백쉬프팅 기법을 이용한 캐쉬 유지 규약의 분석

Analysis of a Cache Management Protocol Using a Back-shifting Approach

조성호
한신대학교 정보통신학과

Sung-Ho Cho(zoch@hs.ac.kr)

요약

클라이언트-서버 컴퓨팅에서 서버의 과부하를 줄이기 위하여 각 클라이언트는 재사용을 위하여 자신만의 캐시를 유지한다. 캐시 유지 규약을 위한 비관적 접근법은 모든 잠금을 획득하기 전까지 완료될 수 없기 때문에 필요 없는 기다림을 만든다. 또한, 낙관적 접근법은 필요 없는 철회를 일으킨다. 본 논문은 이와 같은 단점을 극복할 수 있는 낙관적인 규약을 제안한다. 본 논문에서는 잘 알려진 규약들과 제안하는 기법과의 정량적 성능평가를 보여준다. 성능평가는 웹의 성능분포를 나타낼 수 있는 Zipf 작업부하에서 수행되었다. 본 성능평가를 통하여 제안하는 기법이 적은 오버헤드를 가지고 좋은 성능을 나타낸다는 것을 보인다.

■ 중심어 : | 캐시 유지 규약 | 낙관적 규약 | 비관적 규약 | 백쉬프팅 접근법 |

Abstract

To reduce server bottlenecks in client-server computing, each client may have its own cache for later reuse. The pessimistic approach for cache management protocol leads to unnecessary waits, because, it can not be commit a transaction until the transaction obtains all requested locks. In addition, optimistic approach tends to make needless aborts. This paper suggests an efficient optimistic protocol that overcomes such shortcomings. In this paper, we present a simulation-based analysis on the performance of our scheme with other well-known protocols. The analysis was executed under the Zipf workload which represents the popularity distribution on the Web. The simulation experiments show that our scheme performs as well as or better than other schemes with low overhead.

■ Keyword : | Cache Management Protocol | Optimistic Protocol | Pessimistic Protocol | Back-shifting Approach |

I. 서 론

최근 데이터베이스 시스템은 지역적으로 떨어진 사용자들에게 효율적인 데이터 공유를 지원하기 위하여 클라이언트/서버(client/server) 모델을 사용하고 있다. 이 모델의 단점은 클라이언트들에 의해 요청된 많은 양

* 본 연구는 2005년 한신대학교 학술연구비 지원에 의하여 연구되었습니다.

접수번호 : #050712-001
접수일자 : 2005년 07월 12일
심사완료일 : 2005년 08월 11일
교신저자 : 조성호, e-mail : zoch@hs.ac.kr

의 데이터로 인하여 네트워크나 서버에 병목현상(bottleneck)이 발생하는 것이다. 이러한 문제를 완화시키기 위하여, 각 클라이언트들은 나중에 다시 사용하기 위한 일정양의 데이터를 자신의 캐시(cache)에 유지하고 있다[1-7][10]. 캐시를 사용하게 되면, 트랜잭션의 의미를 이용하여 클라이언트 캐시에 있는 데이터들의 일관성을 유지시켜주는 캐시 유지 규약(management protocol)을 필요로 한다[1-5].

현재까지 제안된 캐시 유지 규약은 크게 비관적인(pessimistic) 방법과 낙관적인(optimistic) 방법으로 나눌 수 있다[2][6][7]. 비관적인 방법은 캐시에 있는 데이터 사용시 서버에게 잠금(lock)을 요청 한 후, 잠금을 획득하면 데이터를 사용할 수 있는 방식이다. 대표적인 기법으로 CBL(Callback Locking)이 있다.

낙관적인 방법은 잠금 없이 캐시에 있는 데이터를 사용하다가 마지막 단계에서 잘못된 데이터를 읽었을 경우 철회(abort)하는 방식이다. 낙관적인 방법은 비관적인 방법에 비하여 적은 통신 오버헤드(overhead)를 가지므로 높은 메시지 비용이나 적은 충돌(conflict) 상황에서는 비관적인 방법보다 우수한 성능을 보인다[7-9].

낙관적인 기법의 가장 큰 단점은 철회에 민감하다는 것이다[8]. 이를 개선하여 비관적인 방법과 낙관적인 방법을 동시에 사용하는 기법이 O2PL(Optimistic Two Phase Locking)이다. 그러나 O2PL은 한 트랜잭션이 완료(commit)를 요청하면 모든 잠금이 얻어질 때까지 해당 트랜잭션은 완료될 수 없기 때문에 필요 없는 기다림을 만든다.

