순차 참조와 순환 참조들을 고려한 버퍼 캐쉬 관리 기법

(An Effective Buffer Management Scheme for Sequential and Looping References)

김 종 민 † 최 종 무 † 김 제 성 † 이 동 희 ††
(Jong Min Kim) (Jongmoo Choi) (Jesung Kim) (Donghee Lee)
노 삼 혁 ††† 민 상 렬 †††† 조 유 근 †††† 김 종 상 ††††
(Sam H. Noh) (Sang Lyul Min) (Yookun Cho) (Chong Sang Kim)

요 약 최근 버퍼 캐쉬의 성능을 향상시키기 위한 많은 블록 교체 기법들이 제안되었으며 이 중에서 작업 집합 (working set) 변화에 잘 적응하고 구현이 용이한 Least Recently Used (LRU) 블록 교체 기법이 널리 사용되고 있다. 그러나 LRU 블록 교체 기법은 블록들이 규칙적인 참조 패턴을 보이면서 순차 참조되거나 순환 참조될 때 이 규칙성을 적절히 이용하지 못해 성능이 저하되는 문제점을 가진다. 본 논문에서는 다중 응용 트레이스를 이용하여 LRU 블록 교체 기법의 문제점을 관찰하고, 이 문제점을 해결하는 통합된 형태의 효율적인 버퍼 관리 (Unified Buffer Management, 이하 UBM) 기법을 제안한다. UBM 기법은 순차 참조 및 순환 참조를 자동 검출하여 분리된 공간에 저장하고 이들 참조에 적합한 블록 교체 기법으로 이 공간을 관리한다. 또한 순차 참조와 순환 참조를 위한 공간과 나머지 참조를 위한 공간의 비율을 최적으로 할당하기 위해 온라인에서 수집된 정보를 이용하여 계산된 단위 공간 증가당 예상 버퍼 적중 증가율을 이용한다. 다중 응용 트레이스 기반 시뮬레이션 실험에서 UBM 기법의 버퍼 적중률은 LRU 블록 교체 기법에 비해 평균 12%, 최대 28%까지 향상된 결과를 보였다.

Abstract The Least Recently Used (LRU) block replacement scheme is still widely used due to its simplicity. While simple, it still adapts well to the changes of the working set, and has been shown to be efficient when recently referenced blocks are likely to be re-referenced in the near future. The main drawback of the LRU scheme, however, is that it exhibits performance degradations because it does not make use of reference regularities such as sequential and looping references. In this paper, we present the Unified Buffer Management (UBM) scheme that exploits these regularities and yet, is simple to deploy. The UBM scheme automatically detects sequential and looping references and stores the detected blocks in separate partitions of the buffer cache. These partitions are managed by appropriate replacement schemes based on their detected patterns. The allocation problem among the divided partitions is also tackled with the use of the notion of marginal gains. The performance gains obtained through the use of this scheme are substantial. Trace-driven simulation experiments show that the hit ratios improve by as much as 28% (with an average of 12%) compared to the LRU scheme for the traces we considered.

· This work was supported in part by the Ministry of Education under the Brain Korea 21 program and by the Ministry of Science and Technology under the National Research Laboratory program.

† 비 회 원 : 서울대학교 컴퓨터공학부
jmkim@lab.snu.ac.kr
choijm@ssrnet.snu.ac.kr
jskim@archi.snu.ac.kr
†† 비 회 원 : 제주대학교 통신컴퓨터공학부 교수
dhlee@cheju.cheju.ac.kr
††† 종신회원 : 홍익대학교 컴퓨터공학과 교수
noh@cs.hongik.ac.kr
†††† 종신회원 : 서울대학교 컴퓨터공학부 교수
symin@dandelion.snu.ac.kr
cho@comp.snu.ac.kr
cskim@sparc.snu.ac.kr
논문접수 : 2000년 5월 2일
심사완료 : 2000년 11월 23일

1. 서 론

처리기에 비해 상대적으로 느린 디스크는 오늘날 전체 시스템의 성능 향상에 병목 현상을유발하는 한 요소가 되고 있다. 이 문제를 해결하기 위해 대부분의 운영 체제들은 참조되는 디스크 블록들 중 일부를 주기억장치에 유지하여 디스크 입출력 횟수와 평균 디스크 접근 시간을 줄이는 버퍼 캐쉬 기법을 사용하고 있다. 버퍼 캐쉬는 재 참조 가능성이 높은 블록들을 선별하는 블록 교체 기법에 의해 관리되며, 효과적인 블록 교체 기법에 의한 효율적인 버퍼 캐쉬 관리는 시스템 성능 향상을 위해 중요하다.

최근 버퍼 캐쉬의 성능을 향상시키기 위해 많은 블록 교체 기법들이 제안되었다 [1, 2, 3, 4, 5, 6, 7]. 이 중에서 Least Recently Used (LRU) 블록 교체 기법은 구현이 용이하기 때문에 많은 시스템에서 널리 사용되고 있다. 참조 최근성 (recency)을 기준으로 블록을 교체하는 LRU 블록 교체 기법은 작업 집합 (working set) 변화에 잘 적응하며, 블록들이 시간 지역성 (temporal locality)을 보이며 참조될 때 좋은 성능을 보인다 [8]. 그러나 LRU 블록 교체 기법은 블록들이 규칙적인 참조 패턴을 보이면서 순차 참조되거나 순환 참조될 때 이 규칙성을 적절히 이용하지 못해 성능이 저하되는 문제점을 가진다. LRU 블록 교체 기법의 이 문제점은 본 논문의 4.2.2 절에서 다중 응용 트레이스를 이용한 실험을 통해 구체적으로 예시한다.

이 관찰로부터 본 논문은 순차 참조와 순환 참조를 고려한 통합된 형태의 효율적인 버퍼 관리 (Unified Buffer Management, 이하 UBM) 기법을 제안한다. UBM 기법은 순차 참조 및 순환 참조를 자동 검출하여 분리된 공간에 저장하고 순환 주기를 기반으로 한 블록 교체 기법으로 이 공간을 관리한다. 반면 여기에 속하지 않는 블록들은 LRU 블록 교체 기법으로 관리되는 다른 공간에 저장된다. 또한 이 두 공간의 크기의 비율은 온라인에서 수집된 정보를 이용하여 계산된 단위 공간 증가당 예상 버퍼 적중 증가율에 기반하여 동적으로 조절된다.

본 논문에서는 UBM 기법의 성능을 평가하기 위해 FreeBSD 운영체제에서 실제 응용들을 수행시켜 수집한 트레이스를 사용하여 시뮬레이션을 수행하였다. 실험결과 UBM 기법이 1) 순차 참조와 순환 참조들 대부분을 자동 검출하고, 2) OPT 블록 교체 기법과 유사하게 순차 참조와 순환 참조들을 처리하며, 3) 버퍼 적중률을 LRU 블록 교체 기법에 비해 평균 12%, 최대 28%까지 향상시킴을 확인하였다.

본 논문은 모두 5 장으로 구성되어 있다. 다음 장에서는 관련 연구들을 살펴본다. 3 장에서는 UBM 기법의 세부 사항과 동작을 설명한다. 4 장은 UBM 기법의 성능 평가 결과를 기술한다. 마지막으로 5 장에서는 본 논문의 결론과 앞으로의 연구 방향을 제시한다.

