블록 분류에 기반한 데이타베이스의 효율적 캐쉬 관리 기법

(Efficient Cache Management Scheme in Database based on Block Classification)

신 일 훈 † 고 건 ††
(Ilhoon Shin) (Kern Koh)

요 약 LRU는 비균등 참조 패턴을 보이는 데이타베이스의 캐쉬 교체 정책으로 적합하지 않음에도 불구하고, 적절한 대안 부재로 인해 대부분의 데이타베이스 시스템에서 캐쉬 교체 정책으로 이용되어 왔다. 본 논문은 실제 데이타베이스 트레이스 분석을 통해 데이타베이스의 블록 참조 패턴을 추출하고, 이를 바탕으로 새로운 캐쉬 교체 정책을 제안한다. 데이타베이스의 트레이스 분석 결과, 전체 시간동안 거의 참조되지 않는 블록이 전체의 70% 가량을 차지하였다. 그리고 블록의 재참조 가능성에 미치는 최근도(recency)의 영향력이 시간적 지역성으로 인해 처음엔 강력하지만, LRU 스택거리가 증가함에 따라 급격히 감소하여, 결국엔 사라지는 현상을 관찰하였다. 이 관찰을 토대로, 본 논문은 전체 블록을 재참조 가능성과 재참조 가능성에 대한 최근도의 영향력을 기준으로 4개의 그룹으로 분류하고, 각 그룹의 참조 특징에 적합한 우선순위 평가 방법을 운용하는 *RCB(Reference Characteristic Based)* 캐쉬 교체 정책을 제안한다. RCB 정책은 재참조 가능성이 극히 낮은 블록은 다른 블록보다 캐쉬에서 빨리 교체하며, 오랜 시간 참조되지 않은 블록에 대해서는 참조빈도에 의거하여 블록의 우선순위를 결정한다. 실제 데이타베이스 워크로드를 통한 모의실험 수행 결과, RCB 정책은 기존의 다른 교체 정책들(LRU, 2Q, LRU-K, LRFU)보다 우수한 성능을 나타냈으며, 특히 LRU에 비해서는 약 5 ~ 12.7% 정도, 캐쉬적중실패 회수를 줄였다. RCB 정책의 시간복잡도는 $O(1)$로서 LRU, 2Q 등과 동일하며, 캐쉬 크기를 N이라 할 때 시간복잡도가 $O(log_2N)$인 LFU와 LRU-K, 그리고 $O(1)$부터 $O(log_2N)$ 사이의 값을 갖는 LRFU보다 우수하다.

키워드 : 캐쉬교체, 최근도, 참조빈도, 블록분류

Abstract Although LRU is not adequate for database that has non-uniform reference pattern, it has been adopted in most database systems due to the absence of the proper alternative. We analyze database block reference pattern with the realistic database trace. Based on this analysis, we propose a new cache replacement policy. Trace analysis shows that extremely non-popular blocks take up about 70 % of the entire blocks. The influence of recency on blocks' re-reference likelihood is at first strong due to temporal locality, however, it rapidly decreases and eventually becomes negligible as stack distance increases. Based on this observation, *RCB(Reference Characteristic Based)* cache replacement policy, which we propose in this paper, classifies the entire blocks into four block groups by blocks' recency and re-reference likelihood, and operates different priority evaluation methods for each block group. RCB policy evicts non-popular blocks more quickly than the others and evaluates the priority of the block by frequency that has not been referenced for a long time. In a trace-driven simulation, RCB delivers a better performance than the existing polices(LRU, 2Q, LRU-K, LRFU). Especially compared to LRU. It reduces miss count by 5~12.7%. Time complexity of RCB is $O(1)$, which is the same with LRU and 2Q and superior to LRU-K($O(log2N)$) and LRFU($O(1)$ ~ $O(log2N)$).

Key workds : Cache Replacement, Recency, Frequency, Block Classification

· 이 연구는 두뇌한국 21의 지원을 받았다.
† 비 회 원 : 서울대학교 전기컴퓨터공학부
jcjc@oslab.snu.ac.kr
†† 종신회원 : 서울대학교 전기컴퓨터공학부 교수
kernkoh@oslab.snu.ac.kr
논문접수 : 2000년 12월 8일
심사완료 : 2002년 4월 15일

1. 서 론

다른 범주의 응용 프로그램들은 서로 다른 참조 패턴을 보이지만, 데이타베이스를 비롯한 대부분의 시스템에서 LRU가 대표적인 캐쉬 교체 정책으로 채택되어 왔다.

LRU는 참조의 최근도(recency)에 의해 블록의 우선순위(priority)를 결정하므로 LRU 스택 모델과 같이 최근에 참조된 블록일수록 더 짧은 미래참조거리(forward distance)를 갖는 참조 패턴에서 최적의 성능을 보인다. 데이타베이스는 블록들이 서로 다른 확률로 참조되는 비균등 참조 패턴(non-uniform reference pattern)을 보이므로, 블록의 참조빈도를 고려하지 않는 LRU는 데이타베이스의 캐쉬 교체 정책으로 적합하지 않을 가능성이 크다[1]. LRU가 데이타베이스의 캐쉬 교체 정책으로 적합하지 않기 때문에, 그리고 LRU가 자체적으로 몇 가지 단점을 갖고 있기 때문에, 데이타베이스를 비롯한 여러 범주의 분야에서 이를 극복하기 위한 연구[1-5]가 진행되어 왔다.