본 논문에서는 백쉬프팅(back-shifting) 기법을 이용하여 캐시 유지 규약의 성능을 높일 수 있는, 효율적인 낙관적 캐시 일관성 유지 기법을 제안한다. 백쉬프팅 기법은 멀티버전(multi-version)[1]과 같은 기법에서 사용되는 방식으로 철회율을 줄일 수 있지만 각 데이터당 많은 양의 다른 버전을 유지해야하는 단점이 있다. 제안하는 기법에서는 단일버전에서 이러한 기법을 적용함으로서 오버헤드를 줄이고 성능을 향상시켰다.

본 논문에서는 다른 기법과의 비교를 위하여 성능평가를 실시한다. 성능평가는 *Zipf* 작업부하(Workload)에서 이루어졌는데, *Zipf* 작업부하는 웹의 사용자환경을 잘 나타내는 *Zipf*의 법칙(*Zipf*'s law)[9]을 기반으로 만들어졌다. 이러한 성능평가를 통하여 제안하는 기법이 다른 기법보다 성능이 좋다는 것을 보인다.

II. 백쉬프팅 기반의 캐시 프로토콜

본 논문에서는 표기를 단순하게 하기 위하여 타임스탬프(time-stamp)를 T로 집합(set)을 S로 표시하기로 한다. 또한, 아래첨자는 분류자(identifier)를 위첨자는 타임스탬프와 집합의 종류(type)를 나타낸다.

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 원할 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, 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 필요 없는 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 돼야 한다면, 해당 트랜잭션은 자신이 읽은 데이터의 타임스탬프들 보다 작은 값으로는 재배열 될 수 없다. 그러므로 하한 타임스탬프 T^L은 자신의 읽은 데이터의 타임스탬프 중 가장 큰 값을 가지고 있어야 한다. 이를 위해서, 트랜잭션 X가 데이터 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.

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 데이터의 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 데이터를 캐시에서 지워버리고, 응답메시지를 서버에게 보낸다. 모든 응답메시지를 받으면, 서버는 집합 S_X^R에 있는 각각의 데이터 D_j에 대하여 타임스탬프 $D_j.T^r$을 완료 타임스탬프 T_X^C로, 집합 S_X^W에 있는 각각의 데이터 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 데이터 값이 변했으므로, 이러한 값을 읽은 트랜잭션들은 재배열되어야만 한다. 한 트랜잭션이 재배열되기 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 트랜잭션 X에 의해 발생된 무효화 메시지를 받았을 때, 트랜잭션 Y가 무효화된 데이터를 접근했다면, 클라이언트는 다음과 같은 규칙에 의해 타임스탬프 T_Y^U를 갱신한다.

- 집합 S_X^R에 있는 각 데이터 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 데이터 D_k는 필요 없는 연산을 줄이기 위하여 집합 S_Y^I에 넣는다.

제안하는 기법에서 재배열을 위한 타임스탬프는 T^L과 T^U 사이에 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 데이터를 쓰기 연산을 하려고 한다면, 쓰기-쓰기 충돌을 방지하기 위해, 트랜잭션 Y는 철회된다.

무효화 된 데이터를 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. 특정한 값이 아닌 극소 값을 사용하게 되면 완료 된 트랜잭션들 사이에 재배열될 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 발생할 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 데이터 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 데이터의 복사본을 가진 모든 클라이언트에게 T_Y^C, S_Y^R, S_Y^W를 보낸다. 모든 응답 메시지를 받으면, 서버는 집합 S_Y^R에 있는 데이터 D_j에 대하여 타임스탬프 $D_j.T^r$이 타임스탬프 T_Y^C보다 작은 경우에 만 $D_j.T^r$을 T_Y^C로 설정한다. 또한, 서버는 집합 S_Y^W에 있는 데이터 D_k에 대하여 타

임스탬프 $D_k.T^w$가 타임스탬프 $T_Y{}^C$보다 작은 경우에만 $D_k.T^w$을 $T_Y{}^C$로 설정한다.