2. 관련 연구

최근 참조 가능성이 높은 페이지/블록들을 주기억장치에 유지하기 위한 다양한 페이지/블록 교체 기법들이 제안되었다 [1, 2, 3, 4, 5, 6, 7, 9, 10]. 본 장에서는 이 교체 기법들 중 규칙적인 참조 패턴을 보이는 순차 참조와 순환 참조를 고려한 교체 기법들로 Johnson과 Shasha에 의해 제안된 2Q [3], Glass와 Cao에 의해 제안된 SEQ [9], Smaragdakis 등에 의해 제안된 EELRU (Early Eviction LRU) [10] 교체 기법들을 차례로 살펴본다.

2Q 블록 교체 기법은 처음 참조된 블록을 짧은 기간만 유지하고 교체하기 위한 별도의 보조 버퍼 캐쉬인 'A1in' 큐를 이용하여 순차 참조 블록과 순환 주기가 긴 순환 참조 블록들을 우선적으로 교체한다. 반면에 재 참조 가능성이 높은 블록들을 선별하기 위해 'A1in' 큐에서 쫓겨난 블록들 중 이들의 주소 정보만을 일시적으로 유지하는 가상의 'A1out' 큐를 이용한다. 순환 주기가 짧은 순환 참조 블록들이 재 참조시 이 블록의 주소 정보가 'A1out' 큐에 존재하는 경우 'A1in' 큐에 유지하지 않고 주 버퍼 캐쉬에 유지한다. 그러나 이 기법에서는 새로 참조된 블록이 주 버퍼 캐쉬에 유지될 때 언제나 부가적인 캐쉬 접근 실패 (cache miss)가 발생하며, 제어 변수인 'A1in' 큐와 'A1out' 큐의 크기를 오프라인에서 결정해 주어야 하는 단점이 있다.

SEQ 페이지 교체 기법은 연속된 가상 주소에서 발생하는 긴 순차 페이지 부재들을 검출하고, 이 페이지들에 대해 Most Recently Used (MRU) 교체 기법을 적용한다. 그러나 이 기법은 검출된 페이지들이 순차 참조되는 페이지들인지 순환 참조되는 페이지들인지 구분하지 않고 우선적 페이지 교체 대상으로 사용한다. EELRU 페이지 교체 기법은 온라인에서 수집된 참조 최근성 분포 (recency distribution)로부터 순환 참조 페이지들의 존재를 확인하고, 이 분포에 기반한 손익 계산을 통해 페이지 교체 위치를 변경한다. 그러나 이 기법은 순환 참조들의 순환 주기를 고려하지 않는다.

3. 통합 버퍼 관리 (UBM) 기법

본 장에서는 UBM 기법을 구성하는 다음 세 부분들을 차례로 기술한다.

- 블록 참조에서 순차 참조와 순환 참조를 자동 검출하여 분류하는 부분
- 분류된 블록들을 참조 패턴에 적합한 블록 교체 기법으로 관리하는 부분
- 분리된 공간들간에 버퍼를 최적으로 할당하는 부분

3.1 순차 참조와 순환 참조의 검출

UBM 기법은 일정 개수 이상의 연속된 블록 참조(consecutive block references)와 이러한 연속된 블록 참조의 반복 유무에 기반하여 순차 참조와 순환 참조들을 자동 검출하여 분류한다. 연속된 블록 참조는 화일이 순차적으로 접근될 때 발생되며, 화일별로 검출된다. 따라서 본 논문에서 블록 참조는 다음 세 가지로 분류된다.

1. 일정 개수[1] 이상의 연속된 블록 참조들이 한번만 발생되는 **순차 참조 (sequential references)**.
2. 순차 참조들이 규칙적인 간격으로 반복해서 발생되는 **순환 참조 (looping references)**.
3. 순차 참조와 순환 참조에 속하지 않는 **기타 참조 (other references)**.

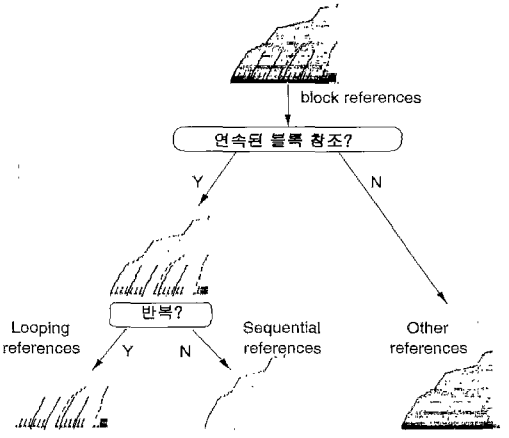


그림 1 통합 버퍼 관리 기법에서 참조 분류

그림 1은 UBM 기법이 블록 참조에서 순차 참조와 순환 참조, 기타 참조를 검출하여 분류하는 과정을 보인다.

순차 참조와 순환 참조들을 온라인에서 검출하기 위해 필요한 인자들은 그림 2에서 보인다. 일정 개수 이상

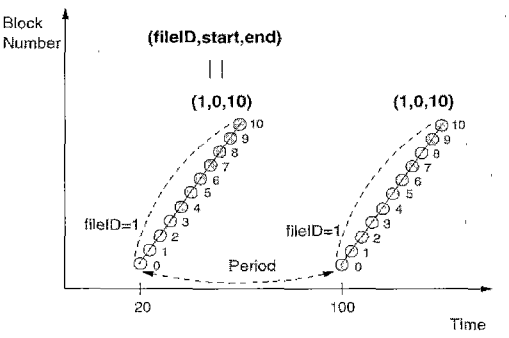


그림 2 순차 참조와 순환 참조를 검출하기 위해 필요한 인자

연속된 블록 참조가 발생하는 경우 화일식별자 (*fileID*), 시작 블록 번호 (*start*), 마지막 블록 번호 (*end*)를 이용하여 순차 참조를 표현한다. 그리고 이러한 순차 참조가 반복해서 참조되는 경우 순환 주기 (*period*)를 추가하여 순환 참조를 표현한다. 실제 시스템에서 순환 참조의 순환 주기는 시스템 부하와 프로세스 스케줄링 정책에 따라 변화할 수도 있다. 이 문제를 해결하기 위해 UBM 기법에서는 이전 순환 주기와 현 순환 주기의 지수 평균 (exponential average) 값을 사용한다.

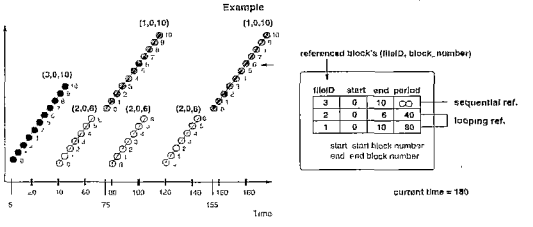


그림 3 순차 참조와 순환 참조의 검출 예

그림 3은 두 개의 순환 참조와 한 개의 순차 참조의 검출 예와 이러한 참조들에 대한 정보를 유지하는 검출 테이블을 보인다. 주 메모리상에 유지되는 제한된 크기의 검출 테이블은 현재까지 검출된 연속된 블록 참조들에 대한 정보들을 유지하며 매 블록 참조시 갱신된다.검출 테이블에 새로운 항을 추가시 가용 항이 존재하지 않는 경우에는 순차 참조와 순환 참조에 대한 정보를 저장하고 있지 않는 가장 오래된 기존 항을 하나 제거하여 사용한다. UNIX 화일 시스템에서 연속된 블록 참조는 동일 vnode 번호와 연속된 블록 번호[2]를 이용하

1) 순차 참조로 분류하기 위한 임계 값이며, 4 장에서 설명한다.

2) 여기서 블록 번호는 화일 내에서의 논리 블록 번호이다.