LRU의 단점은 다음과 같다. 첫째, LRU는 인기 있는 블록과 인기 없는 블록을 차별하지 않는다. 재참조 될 가능성이 거의 없는 블록도 한번 참조되고 난 후에는 긴 시간 동안 캐쉬에 저장된다[1, 2]. 데이타베이스는 참조빈도(frequency)가 극히 낮은 블록이 상당한 비율을 차지하므로[1] 이들을 캐쉬에 오래 저장하는 것은 한정된 메모리 자원의 낭비이다. 둘째, LRU는 참조의 최근도, 즉 과거참조거리 (backward distance)에 의해서 블록 교체를 결정하므로 캐쉬에 저장할 수 있는 블록의 과거참조거리에 제한이 따른다.[2] 즉, 참조간 간격(Inter-Reference Gap)이 캐쉬 크기를 초과하는 블록은 재참조 되기 전에 캐쉬에서 교체될 가능성이 크다. 결국, 참조간 간격이 캐쉬 크기를 초과하는 반복 참조 패턴에서는 계속해서 캐쉬적중실패(cache miss)를 낳을 가능성이 크게 된다[4]. 셋째, LRU는 최근도 외의 다른 블록 속성은 전혀 고려하지 않는데, 재참조 가능성에 미치는 최근도의 영향은 스택거리(stack distance)가 증가함에 따라 급격히 감소한다.[3] 따라서 스택거리가 충분히 큰 블록의 경우 참조빈도와 같은 다른 블록 속성이 재참조 가능성에 더 영향을 미칠 수 있다. 가령, *A*, *B* 두 블록 모두 오랜 시간 참조되지 않았다면, 이들 블록의 재참조 가능성은 과거의 참조빈도에 영향 받을 수 있다.

위에서 열거한 LRU의 한계를 보완하기 위한 새로운 캐쉬 교체 정책은 다음과 같은 특징을 가져야 한다. 첫째, 인기 있는 블록과 인기 없는 블록에게 다른 우선순위를 부여해야 한다. 즉 인기 없는 블록은 페널티(penalty)를 주어서 캐쉬에서 빨리 교체한다. 둘째, 캐쉬에 유지할 수 있는 블록의 과거참조거리에 제한을 두지 않아야 한다. 즉 참조간 간격이 캐쉬 크기보다 훨씬 큰 블록도 재참조 가능성이 다른 블록보다 크다면 캐쉬에 유지될 수 있어야 한다. 셋째, 충분히 큰 과거참조거리로 인해 시간적 지역성(temporal locality)의 영향력이 크지 않은 블록은 참조빈도에 기반하여 블록 교체를 결정한다. 넷째, 블록 교체의 시간복잡도(time complexity)가 LRU와 비슷해야 한다. 다섯째, 저장공간 오버헤드(space overhead)가 크지 않아야 한다. 본 논문은 위의 조건을 대체로 충족하는 *RCB* (*Reference Characteristic Based*) 교체 정책을 제안한다.

본 논문의 구성은 다음과 같다. 2절에서는 LRU의 한계점을 보완하기 위해 등장한 대표적인 캐쉬 교체 정책들에 대해서 기술하고 3절에서는 RCB 정책을 제안하는 동기가 된, 블록 참조 특징의 분석 결과를 제시한다. 이 결과를 바탕으로, 4절에서 RCB 정책의 알고리즘을 기술하며, 5절에서는 모의실험을 통해 RCB의 성능을 대표적인 캐쉬 교체 정책들과 비교, 평가한다. 마지막으로 6절에서 결론을 내리고 향후 연구 과제를 논의한다.

2. 관련 연구

LRU의 단점을 극복하기 위한 연구가 데이타베이스를 비롯한 여러 분야에서 진행되어 왔다. 각각의 연구들은 LRU에 비해 시간복잡도 혹은 저장공간 오버헤드가 있지만, 대체로 LRU보다 좋은 성능을 보인다. LRU-K[1]와 2Q[2]는 인기 있는 블록과 인기 없는 블록을 구별하여 인기 없는 블록에게 낮은 우선순위를 부여한다. LRU-K는 현재로부터 K번째 참조까지의 과거참조거리에 의해 블록 교체를 결정한다. LRU-2를 운용한다면, 현재로부터 2번째 참조까지의 과거참조거리에 의해 블록의 우선순위가 결정된다. 따라서 처음 참조된 블록은 2번째 참조까지의 과거참조거리가 무한대가 되므로, 낮은 우선순위를 받게 된다. LRU-K는 인기 없는 블록을 캐쉬에서 빨리 제거함으로써 LRU보다 좋은 성능을 보이지만, 시간복잡도가 $O(log_2N)$으로 LRU의 $O(1)$보다 큰 단점이 있다. 2Q는 LRU-K의 장점을 취하고 단점인 $O(log_2N)$의 시간복잡도를 해결하기 위해 제안되었다. 캐쉬를 2개의 크기가 다른 세그먼트로 분할하여 인기 없는 블록은 크기가 작은 세그먼트에 저장함으로써 캐쉬에서 빨리 교체한다. 두 세그먼트는 리스트로 운용되므로 2Q의 시간복잡도는 $O(1)$이다.