III. 정량적 평가 및 결과 분석

1. 정량적 평가 모델

본 절에서는 제안하는 기법(RCP)과 O2PL 그리고 CBL을 비교한다. [표 1]은 정량적 평가에 사용된 매개변수 및 그 값을 나타낸다. 본 실험은 과거 연구와의 연속성을 위하여 [8]에 나타난 성능평가 모델에 근접하게 제작되었으며 사용된 매개변수 값도 유사하도록 설정되었다. 그로인하여 O2PL과 CBL의 성능평가는 [8]에 나타난 결과와 일치한다.

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 캐시 관리자는 LRU 페이지 대체 알고리즘을 사용한다. 각 클라이언트는 트랜잭션 생성기로부터 받은 트랜잭션을 한 번에 한 개씩 실행한다. 클라이언트와 비교해보면, 서버는 다음과 같은 차이점을 갖는다. 서버는 중앙처리장치(CPU) 뿐만 아니라 디스크를 관리하며, 페이지의 복사본을 가진 클라이언트의 정보를 저장하며, O2PL을 위해서 잠금을 유지한다.

만약, 트랜잭션이 철회되면 그 트랜잭션은 다시 실행되며, 첫 번째 실행 때와 같은 데이터에 접근하게 된다. 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 데이터에 접근하게 된다. *Ex_Tr*은 트랜잭션들이 도착하는 시간을 나타내며, *Ex_Op*는 연산사이의 평균시간을 나타낸다. 이 실험에서는 두 매개변수를 0으로 정하였다. 데이터베이스 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 데이터를 다시 쓸 확률은 *Write_Prob*에 의해 결정된다.

표 1. 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	5%, 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.

*Client_Buf*와 *Server_Buf*는 각각 클라이언트와 서버의 버퍼 크기를 나타낸다. 본 실험에서는 각 클라이언트가 비교적 작은 크기의 캐시(전체 페이지의 5%)의 경우와 큰 크기의 캐시(전체 페이지의 25%) 두 가지 경우를 가정하여 실험하였다. CPU의 성능은 *CPU_MIPS*에 의존하며 [표 1]에 명시된 연산을 수행하게 된다. 평가에 사용된 CPU는 두 개의 우선순위(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 단순

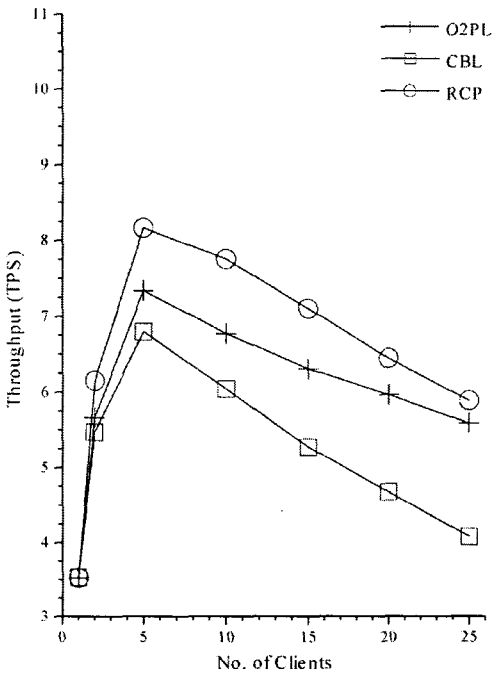

그림 1. 성능(전체페이지의 5%)

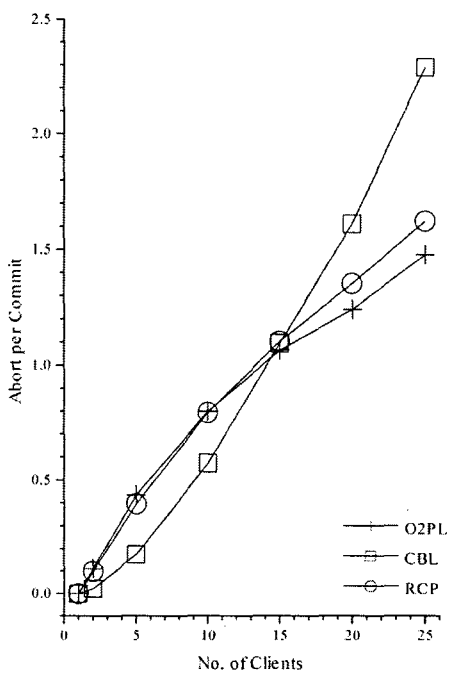

그림 2. 철회율(전체페이지의 5%)

한 FIFO(First In First Out) 서버라고 가정하였다. 대역폭은 충돌로 인한 감소분을 고려하여 이더넷(Ethernet)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.