여 검출된다. 그림에서 (*fileID,start,end*)가 (3,0,10)인 연속된 블록 참조는 현재 시각까지 재 참조가 발생하지 않았기 때문에 순환 주기가 ∞인 순환 참조, 즉 순차 참조로 분류된다. (2,0,6)과 (1,0,10)인 순차 참조들은 각각 시각 120, 155에 재 참조되었기 때문에 순환 주기가 40, 80인 순환 참조들로 분류된다.

UBM 기법에서 순차 참조와 순환 참조를 자동 검출하는데 소요되는 시간은 시스템 성능 향상에 오히려 장애가 될 수 있다. UBM 기법의 동작을 분석한 결과 검출 시간의 대부분이 검출 테이블 내 해당 항목을 검색하는데 소요되는 것으로 나타났다. 본 논문에서는 이 시간을 줄이기 위해 vnode 번호를 이용한 해싱 함수를 사용하여 검출 테이블 내 항목을 검색하도록 하였다. 일반적으로 해싱 함수를 이용하는 경우 평균 검색 시간은 상수 시간으로 알려져 있으며 [11], 실험에서 해쉬 테이블 내 버켓(bucket)의 수가 51개 일 때 해당 항목을 발견하기 위한 평균 검사 횟수는 4.3번으로 측정되었다. 이러한 검출 비용은 오늘날 프로세서와 디스크의 속도 차이를 감안할 때 UBM 기법의 적용 결과로 줄어들 디스크 접근 횟수 (증가된 버퍼 적중 횟수)로 인해 충분히 감쇄 될 것으로 기대된다.

3.2 블록 교체 기법

순차 참조, 순환 참조, 기타 참조로 분류된 블록들은 각각의 참조 특성에 기반하여 버퍼 캐쉬에서 교체된다. 순차 참조에 속하는 블록은 재 참조 가능성이 없기 때문에 가장 최근에 참조된 블록이 우선적으로 버퍼 캐쉬에서 교체된다. 순환 참조에 속하는 블록은 순환 주기가 짧을 수록 재 참조 가능성이 높기 때문에 순환 주기에 기반하여 버퍼 캐쉬에서 교체된다. 이를 위해 순환 주기가 긴 블록부터 짧은 블록 순서로 블록을 교체하고, 동일 순환 주기를 갖는 블록이 여러 개인 경우에는 그들 간에 MRU 블록 교체 기법을 사용하는 순환 주기 기반 블록 교체 기법을 제안한다. 마지막으로 기타 참조에 속하는 블록은 대체로 재 참조 가능성이 과거 참조 정보들(참조 최근성, 참조 빈도 등)에 의해 영향을 받으므로 이들 정보에 기반하여 버퍼 캐쉬에서 교체된다. 이를 위해 Least Recently Used (LRU) 블록 교체 기법, Least Frequently Used (LFU) 블록 교체 기법, LRU-K 블록 교체 기법, Least Recently/Frequently Used (LRFU) 블록 교체 기법 등을 포함한 기존의 블록 교체 기법들 중 하나가 사용된다. 본 논문에서는 이들 중에서 LRU 블록 교체 기법을 가정한다.

3.3 한계 효용에 기반한 버퍼 할당

UBM 기법은 블록들을 참조 패턴에 따라 서로 분리된 공간에 저장하기 때문에 어떤 공간에서 블록을 교체해야 할지를 결정해야 하는 버퍼 할당 문제가 제기된다. 이 문제를 해결하기 위해 UBM 기법은 한계 효용(marginal gain)에 기반하여 한계 효용이 가장 적은 공간으로부터 버퍼를 할당하는 정책을 사용한다 한계 효용의 개념은 최근 데이타 베이스 시스템 분야와 화일 시스템 분야 등에서 자원 할당 문제를 해결하기 위해 많이 사용되었다 [6, 12, 13, 14]. 한계 효용이란 새로운 공간이 추가되었을 때 예상되는 단위 참조당 버퍼 적중 횟수의 증가를 의미한다. 구체적으로 할당된 공간의 크기가 $n-1$에서 n으로 증가할 때 한계 효용은 $MG(n) \simeq Hit(n)-Hit(n-1)$로 정의된다. 여기서 $Hit(n)$은 일정한 크기의 공간 n이 주어졌을 때 예상되는 단위 참조당 버퍼 적중 횟수이다. 각 공간에 대한 $MG(n)$를 온라인에서 구하기 위한 방법은 다음과 같다.

첫 번째로 순차 참조 공간에 대한 한계 효용, $MG_{seq}(n)$은 단위 참조당 예상 버퍼 적중 횟수 $Hit_{seq}(n)=0$이므로 모든 n에 대해 0이다.

두 번째로 순환 참조에 대한 한계 효용, $Hit_{loop}(n)$은 순환 주기 기반 블록 교체 기법을 적용할 때 단위 참조당 예상 버퍼 적중 횟수를 다음과 같이 계산하여 유도한다. 먼저 하나의 순환 참조 i에 대한 단위 참조당 예상 버퍼 적중 횟수는 다음과 같다.

$$Hit_{loop_i}(n)=\frac{\min[\,l_i,n]}{p_i}$$

(여기서 n은 캐쉬 크기이고, l_i는 순환 참조 i의 크기이며, p_i는 순환 참조 i의 주기이다). 따라서 한계 효용은 다음과 같이 계산된다.

$$MG_{loop_i}(n)=\frac{n}{p_i}-\frac{n-1}{p_i}=\frac{1}{p_i}, \qquad n\le l_i$$

$$MG_{loop_i}(n)=\frac{l_i}{p_i}-\frac{l_i}{p_i}=0, \qquad n> l_i.$$

서로 다른 순환 주기를 갖는 다수의 순환 참조들에 대한 단위 참조당 예상 버퍼 적중 횟수와 한계 효용 다음과 같이 계산된다.

$$Hit_{loop}(n)=\sum_{k=1}^{i_{\max}}\frac{l_k}{p_k}+\frac{\min[\,l_{i_{\max}+1},n-m]}{p_{i_{\max}+1}}$$

$$MG_{loop}(n)=\frac{1}{p_{i_{\max}+1}}, \qquad n\le \sum_{k=1}^{i_{\max}+1} l_k$$

(여기서 순환 참조들은 순환 주기에 따라 오름차순으로 정렬되어 있고, 주어진 캐쉬 크기 보다 더 많은 수의 순환 참조 블록들이 있다고 가정한다. 이때 $i_{\max}$는 $m=\sum_{k=1}^{i} l_k < n$인 최대 값이다).