LRFU[3]는 최근도와 더불어 참조빈도도 블록의 재참조 가능성에 영향을 미친다는 사실에 주목하고 최근도와 참조빈도를 결합한 새로운 우선순위 계산법으로

1) 3절의 표 1, 2 참조
2) 가령 LFU의 경우에는 캐쉬에 유지될 수 있는 블록의 과거 참조 거리에 제한이 없다.
3) 3절의 그림 1 참조

블록 교체를 결정한다. 먼저 참조의 최근도를 고려하여 각 참조의 가치를 $\left(\frac{1}{2}\right)^{\lambda x}$ (x는 현재 시점으로부터 해당 참조 시점까지의 거리, λ는 상수) 식을 이용해 구한 다음, 계산된 값을 모두 더해 블록의 가치를 평가한다. LRFU는 블록 교체를 결정할 때 참조빈도와 최근도를 함께 고려한 장점이 있지만 성능을 최대로 하는 λ값을 결정하는 것이 쉽지 않고, λ값에 따라 $O(1)$과 $O(\log_2 N)$ 사이의 시간 복잡도를 갖는 단점이 있다.

EELRU[4]는 캐쉬 크기보다 큰 참조간 간격을 갖는 반복 참조 패턴에서 빈번한 캐쉬적중실패를 낳을 가능성이 큰 LRU의 단점을 극복하기 위해 제안되었다. EELRU는 일반적인 참조 패턴에서는 LRU로 동작하고, 캐쉬 크기보다 큰 참조간 간격을 갖는 반복 참조 패턴이 발견되면, MRU 방식으로 동작한다. 따라서 캐쉬 크기보다 큰 참조간 간격을 갖는 블록도 캐쉬에 머물 수 있다. EELRU는 가상 메모리 (virtual memory)의 교체 정책이다.

FBR[5]은 LFU와 마찬가지로 참조빈도에 따라 블록의 우선순위를 평가한다. 다른 점은 FBR은 *연관참조 (correlated reference)*는 참조빈도에 포함시키지 않는다는 점이다. 가령, A블록이 시간 $t1$과 $t2$에서 참조되었을 때, 만약 두 참조간 간격($t2 - t1$)이 시스템에서 지정한 연관참조거리 (correlated distance) 보다 작으면 두 참조는 서로 연관되었다 라고 판단되며, 연관참조의 경우에는 참조빈도를 증가시키지 않는다. 위의 예에서, A 블록의 참조빈도가 LFU에서는 2가 되지만, FBR에서는 1이 된다. [1]은 데이타베이스에서 연관참조가 발생할 수 있는 네 가지 상황을 제시하였으며 연관참조 개념은 LRU-K, 2Q, LRFU, RCB 정책 모두에서 채택된다.

3. 트레이스 분석

3.1 참조빈도에 따른 블록의 분류

표 1은 DB2트레이스[4]를 대상으로, 표 2는 OLTP 트레이스[5]를 대상으로 전체 블록을 참조빈도에 따라 분류한 표다. DB2의 경우 51%의 블록이 전체 시간동안 1번 참조되었으며 2번 참조된 블록도 약 27%에 달한다. OLTP의 경우에도 1번 참조된 블록이 전체의 약 46%이고 2번 참조된 블록은 전체의 24%이다. 두 트레이스 모두 전체 블록의 70% 정도가 전체 시간 동안 한두 번 참조됨을 보여 준다. 즉, 참조빈도가 극히 낮은 블록이 상당한 비율을 차지한다.

4) IBM에서 개발한 DB2 데이타베이스의 트레이스, 자세한 내용은 5절 참조.

5) CODASYL 데이타베이스의 트레이스, 자세한 내용은 5절 참조.

표 1 참조빈도에 의한 블록 분류(DB2)

참조빈도	비율 (%)
1	50.97
2	26.57
3 - 4	9.78
5 - 9	5.40
10 - 99	6.73
100 - 999	0.48
1000 - 9999	0.06
10000 이상	0.00
계	100

표 2 참조빈도에 의한 블록 분류(OLTP)

참조빈도	비율 (%)
1	45.98
2	23.61
3 - 4	14.67
5 - 9	9.49
10 - 99	5.79
100 - 999	0.43
1000 - 9999	0.03
10000 이상	0.00
계	100

3.2 최근도와 재참조 가능성과의 상관관계

시간적 지역성(temporal locality)의 존재로 인해 최근도는 블록의 재참조 가능성과 상관관계를 갖는다. 본 연구는 최근도와 재참조 가능성의 상관관계를 알아보기 위

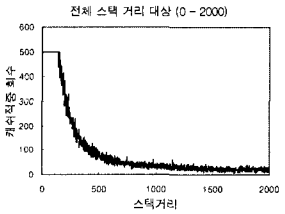

그림 1 스택거리별 캐쉬적중 회수(DB2), 스택거리 (0~2000)