우리의 정량적 평가 모델에서는 *Zipf* 작업부하를 사용하였다. *Zipf* 작업부하는 *Zipf*의 법칙[9]에 의하여 만들어 졌다. *Zipf*의 법칙에 의하면 가장 많이 사용되는 데이터와 가장 적게 사용되는 데이터를 순서대로 배열하면, 각 데이터의 참조 빈도(number of references)는 각 데이터 배열의 역 순위를 따른다는 것이다. 그에 따라, *Zipf* 작업부하에서 각 데이터 D_i(i=1에서 *DB_Size*)의 참조될 확률은 $1/i$가 된다.

Zipf 작업부하는 웹 시스템의 서버부하를 가장 잘 표현하는 것으로 알려져 있으며, 처음 부분의 데이터가 가장 많이 선택되어지는 데이터이기 때문에 [8]의 실험에서 HOT_COLD 작업부하와 유사한 특성을 보인다. HOT_COLD 작업부하란 상위 20%데이터에 80%의 참조빈도가 배정되고 하위 80%의 데이터의 20%의 참조빈도가 배정되는 환경이다. HOT_COLD 작업 부하는 20%의 데이터에 대하여 80%의 참조빈도가 일정하게 분포되어 있는 환경인데 반하여 *Zipf* 작업부하는 각 데이터마다 참조빈도가 다르도록 만들어져 있다.

2. 성능평가 분석

[그림 1]은 작은 캐시(전체 페이지의 5%)에서의 결과이다. 본 작업부하에서는 작은 캐시로 인하여 매우 높은 데이터 경쟁상황(data contention)이 나타난다. [그림 1]이 나타내듯이 제안하는 기법이 가장 좋은 성능을 보이며 O2PL은 조금 낮은 성능을 보이고 CBL이 가장 낮은 성능을 보인다. 5명이하의 클라이언트에서는 모든 기법의 성능이 증가하지만, 5명을 넘어서면서 클라이언트 수가 증가됨에 따라 모든 기법의 성능이 하강하는 "thrashing" 현상이 발생한다. 이러한 현상은 [그림 2]

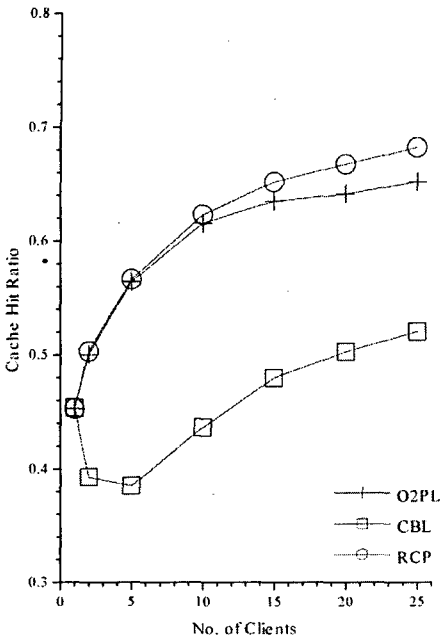

그림 3. 캐시 적중율(전체페이지의 5%)

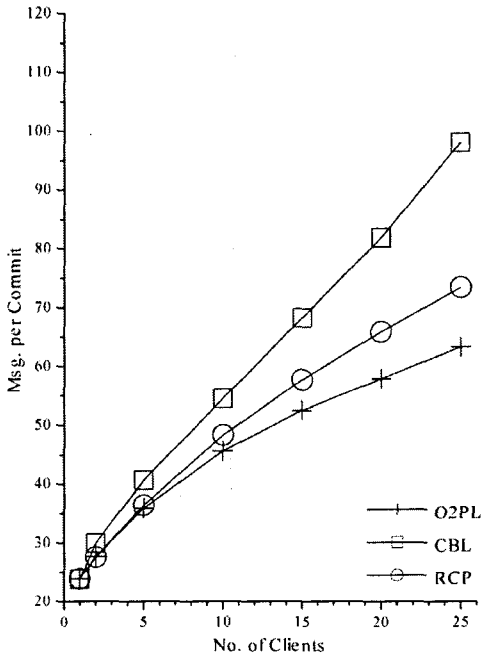

그림 4. 메시지(전체페이지의 5%)

의 철회율과 관련 있다.