마지막으로 기타 참조에 대해 LRU 블록 교체 기법을 적용할 때 예상되는 단위 참조당 버퍼 적중 횟수는 가상 버퍼 (ghost buffer) [6,12]와 Belady의 lifetime 함수를 이용하여 계산된다. 가상 버퍼만을 이용한 단위 참조당 버퍼 적중 횟수 예측 방법은 매번 참조시 참조된 블록의 위치를 파악해야 하는 처리비용 때문에 비실용적이다. UBM 기법에서는 단위 참조당 예상 버퍼 적중 횟수를 좀더 효율적으로 구하기 위해 [14]에서 제안한 근사 방법을 사용한다. 이 방법은 Belady의 lifetime 함수를 이용하여 임의의 캐쉬 크기에서의 단위 참조당 예상 버퍼 적중 횟수, $Hit_{other}(n)$을 구한다. Belady의 lifetime 함수는 다음과 같다.

$$Hit_{other}(n) = h_1 + h_2 + h_3 + \ldots + h_n \approx 1 - c * n^{-k}$$

여기서 h_i는 LRU 리스트의 i번째 위치에서의 단위 참조당 버퍼 적중 횟수이며 c와 k는 제어 변수이다. 이 제어 변수들의 값은 온라인에서 측정한 특정 캐쉬 크기에서의 단위 참조당 버퍼 적중 횟수를 이용해 계산된다. 이전 방식과 동일한 점은 여전히 가상 버퍼를 이용한다는 점이고, 반면에 차이점은 매번 참조시 참조된 정확한 위치를 파악할 필요가 없다는 점이다. 예로, c와 k 제어 변수 값들을 결정하기 위해서는 최소한 임의의 두 캐쉬 크기 p와 q에서의 단위 참조당 버퍼 적중 횟수, $Hit_{other}(p)$와 $Hit_{other}(q)$ $(q > p)$만 측정하면 되기 때문에 참조 성공한 블록이 캐쉬 크기 q내에 속하는지 그리고 캐쉬 크기 p내에 속하는지만 파악하면 된다. 이렇게 측정한 $Hit_{other}(p)$와 $Hit_{other}(q)$를 이용하여 제어 변수 c와k를 계산함으로써 lifetime 함수를 완성할 수 있다. 그리고 완성된 lifetime 함수를 이용하여 $MG_{other}(n) = Hit_{other}(n) - Hit_{other}(n-1) = h_n$를 계산한다.

그림 4는 순차 참조 공간과 순환 참조 공간, 그리고 기타 참조 공간의 전형적인 $MG(n)$을 그 공간의 크기 n에 따라 보여준다. UBM 기법에서 순차 참조 공간의 $MG_{seq}(n)$이 언제나 '0'이기 때문에 버퍼 할당 모듈은 이 공간에 하나의 버퍼만을 할당한다. 따라서 버퍼 할당은 순차 참조 공간과 순환 참조 공간간에 이루어진다. UBM 기법은 순환 참조 공간의 $MG_{loop}(n)$와 기타 참조 공간의 $MG_{other}(n)$을 비교하여 값이 큰 공간에 더 많은 수의 버퍼를 할당함으로써, 궁극적으로 $MG_{loop}(n)$과 $MG_{other}(n)$이 수렴하도록 각 공간에 할당되는 버퍼의 수를 동적으로 조절한다.

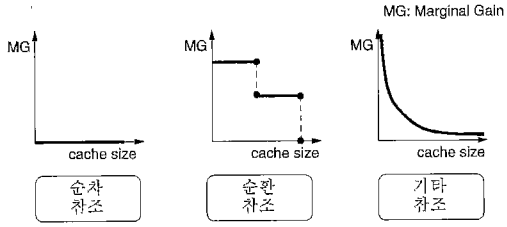


그림 4 통합 버퍼 관리 기법에서 버퍼 할당

3.4 UBM 기법의 동작

그림 5는 UBM 기법을 위한 버퍼 캐쉬 관리자의 전체적인 구조와 동작을 보인다. 새로 참조된 블록이 버퍼 캐쉬에서 접근 실패가 발생한 경우 이 참조는 순차 참조와 순환 참조를 검출하는 부분 (*Detector*)을 통과하여 순차 참조, 순환 참조, 기타 참조로 분류된다 (단계 (1)). 이 과정은 버퍼 캐쉬에서 접근 성공이 발생한 경우에도 검출 정보를 갱신하기 위해 수행된다. 만약 기타 참조로 분류된 경우, 이 블록을 저장하기 위한 버퍼를 얻기 위해 버퍼 할당 모듈 (*Allocator*)을 호출한다 (단계 (2)). 버퍼 할당 모듈은 한계 효용에 기반하여 순환 참조 공간과 기타 참조 공간들 중 $MG(n)$이 작은 공간을 버퍼 추출을 위한 공간으로 선택한다 (단계 (3)). 그러나 만약 순차 참조 공간에서 사용중인 버퍼가 하나 이상인 경우에는 우선적으로 이 공간에서 먼저 추출한다. 선택된 공간의 캐쉬 관리 모듈은 그 공간에 적용된 블록 교체 기법에 의해 하나의 블록을 교체하고 (단계 (4)), 사용했던 버퍼를 반환한다 (단계 (5)). 버퍼 할당 모듈은 반환된 새 버퍼를 접근 실패가 발생한 블록이 저장될 기타 참조 공간에 할당한다 (단계 (6)). 디스크

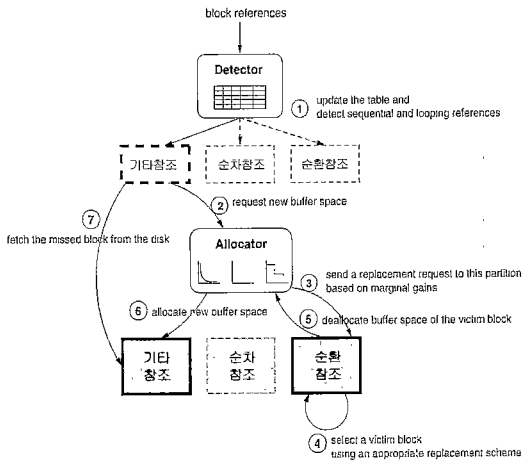


그림 5 통합 버퍼 관리 기법의 전체적인 구조와 동작

로부터 데이타를 읽어서 새 버퍼에 기록한다 (단계 (7)).

4. 성능 평가

본 장에서는 트레이스 기반 시뮬레이션 방법으로 UBM 기법의 성능과 LRU, 2Q, SEQ, EELRU, 그리고 OPT 블록 교체 기법들의 성능을 비교한다[3].

4.1 실험 환경

실험에 사용된 트레이스들은 FreeBSD 운영체제에서 여러 응용들을 동시에 수행시키면서 수집되었다[15]. 트레이스 수집시 사용된 시스템은 주기억장치의 용량이 64M인 Intel Pentium PC이다. 각 응용들의 특성은 다음과 같다.

cscope Cscope은 대화형 C-프로그램 검색 도구로 검색 대상이 되는 C-프로그램들로부터 인덱스 화일을 생성한 후, 요청된 C 심볼과 함수를 찾는 도구이다. 검색 대상으로 사용된 C-프로그램들의 전체 크기는 9MB이다. 수행시 기타 참조 패턴과 하나의 순환 참조 패턴이 검출된다.

glimpse Glimpse는 텍스트 문서 검색 도구로 검색 대상이 되는 텍스트 문서들로부터 인덱스 화일을 생성한 후, 요청된 단어를 찾는 도구이다. 검색 대상으로 사용된 텍스트 문서들의 전체 크기는 50MB이다. 수행시 기타 참조 패턴, 순차 참조 패턴, 그리고 서로 다른 순환 주기를 갖는 다수의 순환 참조 패턴들이 검출된다.

gnuplot Gnuplot은 대화형 그래프 작성 도구이다. 그래프 작성을 위해 사용된 데이타의 크기는 8MB이다. 수행시 기타 참조 패턴과 하나의 순환 참조 패턴이 검출된다.

cpp Cpp는 GNU C 언어를 위한 전 처리기이다. 응용의 입력으로 사용된 C-프로그램들의 전체 크기는 11MB이다. 수행시 기타 참조 패턴과 순차 참조 패턴이 검출된다.

postgres Postgres는 관계형 DB 시스템이다. 네 개의 *relation*에 대해 *join* 명령을 수행하였으며 각 *relation*의 크기는 150KB, 1.5MB, 7.5MB, 15MB이다. 수행시 기타 참조 패턴, 순차 참조 패턴, 그리고 서로 다른 순환 주기를 갖는 다수의 순환 참조 패턴들이 검출된다.