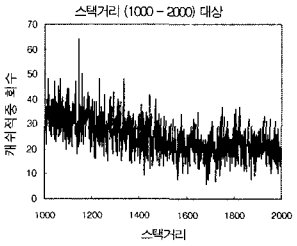

그림 2 스택거리별 캐쉬적중 회수(DB2), 스택거리 (1000~2000)

해 DB2 트레이스를 가지고 LRU를 운용하며 각 LRU 스택거리에서 캐쉬적중이 일어난 회수를 계수하였다. 캐쉬는 2000개의 블록을 저장할 수 있는 크기이다. 그림 1은 모든 스택거리를 대상으로 캐쉬적중 회수를 표시하였고, 그림 2는 스택거리(1000 - 2000)을 대상으로 적중 회수를 표시하였다. 그림을 보면, 스택의 앞부분에서는 캐쉬적중이 매우 빈번하게 일어나지만, 스택거리가 증가함에 따라 적중 회수가 급격히 감소하며(그림 1), 일정한 스택거리(약 1600 정도)를 넘어서면 스택거리에 관계없이 적중 회수가 거의 무작위한 형태로 분포함을 보여준다(그림 2). 이 결과를 통해, 스택거리가 작을 때는 최근도와 재참조 가능성의 상관관계가 크지만, 스택거리가 충분히 길어지면 거의 무관해 짐을 알 수 있다.

4. RCB 캐쉬 교체 정책

4.1 최근도와 참조빈도에 따른 블록의 분류

3절의 트레이스 분석 결과를 볼 때, 최근도와 참조빈도를 기준으로 서로 다른 특징을 갖는 4개의 블록 그룹이 캐쉬에 존재한다. 첫 번째 그룹은 참조빈도가 크고 비교적 최근에 참조되어 최근도가 재참조 가능성에 영향을 미치는 그룹이다. 최근도가 재참조 가능성에 영향을 미치므로 LRU가 적합한 교체 정책이다. 두 번째 그룹은 참조빈도가 극히 낮지만, 비교적 최근에 참조되어 최근도가 재참조 가능성에 영향을 미치는 그룹이다. 이 그룹은 첫째 그룹처럼 LRU가 적합한 교체 정책이며, 첫째 그룹에 비해 참조빈도가 낮아 재참조 가능성이 낮기 때문에 적절한 페널티를 받아야 한다. 세 번째 그룹은 참조빈도가 크지만, 참조된 지 오랜 시간이 지나서 최근도가 재참조 가능성에 거의 영향력을 갖지 않는 그룹이다. 이 경우에는 블록의 참조빈도가 재참조 가능성과 상관 관계를 가질 가능성이 크기 때문에 참조빈도에 기반한 교체 정책이 운용되어야 한다. 네 번째 그룹은 참조빈도가 극히 낮고, 참조된 지 오랜 시간이 지나서 최근도가 재참조 가능성에 거의 영향을 미치지 않는 그룹이다. 참조빈도에 기반한 교체 정책이 운용되어야 하며, 재참조 가능성이 낮으므로 페널티를 받는다.

RCB는 각 그룹의 참조 특징에 적합한 우선 순위 평가 방법을 운용하기 위해 캐쉬를 3개의 세그먼트로 분할한다. 첫째 세그먼트인 ***lru_main***은 첫 번째 그룹을 저장하며 LRU를 운용한다. 둘째 세그먼트 ***lru_cold***는 두 번째 그룹을 저장하며 역시 LRU를 운용한다. 두 번째 그룹에게 페널티를 부여하기 위해 lru_cold는 lru_main보다 크기가 작다. 셋째 세그먼트 ***popularity_lists***는 세 번째와 네 번째 그룹을 저장하며 참조빈도에 기반한 정책을 운용한다. Popularity_lists가 참조빈도에 기반한 정책을 운용하므로 참조빈도가 낮은 네 번째 그룹은 상대적인 페널티를 받는다.

4.2 RCB 알고리즘 기술

RCB의 구체적인 알고리즘은 두 단계를 통해 기술된다. 첫 단계에서는 블록참조가 발생했을 때, 각 세그먼트 내에서와 세그먼트들 사이에 어떠한 동작이 일어나는 지를 기술한다. 두 번째 단계에서는 popularity_lists에서 운용할 참조빈도 기반 정책의 구체적인 모습을 정의한다.

먼저, 요청된 블록이 캐쉬에서 발견되었다면 세가지 가능한 경우가 있다. 첫째 블록이 lru_main에서 발견된 경우, 둘째 블록이 lru_cold에서 발견된 경우, 마지막으로 블록이 popularity_lists에서 발견된 경우이다. lru_main이나 lru_cold에서 발견되었다면 이들이 LRU를 운용하므로 참조된 블록은 각 세그먼트에서 가장 높은 우선 순위를 받게 된다. 블록이 popularity_lists에서 발견되면 블록의 스택거리가 1이 되었기 때문에 블록은 popularity_lists에서 제거되어 lru_main으로 삽입된다. lru_main에서 스택거리가 가장 길었던 블록은 lru_main에서 제거되어 popularity_lists로 삽입된다. 참조된 블록이 각 세그먼트에 삽입될 때 삽입되는 위치는 세그먼트의 우선 순위 평가 방법에 의해 결정된다. 즉, LRU를 운용한다면 최근도에 의해, 참조빈도 기반 정책을 운용한다면 참조빈도에 의해 삽입되는 위치가 결정된다.