일반적으로 CBL과 같은 비관적인 기법의 철회율은 낙관적인 기법보다 작은 것으로 알려져 있다. 그러나 [그림 2]가 보여주듯이, CBL의 철회율은 다른 두 개의 기법보다 높게 나타나다. 이러한 현상은 O2PL이 쓰기 잠금을 얻는 시점이 실행의 맨 마지막 단계까지 이루어지는 반면에, CBL은 실행의 중간에 이루어지기 때문에 발생한다. O2PL은 쓰기 잠금을 실행의 맨 마지막 시점에 한꺼번에 요청하기 때문에 쓰기 잠금을 유지하는 시간이 짧지만, CBL은 실행 도중 필요할 때마다 잠금을 요청하기 때문에 쓰기 잠금을 유지하는 시간이 길어져서 충돌이 많아지게 된다.

또 다른 특징으로, 일반적으로 낙관적인 기법의 철회율이 비관적인 기법보다 높지만, RCP가 백쉬프팅 기법을 사용함에 따라 CBL보다는 낮은 철회율을 보여준다. 비록, O2PL 기법이 RCP보다는 낮은 철회율을 보여주지만. 잠금을 사용함에 따라 실제적으로 작동하는 트랜잭션의 수가 감소하게 되어 데이터의 가용성이 떨어지는 현상이 발생한다. 뿐만 아니라, 잠금을 사용하더라도 교착상태(dead lock)로 인하여 트랜잭션이 철회되기 때문에 필요 없는 연산이 많아지게 된다.

RCP의 철회율은 O2PL보다 약간 높지만, 잠금을 사용하지 않아 데이터의 가용성이 높아지고, 트랜잭션 실행 중간에 충돌이 일어난 트랜잭션들을 미리 철회시킴으로서 가장 좋은 성능을 보여준다. 이러한 특징으로 인하여 [그림 3]과 같이 RCP는 높은 데이터 가용성능을 보이게 된다.

[그림 4]는 메시지율을 보여준다. CBL이 높은 철회율을 보이기 때문에 다른 두 기법보다 많은 메시지를 사용하게 된다. CBL는 높은 철회율, 낮은 데이터 가용율, 많은 메시지 사용으로 인하여 낮은 성능을 나타낸다.

[그림 5]는 큰 캐시(25\% of DB)를 사용하여 실험한 결과이다. [그림 1]과 비교하여보면 캐시의 크기가 커졌기 때문에 3가지 기법 모두 성능이 향상되었다. 그러나 캐시가 5배 커진 것에 비하여 성능은 크게 증가하지 않았다. 그 이유는, *Zipf* 작업부하이 매우 높은 서버의 부하를 가정하고 있기 때문이다.

[그림 6]은 큰 캐시 설정에서의 캐시 적중률을 보여준

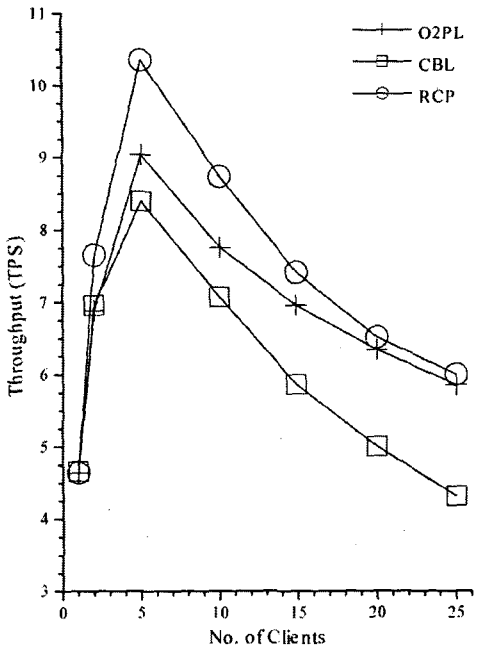

그림 5. 성능(전체페이지의 25%)