표 1은 실험에서 사용된 각 트레이스별로 생성시 수행된 응용들과 그들이 참조한 블록 수와 총 참조 횟수를 보여준다. 동시에 수행되는 응용들은 전체 작업 집합의 크기와 수행시 발생하는 참조 패턴들을 고려하여 채택되었다.

표 1 동시에 수행된 응용들의 집합

트레이스명	수행된 응용들	참조 횟수	참조된 블록 수
Multi1	cscope, cpp	15858	2606
Multi2	cscope, cpp, postgres	26311	5684
Multi3	cpp, gnuplot, glimpse, postgres	30241	7453

실험에서 UBM 기법과 2Q, SEQ, EELRU 블록 교체 기법들은 하나 이상의 실험 인자들을 가진다. 2Q, SEQ, EELRU 블록 교체 기법들에서 사용되는 실험 인자들은 다음과 같다. 2Q 블록 교체 기법에서는 'A1in', 'A1out'의 크기가 요구된다. SEQ 블록 교체 기법에서는 순차 참조 실패가 발생한 블록들 중 교체 대상이 되는 블록을 선택하기 위한 경계 값들이 요구된다. EELRU 블록 교체 기법에서는 블록이 교체되는 이른 위치 (early eviction point)와 늦은 위치 (late eviction point)들이 요구된다. 실험에서는 2Q [3], SEQ [9], EELRU [10]의 저자들이 제시한 인자 값들을 그대로 사용하였다.

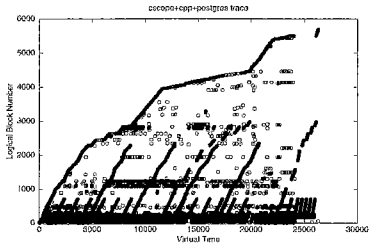
UBM 기법에서 사용되는 실험 인자들은 다음과 같다. 첫 번째는 연속된 블록 참조들 중 순차 참조로 분류하기 위한 임계 값이다. 본 실험에서 *10* 블록 이상 연속된 참조들을 순차 참조로 분류하였다. (실험에서 이 값이 *4 - 40*일 경우 거의 동일한 성능을 보였다). 일반적으로 순차 참조되거나 순환 참조되는 화일의 크기가 크기 때문에 이 값의 변화에 따른 UBM 기법의 성능 변화는 미미하다. 두 번째는 기타 참조 공간의 단위 참조당 버퍼 적중 횟수를 예측하기 위해 온라인에서 측정해야 하는 특정 캐쉬 크기에서의 단위 참조당 버퍼 적중 횟수이다. 실험에서는 다섯 군데 캐쉬 크기들 (전체 버퍼 캐쉬 크기의 10%, 20%, 40%, 60%, 100%)에서의 단위 참조당 버퍼 적중 횟수를 온라인으로 측정하고 이 값들을 이용해 Belady의 lifetime 함수를 완성하도록 하였다.

4.2 실험 결과

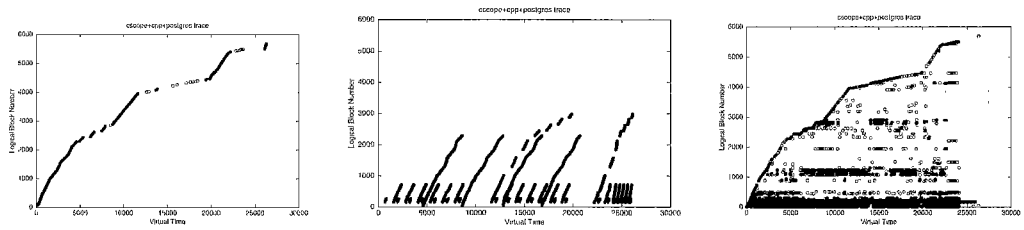
4.2.1 참조 분류 결과

그림 6은 'Multi2' 트레이스의 블록 참조에 대해 UBM 기법이 순차 참조와 순환 참조, 기타 참조로 분류한 결과를 보인다. 그림에서 모든 그래프의 X 축은 시간이고 Y 축은 이 시간에 참조된 블록의 블록 번호이다. 그림 6(a)는 전체 블록 참조들을 시간상에 표현한

3) SEQ와 EELRU 기법은 페이지 교체 기법으로 제안되었지만 본 논문에서는 블록 교체 기법으로 수정하여 사용하였다.



(a) 블록 참조



순차 참조 순환 참조 기타 참조

(b) 순차, 순환, 기타 참조 분류

그림 6 순차, 순환, 기타 참조 분류 결과 (트레이스: Multi2)

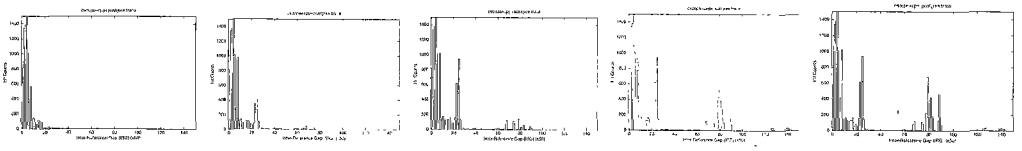
그래프이며 온라인에서 이 참조들로부터 순차 참조와 순환 참조, 기타 참조로 분류된 결과가 그림 6(b)의 그래프들이다.

그림 6(a)에서 10 블록 이상 연속된 블록 참조는 모두 순차 참조로 분류되었다 (그림 6(b)의 순차 참조 그래프 참고). 그리고 이들 순차 참조들은 재 참조 시점에 모두 순환 참조로 분류되었다 (그림 6(b)의 순환 참조 분류 그래프 참고). 그림 6(b)의 순환 참조들만을 표시한 그래프에는 서로 다른 순환 주기를 갖는 다수의 순환 참조들이 보인다. 그림 6(b)의 기타 참조들은 그림 6(a)에서 순차 참조와 순환 참조를 제외한 나머지 참조들이다.

4.2.2 접근 성공한 블록들의 IRG 분포 비교

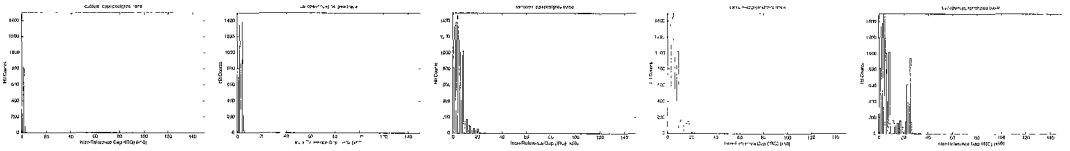
본 절에서는 다양한 캐쉬 크기에서 참조들간 간격, IRG 분포를 통해 미래 참조 정보에 기반한 오프라인 최적 (OPT) 블록 교체 기법과 참조 최근성 정보에 기반한 LRU 블록 교체 기법, 순차 참조와 순환 참조를 고려한 UBM 기법의 성향을 비교 분석한다. 실험에 사용된 트레이스는 6(a)에서 시간 흐름상에 블록 참조를 표현한 'Multi2' 트레이스이며, 순차 참조와 순환 참조들을 포함한 다양한 참조 패턴들이 이 트레이스 상에 존재한다. 그림 7의 그래프들은 다양한 캐쉬 크기에서 OPT, LRU, UBM 기법들을 적용시 관찰되는 동일 블록에 대한 참조들간 간격 (Inter-Reference Gap) (이하 IRG) [5] 분포를 표현한 결과이다. 그림 7(a)와 7(b), 7(c)의 모든 그래프들에서 X 축은 IRG이며 Y 축은 해당 IRG값을 갖는 참조들 중 버퍼 캐쉬에서 접근 성공이 발생한 참조들의 총 횟수 (Hit Counts)이다.