블록이 캐쉬에서 발견되지 않는다면, 블록은 디스크로부터 캐쉬로 로드되어야 한다. 만약 현재 캐쉬가 포화상태라면, 요청된 블록을 저장할 공간확보를 위해 popularity_lists에서 우선순위가 가장 낮은 블록을 캐쉬에서 교체한다. 새로 요청된 블록은 스택거리가 1이므로, 재참조 가능성에 따라 lru_main이나 lru_cold에 삽입된다. 이 때 lru_main이나 lru_cold에서 스택거리가 가장 길었던 블록은 교체되어 popularity_lists로 삽입된다. 블록의 재참조 가능성은 연구[2]에서 사용된 방법에 의해 판단된다. RCB는 과거에 캐쉬에서 교체된 블록들의 블록번호를 ***history_list***를 통해 기억하고, 새로 참조된 블록이 history_list에서 발견되면, 즉 블록이 예전에도 참조된 블록이면 이 블록의 재참조 가능성은 높다고 판단하며, 그렇지 않으면 재참조 가능성이 낮다고 판단한다. 블록번호를 저장하는 것은 2~3바이트를 요구하므로, 그리 큰 저장공간 오버헤드가 아니다[2].

한편, 참조빈도에 기반한 교체 정책을 사용할 때는 두 가지의 문제가 따른다. 첫 번째 문제는 시간복잡도 오버헤드다. 일반적으로 참조빈도에 기반한 교체 정책은 블록

을 참조빈도에 따라 정렬하기 위해 힙 자료구조(heap data structure)를 필요로 하며, 힙은 $O(log_2N)$과 같은 로그 스케일의 시간복잡도를 갖는다. RCB는 popularity_lists의 자료구조로 힙 대신에 다수의 연결 리스트(linked list)로 이루어진 리스트 집합을 사용함으로써 이 문제를 해결하였다.[6] popularity_lists는 서로 다른 ***인기도(popularity)*** 값을 갖는 리스트들의 집합으로 구성된다. 인기도는 다음과 같이 정의된다.

정의 1. *인기도=[log₂(참조빈도)](1<=참조빈도<=32)*
= 5 (참조빈도>32)

인기도의 최대값이 5이고 최소값이 0[7]이므로 popularity_lists는 0부터 5까지의 값을 갖는, 최대 6개의 리스트로 구성된다. 각각의 리스트는 LRU를 운용한다. 블록은 참조빈도에 따라 해당하는 리스트에 삽입되며 교체될 블록은 항상 인기도 값 0을 갖는 리스트에서 선택된다. 만약 인기도 값 0을 갖는 리스트에 두 개 이상의 블록이 존재한다면, 가장 오래 참조되지 않은 블록이 교체 대상으로 선택된다. 각 리스트가 LRU를 운용하므로 교체 작업의 시간복잡도는 $O(1)$이다.

참조빈도에 기반한 교체 정책의 두 번째 문제는 캐쉬 오염(cache pollution)이다. 이 문제를 해결하기 위해 RCB는 일정한 주기마다 popularity_lists 블록들의 인기도를 1만큼 감소시키는 에이징(aging)을 실시한다. 따라서 과거에 참조빈도가 잦았던 블록이라도 오랜 시간 참조되지 않으면 결국엔 인기도 값이 0이 되어 캐쉬에서 교체된다. 에이징 작업은 인기도 값이 큰 리스트의 블록들을 인기도 값이 작은 리스트에 단순히 덧붙이는 것을 의미하기 때문에 시간복잡도는 $O(1)$이다. 가령, 인기도 3을 갖는 리스트의 모든 블록이 인기도 2를 갖는 리스트에 덧붙여진다. 인기도 0을 갖는 리스트의 블록들은 더 낮은 값을 갖는 리스트가 존재하지 않으므로 에이징이 되지 않는다. 주의해야 할 점은 RCB는 단지 인기도만을 에이징하며, 참조빈도는 그대로 유지한다는 사실이다. 참조빈도를 에이징한다면 에이징 작업의 시간복잡도가 $O(1)$보다 크게 될 것이다.

RCB의 알고리즘은 다음과 같이 기술된다.

```
RCB 알고리즘 기술
If (the requested block is found in cache) {   /* cache hit */
    If (the block is in lru_main) {
        frequency++
        move the block to the tail of lru_main.
    }
    Else if (the block is in lru_cold) {
        frequency++
        move the block to the tail of lru_cold.
    }
    Else if (the block is in popularity_lists) {
        frequency++
        remove the block from the popularity_lists.
        insert the block into the tail of lru_main.
        remove the head block of lru_main and insert it into
        popularity_lists according to popularity value.
    }
}
Else {                          /* cache miss */
    evict the head block from the lowest popularity value list
    If (the requested block is found in history list)  /*the block
      is apt to be popular */
    {
        insert the block into the tail of lru_main
        remove the head block of lru_main and insert it into
          the popularity_lists according to popularity value.
    }
    Else { /* the block is apt to be non-popular */
        insert the block into the tail of lru_cold.
        remove the head block of lru_cold and insert it into
          the popularity_lists according to popularity value.
    }
}
```

