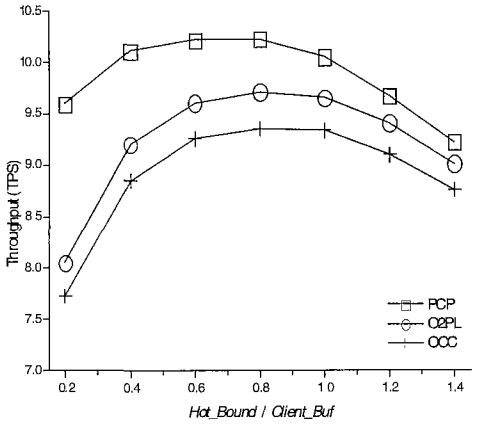

그림 4 처리율

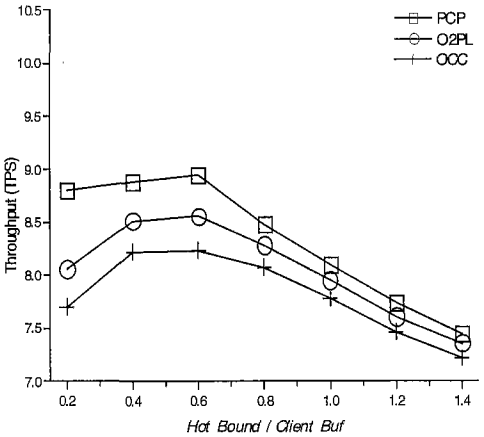

그림 5 처리율

것은 완료를 빨리 처리할 뿐 아니라 데이타의 가용성도 증가시킨다. 또한, OCC의 경우에 있어서 철회율이 2배 이상 감소하였기 때문에, *Hot_Bound* = 5%일 때보다는 좋은 성능을 나타내었다.

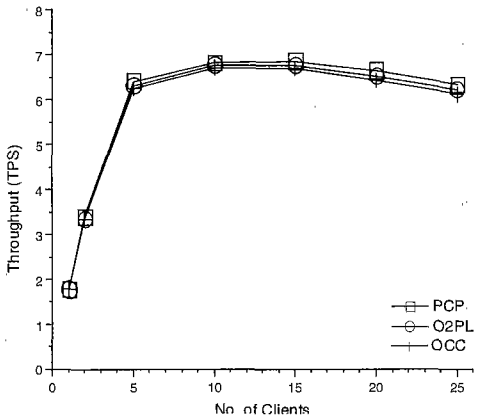

그림 6 처리율(UNIFORM)

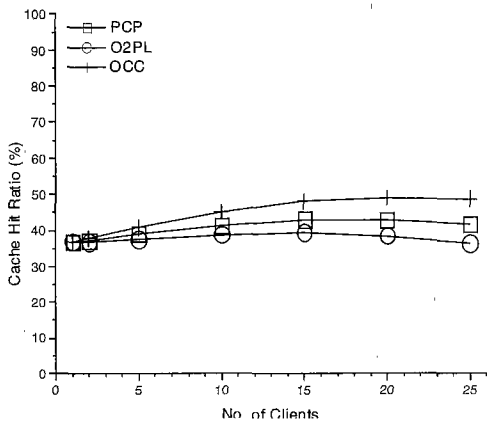

그림 7 캐쉬 적중율(UNIFORM)

본 논문에서는 클라이언트 캐쉬 크기와 HOT 영역의 크기와의 연관성을 찾기 위하여 *Hot_Bound*와 *Hot_Acc_Prob*를 변화시켜가며 실험한 결과를 보여준다. 그림 6과 7은 HOT의 크기, 클라이언트 캐쉬의 크기, HOT의 접근 확률사이의 관계를 보여준다. HICON에 있어서 HOT의 크기는 캐쉬 일관성 유지 기법의 성능에 큰 영향을 미친다. 최적의 HOT의 크기는, 두 개의 매개변수가 정해졌다면, *Client_Buf*에 *Hot_Acc_Prob*를 곱한 값이다. 이는 *Hot_Bound*와 *Hot_Acc_Prob*가 주어졌다면 *Client_Buf*의 최적 값을 정할 수 있다는 의미이다. 그림 6은 *No_Client* = 10, *Hot_Acc_Prob* = 0.8로 고정시키고 *Hot_Bound*를 5%에서 35%로 변화시켜가며 실험한 결과이다. 그림 6이 보여주듯 *Hot_Bound* / *Client_Buf* = 0.8 일 때 최고의 성능을 보여준다. *Hot_Bound*가 20% (*Hot_Bound* / *Client_Buf* = 0.8) 아래로 감소하면, 높은 데이타 집중도에 의해 캐쉬 실패(miss)와 철회되는 트랜잭션이 수가 급격히 증가된다. 반대로 *Hot_Bound*가 20%가 넘으면, *Hot_Bound*가 클라이언트 버퍼크기에 비해 상대적으로 커지므로 지역성(locality)이 감소하게 된다. 이러한 지역성의 감소는 캐쉬의 이점을 감소시킨다.