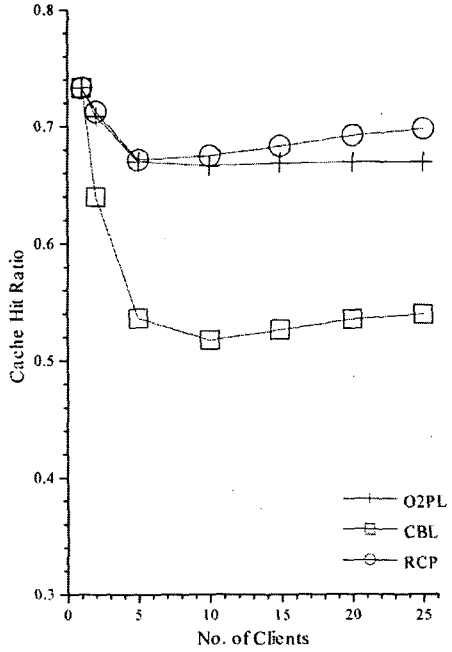

그림 6. 캐시 적중율(전체페이지의 25%)

다. [그림 3]과 비교해보면 캐시 적중률은 다른 모양을 보여준다. 우선 CBL의 경우 여전히 낮은 캐시 적중률을 보인다. 비록 캐시의 크기가 커졌지만, 쓰기 잠금을 오랫동안 유지하다가 철회되기 때문에 캐시에 있는 많은 양의 데이터가 쓸모없어지는 현상이 발생한다. 그러나 RCP와 O2PL의 경우 철회가 많이 일어나기는 하지만, 철회가 일어나고 난 후, 다시 시작하는 시점에서 최신의 데이터를 캐시에 유지하고 있기 때문에 캐시의 적중률이 CBL보다 높게 나타나게 된다.

Ⅳ. 결 론

본 논문에서 완료된 트랜잭션들을 백쉬프팅 기법을 이용하여 재배열 시키는 낙관적 캐시 유지 기법을 제안하였다. 낙관적 기법인 O2PL과 비교해 볼 때, 제안하는 기법은 데이터의 가용성을 높이고, 필요 없는 연산을 줄이며, 교착상태 검증 알고리즘이 필요 없는 장점을 가진다. 또한, CBL과 비교 해볼 때, 제안하는 기법은, 잠금을 유지하지 않기 때문에 데이터의 가용율을 높인다.

본 논문에서 정량적 평가를 통하여 제안하는 기법을 CBL, O2PL과 비교하였다. 성능평가에서는 *Zipf* 작업부하를 사용하여 웹 환경과 같은 실제적인 작업환경에서 실험을 하였다. 정량적 평가 결과, CBL은 잠금과 낮은 데이터 가용율로 인하여 가장 낮은 성능을 보였으며 O2PL은 제안하는 기법보다 낮은 철회율을 보여지만 데이터의 가용율이 떨어져 제안하는 기법보다 조금 낮은 성능을 보였다. 제안하는 기법은 O2PL보다는 철회율이 높지만 데이터의 가용율을 높이고 필요 없는 연산을 줄임으로서 가장 좋은 성능을 나타내었다.

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] D. Barbara, "Mobile Computing and

Database - a Survey," IEEE Transactions on Knowledge and Data Engineering, Vol.11, No.1, pp.108-117, 1999.

[3] 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.

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

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

[6] A. Adya, R. Gruber, B. Liskov, and U. Maheshwari, "Efficient optimistic concurrency control using loosely synchronized clocks," Proceedings of the ACM SIGMOD Conf. on Management of Data, pp.23-34, 1995.

[7] M. J. Carey, M. J. Franklin, M. Livny, and Shekita, "Data caching tradeoffs in client-server DBMS architectures," Proceedings of the ACM SIGMOD Conf. on Management of Data, pp.357-366, 1991.

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

[9] G. K. Zipf, *Human Behavior and the Principles of Least Effort, Reading*, Mass., Addison Wesley, 1949.

[10] V. Almeida, A. Bestavros, M. Crovella, and A. D. Oliveira, "Characterizing reference locality in the WWW," Proceedings of the 1996 International Conference on Parallel and Distributed Information Systems, pp.92-103, 1996.

저 자 소 개

조 성 호(Sung-Ho Cho) 정회원

- 1994년 2월 : 한국외국어대학교 전산과(이학사)
- 1997년 2월 : 고려대학교 컴퓨터학과(이학석사)
- 2000년 2월 : 고려대학교 컴퓨터학과(이학석사)
- 2000년~2001년 : (주)MPSCOM 기술개발 이사
- 2001년~2002년 : 천안대학교 전임교수
- 2002년~현재 : 한신대학교 조교수 및 산학연센터장

<관심분야> : e러닝, 분산시스템, 데이터베이스