5. 성능 평가

이 절에서는 실제 데이타베이스 트레이스를 가지고 모의실험을 수행하여 각 정책의 성능을 비교한다. 공정하고 또한 검증된 비교를 위해 [1,2,3] 등에서 사용한 트레이스인 DB2와 OLTP 트레이스를 가지고 모의실험을 수행했다. DB2는 IBM에서 개발한 상업용 데이타베이스 프로그램인 DB2의 트레이스 파일로서 총 75514개의 블록에 대해 총 500000번의 블록 참조를 수행한다. OLTP는 CODASYL 데이타베이스의 트레이스 파일로서 총 186880개의 블록에 대해 총 914145번의 블록 참조를 수행한다. 성능 비교는 대표적인 캐쉬 교체 정책인 LRU와, 90년대 들어 등장한 LRU-K, 2Q, LRFU등을 대상으로 한다. LFU는 성능이 다른 정책에 비해 많이 떨어지므로 제외하였고 연구[5]에서 제안된 FBR은 LRFU에 포함되는 정책이므로 비교에서 제외하였다[3]. DB2는 캐쉬를 블록을 1000개 저장할 수 있는 크기(1000*블록크기)부터 10000개 저장할 수 있는 크기(10000*블록크기)까지 1000개씩 증가시켰고, OLTP는 캐쉬를 블록을 1000개 저장할 수 있는 크기(1000*블록크기)부터 15000개 저장할 수 있는 크기(15000*블록크기)까지 2000개씩 증가시켰다.

6) 리스트 집합은 연구[6] 등에서도 비슷한 방법으로 이용되었다.
7) 인기도는 캐쉬(popularity_lists)에 존재하는 블록에 대해 계산되므로 참조빈도의 최소값은 1이고 인기도의 최소값은 0이다.

성능 비교를 할 때 주의해야 할 점은 각 정책이 성능에 영향을 미치는 튜닝요소(tuning factor)를 가지고 있다는 사실이다. LRU는 이러한 튜닝요소를 포함하고 있지 않지만, RCB를 비롯한 다른 정책들은 모두 성능에 영향을 미치는 튜닝요소를 갖는다. 즉 튜닝요소의 값을 어떻게 정하느냐에 따라 성능이 다르게 나타난다. 최대 성능을 내는 튜닝요소의 값은 일반적으로 트레이스의 특성과 캐쉬 크기에 따라 조금씩 달라지기 때문에, 모의실험 수행을 통해 우수하다고 결론 내릴 수 있는 정책은 튜닝요소의 값에 크게 영향을 받지 않으면서 다양한 트레이스와 캐쉬 크기에 대해 지속적으로 좋은 성능을 낼 수 있는 정책이어야 할 것이다. 이러한 점을 고려하여 본 연구는 모의실험을 두 가지 방법으로 수행한다. 먼저 각 정책의 실용적 성능 측정을 위해 튜닝요소를 모두 한가지 값으로 고정하여 모의실험을 수행한다. 이 때 튜닝요소의 값은 각 논문에서 제안하는 값을 사용한다. 그리고 각 정책이 최대로 낼 수 있는 성능의 측정을 위해 튜닝요소의 값을 변화시켜가며 모의실험을 수행한다. 이 때에도 튜닝요소의 값은 각각의 논문이 제안하는 값을 사용한다.

5.1 튜닝요소의 고정

2Q의 경우 [2]의 제안대로 Ain의 크기를 캐쉬크기의 20%, Aout의 크기는 캐쉬의 50%로 한다. LRU-K는 [1]의 제안대로 LRU-2를 사용하고 Ain의 크기는 캐쉬 크기의 30%로 한다. LRFU는 블록의 가치를 평가하는 함수인 $\left(\frac{1}{2}\right)^{\lambda r}$의 λ와 연관참조거리가 튜닝요소인데, [3]에서 특정한 값을 제안하고 있지 않기 때문에, 여러 값을 대입한 실험을 통해 평균적으로 좋은 성능을 보인 값을 사용했다. λ는 0.00003, 연관참조거리는 1000으로 하였다. RCB는 lru_cold와 history_list의 크기는 2Q와 마찬가지로 각각 캐쉬크기의 20%와 50%로 하였고 popularity_lists의 크기는 캐쉬의 15%, 연관참조거리는 캐쉬의 40%로 하였다.

튜닝요소의 값을 위와 값이 고정하고 캐쉬 크기를 변화시켜 가며 모의실험을 수행한 결과가 그림 3과 4에 나와 있다. 그림 3은 DB2를 수행한 결과이며 그림 4는 OLTP를 수행한 결과이다. LRU의 캐쉬적중실패 회수를 100으로 하였을 때, 각 정책의 캐쉬적중실패 회수의 비례값을 계산하여 성능 비교 기준으로 이용하였다. 가령 LRU의 적중실패 회수가 200이고 RCB의 적중실패 회수가 120이라면 비례값은 60이 된다.