규칙적인 참조 패턴을 보이면서 순환 참조되는 블록들에 대해 OPT 블록 교체 기법은 미래 참조 시점을 알고 블록을 교체하기 때문에, 캐쉬 크기가 증가함에 따라 순환 주기가 짧은 것부터 긴 순서로 순환 참조 블록들을 캐쉬에 유지한다. 그림 7(a) 그래프들에서 캐쉬 크기가 증가함에 따라 IRG 70과 IRG 90 사이의 값을 갖는 참조들에서의 접근 성공 횟수가 점진적으로 증가하는 것을 관찰할 수 있다. 이에 반해 LRU 블록 교체 기법은 순환 참조들의 순환 주기 정보를 이용하지 않고 참조 최근성을 기준으로 블록을 교체하기 때문에, 순차



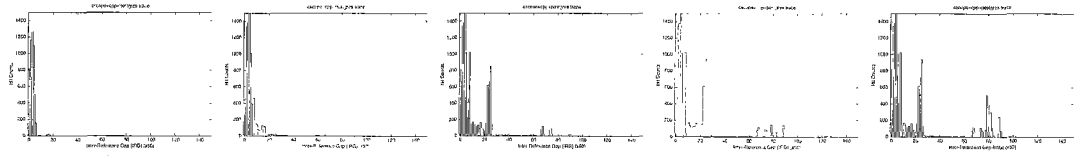
100 캐쉬블록 200 캐쉬블록 600 캐쉬블록 800 캐쉬블록 1000 캐쉬블록

(a) OPT 블록 교체 기법



100 캐쉬블록 200 캐쉬블록 600 캐쉬블록 800 캐쉬블록 1000 캐쉬블록

(b) LRU 블록 교체 기법



100 캐쉬 블록 200 캐쉬 블록 600 캐쉬 블록 800 캐쉬 블록 1000 캐쉬 블록

(c) 통합 버퍼 관리 기법

그림 7 오프라인 최적 (OPT) 블록 교체 기법과 LRU 블록 교체 기법, UBM 블록 교체 기법의 성향 비교 (트레이스: Multi2)

참조 블록들과 순환 주기가 긴 순환 참조 블록들이 참조될 때 오히려 순환 주기가 상대적으로 짧은 순환 참조 블록들이 캐쉬에서 교체한다. 또한 많은 경우 순환 주기가 긴 이러한 순환 참조 블록들은 재 참조되기 전에 캐쉬에서 제거된다. 그림 7(b) 그래프들에서 캐쉬 크기가 증가함에 따라 IRG 70과 IRG 90 사이의 값을 갖는 참조들에서의 접근 성공 횟수가 그림 7(a)의 결과와는 다르게 증가하지 않는 것을 관찰할 수 있다. 이는 LRU 블록 교체 기법이 순환 참조들을 효과적으로 처리하지 못한다는 점을 시사한다. UBM 기법은 순차 참조와 순환 참조를 자동 검출하고 순환 주기 정보를 이용하기 교체하기 때문에 OPT 블록 교체 기법처럼 순환 주기가 짧은 것부터 긴 순서로 순환 참조 블록들을 캐쉬에 유지한다. 그림 7(c) 그래프들에서 캐쉬 크기가 증가함에 따라 IRG 70과 IRG 90 사이의 값을 갖는 참조들에서의 접근 성공 횟수가 그림 7(a)의 결과와 유사하게 점진적으로 증가하는 것을 관찰할 수 있다. 이는 UBM 기법이 OPT 블록 교체 기법과 유사하게 블록들을 유지하고 있음을 시사한다.

4.2.3 UBM 기법과 다른 기법들과의 성능 비교

그림 8은 버퍼 캐쉬 크기 변화에 따른 UBM 기법과 다른 기법들의 버퍼 적중률을 보인다. 그림에서 UBM 기법은 대부분의 버퍼 캐쉬 크기에서 OPT 블록 교체 기법에 근접하는 좋은 성능을 보인다. 캐쉬 크기 변화에 따른 각 기법의 적중률 변화 분석은 다음과 같다.

SEQ 블록 교체 기법은 긴 순차 참조 실패가 발생한 블록들을 우선적으로 교체하기 때문에 대체적으로 좋은 성능을 보인다. 그러나 순환 참조를 고려하지 않기 때문에 UBM 기법보다 낮은 성능을 보인다.