그림 7은 *Hot_Acc_Prob* = 0.6일 때의 처리율을 보여준다. *Hot_Acc_Prob*가 0.8에서 0.6으로 감소함에 따라 지역성의 감소로 인하여 모든 방법의 성능이 현저히 감소하였다. 그러나, 우리의 분석대로, 두 방법의 성능이 *Hot_Bound* / *Client_Buf* = 0.6에서 정점에 도달했다. 다시 그림 3과 5를 고려해 보자. 비록 *Hot_Bound*가 5%인 경우(그림 3)와 20%인 경우(그림 5)의 결과만을 보여주었지만, 우리는 *Hot_Bound*를 변화시켜가며 다양한 실험을 하였다. 이 실험을 통하여 보여준 그래프 중, 그림 5의 결과가 주어진 값에서 최고의 성능을 보여주었다. 이러한 결과는 그림 6과 그림 7을 통한 우리의 분석과 일치한다.

다음은 UNIFORM에 대한 실험 결과이다. 이 모델은 캐쉬의 장점이 없는 모델이다. UNIFORM은 HICON에서 *Hot_Bound*가 *Hot_Acc_Prob*와 같은 경우이므로, HICON의 특별한 경우이다. 그러므로, *Hot_Bound*가 상대적으로 크기 때문에 적은 지역성을 가지며 캐쉬를

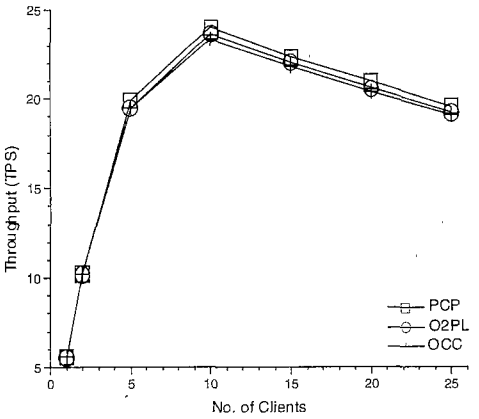

그림 8 처리율(HOTCOLD)

사용하는 이득이 적다.

그림 8은 *Write_Prob* = 20%이고, 클라이언트 수를 1에서 25로 증가시키면서 실험한 결과이다. HICON에서는 PCP가 높은 가용성을 제공하였다. 그러나, UNIFORM에서는 지역성이 적으므로, PCP의 장점이 작아진다. 그림 8이 보여주듯, 모든 프로토콜이 비슷한

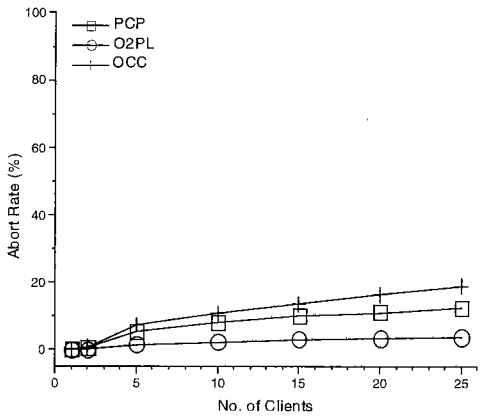

그림 9 철회율(HOTCOLD)

성능을 나타내었다.

그림 9는 UNIFORM에서의 캐쉬 적중율(Hit ratio)을 나타낸다. 그림 9가 보여주는 가장 큰 특징은 모든 프로토콜이 비슷한 성능을 보일지라도, 캐쉬 적중율은 OCC가 다른 프로토콜보다 높게 나타난다는 것이다. 이는 철회로 인한 캐쉬 적중율이 향상되는 현상 때문에 발생한다. 이미 언급 한 바와 같이 OCC의 철회율이 다른 프로토콜보다 높다. 그러나, 재실행되는 트랜잭션들은 일부의 데이타를 자신이 속한 클라이언트의 캐쉬에 가지고 있음으로서 캐쉬 적중율이 높아지게 된다. 이러한 현상으로 인하여 OCC가 비록 높은 철회율을 보이지만 다른 프로토콜과의 성능 차이가 없게 된다.

마지막 실험은 HOTCOLD이다. 그림 10은 *Write_Prob* = 20%이고, 클라이언트 수를 1에서 25로 증가시키면서 실험한 결과이다. HOTCOLD는 서로 겹치지 않는 자신만의 고유 영역80%이상을 접근하게 됨으로 높은 지역성을 가진다. 또한, 데이타들이 클라이언트 사이에 잘 나뉘어진 모델을 가정한 것이다. 이러한 이유로 HOTCOLD는 데이타 충돌 확률이 작다. 이러한 이유로 인하여, 지금까지의 실험 중, 모든 프로토콜이 가장 좋은 성능을 보여 주었다. 그림 11은 철회율을 나타낸다. 다른 실험과 비교 해 보면 가장 작은 철회율을 보인다. 다른 실험들과 비교하여 철회율이 작기 때문에, 그림 10이 보여 주듯, 성능의 차이가 거의 없다.