RCB는 DB2와 OLTP 모두, 전 캐쉬 크기에서 다른 정책들보다 우수한 성능을 보인다. 특히 LRU에 비해서 캐쉬적중실패 회수를, DB2에서는 약 5.5 ~ 10.1 % 까지, OLTP에서는 약 6.5 ~ 12.7% 까지 줄일 수 있었다. 다른 정책들의 경우에는 대체로 LRFU가 우수한 성능을 보이지만, 캐쉬 크기가 작을 때, 특히 OLTP 트레이스에서 2Q가 LRFU보다 나은 성능을 보인다. 즉, 다른 정책들은 캐쉬 크기와 트레이스 성격에 따라 성능 정도가 약간씩 다르게 나타났다. 이에 반해 RCB는 두 트레이스 모두, 전 캐쉬 크기에서 좋은 성능을 나타낸다.

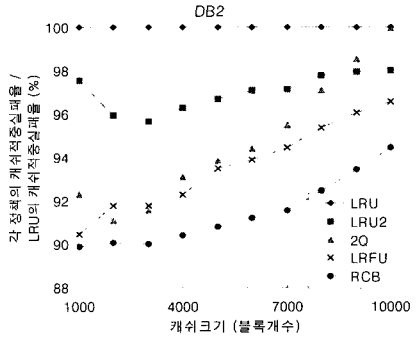

그림 3 DB2를 가지고 튜닝요소를 고정시켜 수행한 캐쉬적중실패 비교 실험

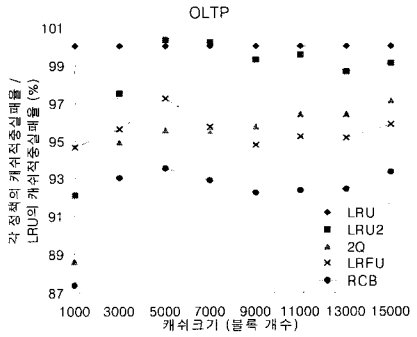

그림 4 OLTP를 가지고 튜닝요소를 고정시켜 수행한 캐쉬적중실패 비교 실험

5.2 최대 성능 측정(튜닝요소의 변화)

이제는 각 정책의 튜닝요소 값들을 변화시켜 가며 최대 성능을 측정한다. 2Q와 LRU-2는 [2]의 제안대로 Ain의 크기를 각각 캐쉬 크기의 20%와 30%로 하였고 LRFU의 경우는 λ의 값을 0.00001부터 0.009까지[8] 변

8) 0.00001, 0.00003, ..., 0.00009, 0.0001, 0.0003, ..., 0.007, 0.009 의 값들을 대입했다.

화시키고 연관참조거리는 500부터 3000까지 500씩 증가시켜 가며 실험 하였다. 여러 결과들 중 가장 좋은 성능을 보이는 결과를 선택한다. RCB는 history_list와 연관참조거리는 각각 캐쉬 크기의 50%와 40%로 고정한 채, lru_cold는 캐쉬의 10 ~ 30%까지 5%씩 변화시켰고 popularity_lists는 캐쉬의 15~35%까지 5%씩 변화시켰다. 역시 가장 좋은 성능을 보이는 결과를 선택한다. RCB의 연관참조거리를 캐쉬의 40%로 고정한 이유는 RCB의 성능은 연관참조거리의 값에 그다지 큰 영향을 받지 않았기 때문이다.

그림 5와 6에 DB2와 OLTP의 실험 결과가 나와 있다. 그림 5는 DB2의 결과이며 그림 6은 OLTP의 결과이다. 튜닝요소를 고정한 실험과 마찬가지로 RCB는 두 트레이스 모두, 전 캐쉬 크기에서 다른 정책들보다 뛰어난 성능을 보인다. 특히, LRU에 비해서는, DB2의 경우에 캐쉬적중실패를 약 5.8~10.7%, OLTP의 경우에는 약 6.5~12.7%까지 줄인다. 다른 캐쉬 교체 정책들의 경우에는 첫번째 실험과 비슷하게 LRFU가 대체적으로 가장 좋은 성능을 보이고 캐쉬 크기가 작을 때는 2Q가 LRFU보다 좋은 성능을 보일 때도 있다. 이는 2Q가 캐쉬크기가 작을 때, 재참조 가능성이 낮은 블록을 빨리 교체함으로써 캐쉬를 효율적으로 사용할 수 있기 때문인 것으로 판단된다.

이상의 두 실험을 통해, 데이타베이스와 같이 시간적 지역성과 비균등 참조 패턴을 함께 가지고 있는 워크로드에서 RCB가 기존의 정책들보다 우수한 교체 정책이라고 결론 내릴 수 있다. RCB의 시간복잡도는 *O(1)*로서 LRU와 동일하며, 저장공간 오버헤드는 크지 않고, 전 캐쉬 크기에서 캐쉬적중실패 회수를 감소시키기 때문이다.