2Q 블록 교체 기법은 순차 참조 블록과 순환 주기가

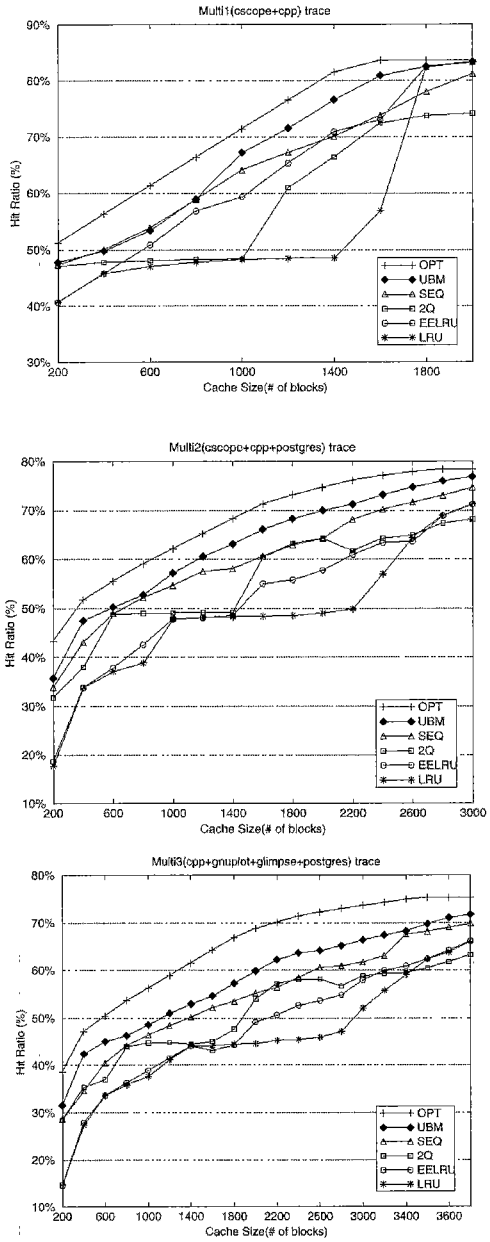


그림 8 UBM 기법과 다른 기법들과의 성능 비교

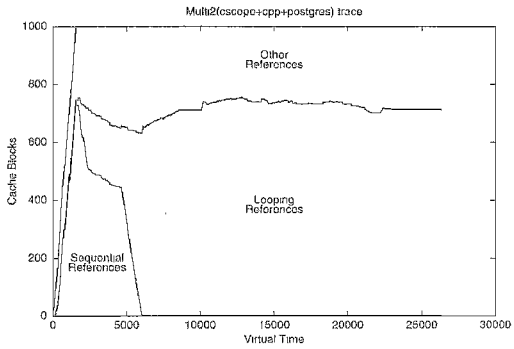
긴 순환 참조 블록들을 우선적으로 제거하기 때문에 대부분의 결과에서 LRU 블록 교체 기법보다 좋은 성능을 보인다. 그러나 캐쉬 크기가 클 때에는 LRU 블록 교체 기법보다 오히려 낮은 적중률을 보이며, 그 이유는 다음과 같다. 첫 번째는 이 기법이 처음 참조되는 모든 블록들을 짧은 기간만 캐쉬에 유지한 후 교체하기 때문에 이들이 재 참조될 때 부가의 캐쉬 접근 실패가 발생한다. 이 실패 횟수는 캐쉬 크기가 증가하면서 전체 캐쉬 접근 실패 횟수에서 상대적으로 점점 더 많은 비중을 차지하게 된다. 두 번째는 서로 다른 순환 주기를 갖는 순환 참조 블록들을 순환 주기에 따라 구분하여 관리하지 않는다. 결과에서 2Q 교체 기법의 성능은 캐쉬 크기가 증가하면서 점진적으로 향상되지 않는다. 특히, 그림 8의 'Multi2' 트레이스와 'Multi3' 트레이스를 이용한 결과에서 캐쉬 크기가 각각 2000 블록, 2800 블록일 때 이 기법의 버퍼 적중률은 이보다 작은 크기일 때 버퍼 적중률보다 오히려 감소하였다. 이는 잘 조절되지 못한 크기의 'A1out' 큐로 인해 버퍼 캐쉬에 일시적으로 순환 주기가 짧은 순환 참조 블록들이 유지되지 못했기 때문이다.

EELRU 블록 교체 기법은 캐쉬 크기가 증가하면서 LRU 교체 기법과 비슷하거나 좋은 성능을 보인다. 하지만 모든 경우에서 UBM 기법보다 낮은 성능을 보인다. 이 방식은 단지 참조 최근성 분포에 기반해서 블록을 교체하기 때문에 순차 참조와 순환 주기가 긴 순환 참조 블록들을 신속히 제거하지 못하며, 2Q 블록 교체 기법과 마찬가지로 서로 다른 순환 주기를 갖는 순환 참조 블록들을 순환 주기에 따라 구분하여 관리하지 않는다.

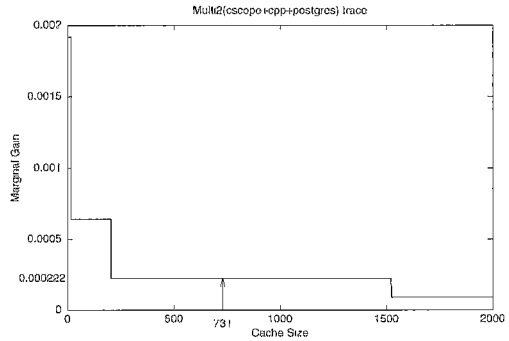
기존의 기법들에 비해 본 논문에서 제시한 UBM 기법은 순차 참조 블록들을 신속히 제거하며,한계 효용에 기반한 버퍼 할당을 통해 캐쉬 크기가 증가함에 따라 기타 참조 블록들과 순환 주기가 짧은 것부터 긴 순서로 순환 참조 블록들을 유지하여 LRU 블록 교체 기법에 비해 평균 12%, 최대 28%까지 향상된 성능을 보인다.

4.2.4 동적 버퍼 할당 결과

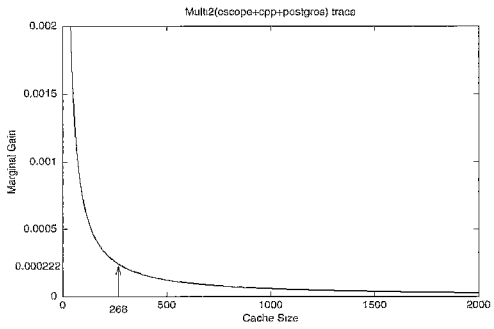
그림 9(a)는 'Multi2' 트레이스에 대해 캐쉬 크기가 1000 블록일 때 시간 흐름에 따라 순차 참조, 순환 참조, 기타 참조 공간에게 할당되는 버퍼의 양을 보여준다. 그래프의 X 축은 시간이고 Y 축은 전체 버퍼 캐쉬의 크기이다. 그래프에서 대략 2000 시각까지는 아직 버퍼 캐쉬 전체가 사용되지 않은 상태이다. 2000 시각 이후부터는 순환 참조 공간과 기타 참조 공간이 증가되고,순차 참조 공간은 줄어들어, 대략 6000 시각 이후부터는 순차 참조 공간에는 하나의 디스크 블록을 저장할 공간만 주어진다. 그림에서 대략 10000 시각 이후부터는 기타 참조 공간과 순환 참조 공간에 할당된 버퍼의 비율이 안정화되는 것을 볼 수 있다.



(a) 동적 버퍼 할당



순환 참조



기타 참조

(b) 단위 공간 증가당 예상 버퍼 적중 증가율 (시간 20000 지점)

그림 9 동적 버퍼 할당 결과(캐쉬 크기: 1000 블록, 트레이스: Multi2)

그림 9(b)는 그림 (a)의 20000 시각에서 계산된 순환 참조 공간의 $MG_{loop}(n)$과 기타 참조 공간의 $MG_{other}(n)$을 그 공간에 할당될 버퍼의 양에 따라 보여준다. 그림 (b)의 X 축은 공간의 크기이며 Y 축은 실제 계산된 $MG(n)$ 값이다. 실험에 사용된 'Multi2' 트레이스에는 서로 다른 순환 주기를 갖는 다수의 순환 참조들이 존재하기 때문에 그림 (b)의 왼쪽 그래프에서 순환 참조 공간에 할당된 버퍼의 양이 증가할수록 예측된 $MG_{loop}(n)$ 값이 단계적으로 감소하는 것을 볼 수 있다. 그림 (b)의 오른쪽 그래프에서는 온라인에서 측정된 특정 캐쉬 크기에서의 단위 참조당 버퍼 적중 횟수를 이용해 계산된 $MG_{other}(n)$ 값을 보인다. UBM 기법은 온라인에서 예측된 이 두 공간들 중 $MG(n)$ 값이 작은 공간의 버퍼를 교체하므로 점차 두 공간의 $MG(n)$ 값이 하나의 값으로 수렴한다. 실험에서 20000 시각일 때 두 공간에서 예측된 $MG(n)$ 값은 대략 0.000222에 수렴하며, 이때 할당된 블록 수는 그림 (b)에 표시된 바와 같이 순환 참조 공간에 대략 731 블록, 기타 참조 공간에 대략 268 블록이다.

5. 결 론

본 논문에서는 다중 응용 트레이스를 이용하여 Least Recently Used (LRU) 블록 교체 기법의 문제점을 관찰하고, 이 문제점을 해결하는 통합된 형태의 효율적인 버퍼 관리 (Unified Buffer Management, 이하 UBM) 기법을 제안하였다. UBM 기법은 순차 참조와 순환 참조를 자동 검출하여 분리된 공간에 저장하고 순환 주기를 고려한 블록 교체 기법으로 이 공간을 관리한다. 반면 여기에 속하지 않는 블록들은 LRU 블록 교체 기법으로 관리되는 공간에 저장된다.또한 이 두 공간의 크기의 비율은 온라인에서 수집된 정보를 이용하여 계산된 단위 공간 증가당 예상 버퍼 적중 증가율 (marginal gain of buffer hit ratio)에 기반하여 동적으로 조절된다.