4. 결 론

이 논문에서 완료된 트랜잭션들의 직렬화 가능성을 제공하는 새로운 낙관적 캐쉬 병행수행 제어 기법을 제안하였다. O2PL과 비교해 볼 때, 제안하는 기법은 데이타의 가용성을 높이고, 필요없는 연산을 줄이며, 교착상태 검증 알고리즘이 필요없다는 장점을 가진다. 교착상태 검증 알고리즘이 없다는 것은 실제구현에 있어서 단순함을 제공한다. 또한, OCC와 비교 해볼 때, 제안하는 기법은, 각 데이타 당 단일 버전을 유지하면서도, 재배열을 시킬 수 있기 때문에 철회율을 줄인다.

본 논문에서 정량적 평가를 통하여 제안하는 기법을 OCC와 O2PL과 비교하였다. 우리는 정량적 평가에서 데이타 집중도가 낮은 상황에서는 모든 낙관적인 방법을 기법의 성능 차이가 없다는 것을 보였다. 그러나 데이타 집중도가 높은 상황에서는 O2PL은 필요없는 기다림을, OCC는 필요없는 철회를 만들기 때문에 PCP보다 낮은 성능을 나타내었다. 우리는 정량적 평가를 통하여 잠금을 사용하는 오버헤드 없이도 PCP가 O2PL보다 안정된 성능을 나타낸다는 것을 보였다.

참 고 문 헌

[1] M. J. Carey, M. J. Franklin, M. Livny and E. Shekita, "Data Caching Tradeoffs in Client-Server DBMS Architecture," *Proc. of the ACM SIGMOD International Conference on the Management of Data*, May, 1991.

[2] M. J. Franklin and M. J. Carey, "Client-Server Caching Revisited," *Proc. of the International Work-shop on Distributed Object Management*, August, 1992.

[3] A. S. Tannenbaum, *Distributed Operating Systems*, Prentice Hall, 1995.

[4] B. Liskov, M. Day and L. Shrira, "Distributed Object Management in Thor," *Proc. of the International Workshop on Distributed Object Management*, August, 1992, (Published as Distributed Object Management, Ozsu, Dayal, Valduriez, Morgan Kaufmann, 1994).

[5] B. Nitzberg and V. Lo, "Distributed Shared Memory: A Survey of Issues and Algorithms," *IEEE Computer*, 20(8), August, 1991.

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

rence on Data Engineering, February, 1996.
[7] M. J. Franklin, "Caching and Memory Management in Client-Server Database Systems," *Phd. Thesis*, Dept. of Computer Science, University of Wisconsin, July, 1993.
[8] A. J. Bernstein and P. M. Lewis, *Concurrency in Programming and Database Systems*, Jones and Bartlett Publishers, 1993.
[9] P. A. Bernstein, V. Hadzilacos and N. Goodman, *Concurrency Control and Recovery in Database Systems*, Addison-Wesley, 1987.
[10] Wilkinson, K. and Neimat, "Maintaining consistency of client-cached data," *In Proc. of the Conf. on Very Large Data Bases (VLDB)*, pp. 122-134, 1990.
[11] M. J. Carey, M. J. Franklin, M. Livny, and Shekita, "Data caching tradeoffs in client-server DBMS architectures," *In Proc. of the ACM SIGMOD Conf. on Management of Data*, pp. 357-366, 1991.
[12] A. Adya, R. Gruber, B. Liskov, and U. Maheshwari, "Efficient optimistic concurrency control using loosely synchronized clocks," *In Proc. of the ACM SIGMOD Conf. on Management of Data*, pp. 23-34, 1995.
[13] Y. Wang and Rowe, "Cache consistency and concurrency control in a client-server DBMS architecture," *In Proc. of the ACM SIGMOD Conf. on Management of Data*, pp. 367-377, 1991.
[14] M. J. Franklin, M. J. Carey, and M. Livny, "Local disk caching in client-server database systems," *In Proc. of the Conf. on Very Large Data Bases (VLDB)*, pp. 543-554, 1993.
[15] P. S. Yu, H. Heiss, and D. M. Dias, "Modeling and Analysis of a Time-Stamp History Based Certification Protocol for Concurrency Control", *IEEE Transactions on Knowledge and Data Engineering*, vol. 3. no. 4, pp. 525-537, Dec. 1991.
[16] S. Cho, K. Y. Bae and C. Hwang, "A Certification Protocol with Low Space Overhead," *International Conference on Parallel and Distributed Systems*, pp. 67 - 74, 1998.

조 성 호
1994년 한국외국어대학교 전산학과 졸업(이학사). 1997년 고려대학교 전산과학과 졸업(이학석사). 2000년 고려대학교 전산과학과 졸업(이학박사). 현재 천안대학교 정보통신학부 교수. 관심분야는 병행수행제어, 이동 컴퓨팅, 분산 운영체제

황 정 현
1998년 고려대학교 이과대학 컴퓨터학과 졸업(이학사). 2000년 고려대학교 전산과학과 졸업(이학석사). 현재 미국 Brown Univ., Dept. of Computer Science 박사과정 재학중. 관심분야는 이동 컴퓨팅, 분산 운영체제, 센서 네트웍