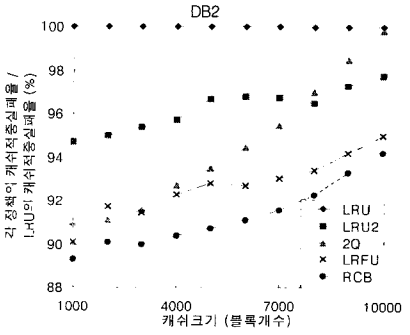

그림 5 DB2를 가지고 튜닝요소를 변화시켜 수행한 캐쉬적중실패 비교 실험

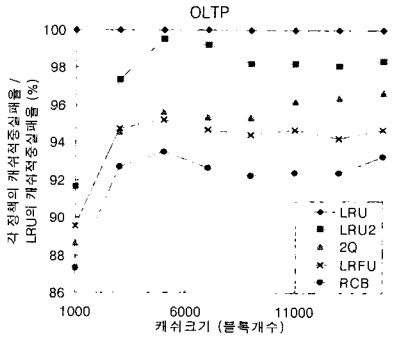

그림 6 OLTP를 가지고 튜닝요소를 변화시켜 수행한 캐쉬적중실패 비교 실험

6. 결 론

본 연구에서는 데이타베이스 트레이스 분석을 통해 다음과 같은 블록 참조 특징을 추출했다. 첫째, 전체 블록의 70% 이상이 전체 시간동안 한두 번 참조된다. 둘째, 블록의 스택거리가 짧을 때는 최근도가 재참조 가능성과 강한 상관관계를 갖지만, 스택거리가 길어짐에 따라 상관 관계는 급격히 감소하여 결국엔 무관해진다. 이러한 관찰에 기반하여 본 논문은 전체 블록을 서로 다른 인기도와 최근도를 갖는 네 개의 그룹으로 분류하고, 각 그룹의 특성에 적합한 우선순위 평가 방법을 운용하는 교체 정책, RCB를 제안했다. 블록의 그룹화와 그룹의 블록 참조 특징에 맞는 우선순위 평가 방법의 운용을 통해, RCB는 재참조 가능성이 낮은 블록은 다른 블록들보다 빨리 캐쉬에서 교체하며, 스택거리가 길어서 최근도보다 참조빈도가 재참조 가능성에 더 영향을 미치는 블록에 대해서는 참조빈도에서 유추한 개념인 인기도에 따라 블록의 우선순위를 부여했다.

트레이스를 통한 모의실험 결과, RCB는 튜닝요소의 값을 고정시킨 실험과 변화시킨 실험 모두에서 다른 정책들보다 우수한 성능을 보였다. 즉, 캐쉬적중실패 회수를 감소시켰다. RCB의 시간복잡도는 *O(1)*로서 LRU, 2Q 등과 동일하며, 캐쉬 크기를 N이라 할 때 시간복잡도가 $O(log_2N)$인 LFU와 LRU-K, 그리고 *O(1)* 부터 $O(log_2N)$ 사이의 값을 갖는 LRFU보다 우수하다. 결국 RCB는 데이타베이스와 같이 시간적 지역성과 비균등 참조 패턴을 함께 갖고 있는 워크로드에서 기존의 제안된 정책들에 대해 성능 개선 효과가 있는 교체 정책이라고 결론내릴 수 있다.

참 고 문 헌

[1] E. J. O'Neil, P.E. O'Neil, G. Weikum. "The LRU-K Page Replacement Algorithm For Database Disk Buffering," Proc. Of the 1993 ACM SIGMOD Conference, pp 297-306, 1993.

[2] T. Johnson, D. Shasha.. "2Q: A Low Overhead High Performance Buffer Management Replacement Algorithm," Proc. Of the 20th International Conference on Very Large Data Bases, pp 439-450, 1994.

[3] Donghee Lee, Jongmoo Choi, Jong-Hun Kim, Sam H. Noh, Sang Lyul Min, Yookun Cho, Chong-Sang Kim. "On the Existence of a Spectrum of Policies that Subsumes the Least Recently Used (LRU) and Least Frequently Used (LFU) Policies," Proc. Of the 1999 ACM SIGMETRICS International Conference on Measurement and Modeling of Computer Systems, pp 134-143, May 1999.

[4] Yannis Smaragdakis, Scott Kaplan, and Paul Wilson. "EELRU: Simple and Effective Adaptive Page Replacement," Proc. Of the 1999 ACM SIGMETRICS International Conference on Measurement and Modeling of Computer Systems, pp 134-143, May 1999.

[5] J. T. Robinson, N. V. Devarakonda. "Data Cache Management Using Frequency-Based Replacement," Proc. Of the 1990 ACM SIGMETRICS Conference, pp 134-142, 1990.

[6] Stephen Williams, Marc Abrams, Charles Standridge, Ghaleb Abdulla, and Edward Fox. "Removal Policies in Network Caches for World-Wide Web Documents," Procedings of the ACM SIGCOMM'96 conference, 1996.

신 일 훈

서울대학교 전산과학과 학사. 서울대학교 전산과학과 석사. 현재 서울대학교 전기컴퓨터공학부 박사과정. 관심분야는 파일 시스템, 멀티미디어 시스템, 저전력 시스템

고 건

서울대학교 응용물리학과 학사. 버지니아 대학교 전산학 석사. 버지니아 대학교 전산학 학사. 현재 서울대학교 전기컴퓨터 공학부 교수. 관심분야는 운영체제, 멀티미디어 시스템, 클러스터 시스템, 웹캐쉬, 저전력 시스템