UBM 기법의 성능을 평가하기 위해 본 논문에서는 FreeBSD 운영체제에서 실제 응용들을 수행시켜 수집한 트레이스를 사용하여 시뮬레이션을 수행하였다. 실험에서 UBM 기법이 1) 순차 참조와 순환 참조들 대부분을 자동 검출하고, 2) 접근 성공한 블록들의 참조간 간격 (Inter-Reference Gap) 분포 결과를 통해서 오프라인 최적 (OPT) 블록 교체 기법과 유사하게 순차 참조와 순환 참조들을 처리하며, 3) 버퍼 적중률을 LRU 블록 교체 기법에 비해 평균 12%, 최대 28%까지 향상시킴을 보였다.

향후 연구로 순차 참조와 순환 참조에 속하지 않는 기타 참조 블록들을 참조 최근성만을 고려한 LRU 블록 교체 기법이 아닌 참조 최근성과 참조 빈도를 함께 고

려한 LRFU 블록 교체 기법으로 관리하기 위해 UBM 기법을 확장할 계획이며, 상용 UNIX 시스템에 구현하여 UBM 기법의 성능과 수행 오버헤드를 평가할 계획이다.

참 고 문 헌

[1] J. T. Robinson and M. V. Devarakonda. Data Cache Management Using Frequency-Based Replacement. In *Proceedings of the 1990 ACM SIGMETRICS Conference on Measurement and Modeling of Computer Systems*, pages 134-142, 1990.

[2] E. J. O'Neil, P. E. O'Neil, and G. Weikum. The LRU-K Page Replacement Algorithm for Database Disk Buffering. In *Proceedings of the 1993 ACM SIGMOD Conference*, pages 297-306, 1993.

[3] T. Johnson and D. Shasha. 2Q : A Low Overhead High Performance Buffer Management Replacement Algorithm. In *Proceedings of the 20th International Conference on VLDB*, pages 439-450, 1994.

[4] P. Cao, E. W. Felten, and K. Li. Application-Controlled File Caching Policies. In *Proceedings of the USENIX Summer 1994 Technical Conference*, pages 171-182, 1994.

[5] V. Phalke and B. Gopinath. An Inter-Reference Gap Model for Temporal Locality in Program Behavior. In *Proceedings of the 1995 ACM SIGMETRICS Conference on Measurement and Modeling of Computer Systems*, pages 291-300, 1995.

[6] R. H. Patterson, G. A. Gibson, E. Ginting, D. Stodolsky, and J. Zelenka. Informed Prefetching and Caching. In *Proceedings of the 15th Symposium on Operating System Principles*, pages 1-16, 1995.

[7] D. Lee, J. Choi, S. H. Noh, S. L. Min, Y. Cho, and C. S. Kim. On the Existence of a Spectrum of Policies that Subsumes the Least Recently Used (LRU) and Least Frequently Used (LFU) Policies. In *Proceedings of the 1999 ACM SIGMETRICS Conference on Measurement and Modeling of Computer Systems*, pages 134-143, 1999.

[8] E. G. Coffman, JR. and P. J. Denning. *Operating Systems Theory*. Prentice-Hall International Editions, 1973.

[9] G. Glass and P. Cao. Adaptive Page Replacement Based on Memory Reference Behavior. In *Proceedings of the 1997 ACM SIGMETRICS Conference on Measurement and Modeling of Computer Systems*, pages 115-126, 1997.

[10] Y. Smaragdakis, S. Kaplan, and P. Wilson. EELRU: Simple and Effective Adaptive Page Replacement. In *Proceedings of the 1999 ACM SIGMETRICS Conference on Measurement and Modeling of Computer Systems*, pages 122-133, 1999.

[11] T. H. Cormen, C. E. Leiserson, and R. L. Rivest. *Introduction to Algorithms*. The MIT Press and McGraw-Hill Book Company, 1992.

[12] D. Thiebaut, H. S. Stone, and J. L. Wolf. Improving Disk Cache Hit-Ratios Through Cache Partitioning. *IEEE Transactions on Computers*, 41(6):665-676, 1992.

[13] C. Faloutsos, R. Ng, and T. Sellis. Flexible and Adaptable Buffer Management Techniques for Database Management Systems. *IEEE Transactions on Computers*, 44(4):546-560, 1995.

[14] J. Choi, S. Cho, S. H. Noh, S. L. Min, and Y. Cho. Analytic Prediction of Buffer Hit Ratios. *IEE Electronics Letters*, 36(1):10-11, 2000.

[15] J. Choi, S. H. Noh, S. L. Min, and Y. Cho. An Implementation Study of a Detection-based Adaptive Block Replacement Scheme. In *Proceedings of the 1999 USENIX Annual Technical Conference*, pages 239-252, 1999.

김 종 민

1992년 부산대학교 공과대학 컴퓨터공학과 학사. 1994년 서울대학교 공과대학 컴퓨터공학과 석사. 1996년 ~ 현재 서울대학교 공과대학 컴퓨터공학부 박사과정. 관심분야는 캐쉬 메모리 (CPU 캐쉬, 버퍼 캐쉬, 디스크 캐쉬 관리), 가상 메모리, 파일시스템 (RAID 시스템, FLASH 메모리 기반 시스템)

최 종 무

1993년 서울대학교 해양학과 이학사. 1995년 서울대학교 컴퓨터공학과 공학석사. 1995년 ~ 현재 서울대학교 컴퓨터공학과 박사과정. 관심분야는 운영체제, 입출력 시스템, 데이타 마이닝

김 제 성

1991년 서울대학교 공과대학 컴퓨터공학과 학사. 1993년 서울대학교 대학원 컴퓨터공학과 석사. 1998년 서울대학교 대학원 컴퓨터공학과 박사. 1998년 ~ 2000년 현대전자산업(주) 네트웍 시스템 연구팀. 2000년 ~ 현재 서울대학교 BK21 정보기술사업단 박사후 연구원. 관심분야는 캐쉬 메모리, 파일 시스템, Personal Area Network (Bluetooth)

이 동 희

1989년 서울대학교 컴퓨터공학과 공학사. 1991년 서울대학교 컴퓨터공학과 공학석사. 1998년 서울대학교 컴퓨터공학과 공학박사. 1999년 ~ 현재 제주대학교 통신 컴퓨터 공학부 조교수. 관심분야는 시스템 소프트웨어, 입출력 시스템

노 삼 혁

1986년 서울대학교 컴퓨터공학과 공학사. 1993년 메릴랜드대학교 컴퓨터공학과 박사. 1994년 ~ 현재 홍익대학교 컴퓨터공학과 조교수. 관심분야는 시스템 소프트웨어, 병렬처리 시스템

민 상 렬

1983년 서울대학교 전자계산기공학과, 공학사. 1985년 서울대학교 전자계산기공학과 공학석사. 1989년 워싱턴대학 전산학과 박사. 1989년 ~ 1990년 IBM Watson 연구소 객원 연구원. 1990년 ~ 1992년 부산대학교 컴퓨터공학과 조교수. 1992년 ~ 현재 서울대학교 컴퓨터공학부 조교수. 관심분야는 컴퓨터구조, 병렬처리 시스템, 캐쉬 메모리 시스템 등

조 유 근

1971년 서울대학교 공대 졸업. 1978년 미네소타대학교 전산학과 박사. 1979년 ~ 현재 서울대학교 컴퓨터공학부 교수. 관심분야는 알고리즘, 운영체제, 데이타 구조 등

김 종 상

1960년 서울대학교 공과대학 전자공학과 학사. 1965년 서울대학교 공과대학 전자공학과 석사. 1975년 서울대학교 공과대학 전자공학과 박사. 1979년 ~ 현재 서울대학교 공과대학 컴퓨터공학부 교수. 1986년 ~ 1988년 한국정보과학회 회장.