論文
5-1-7

2-레벨 디스크 캐쉬 시스템에서 디스크 블록 중복 저장을 최소화하는 효율적인 캐싱 알고리즘

(An Efficient Caching Algorithm to Minimize Duplicated Disk Blocks in 2-level Disk Cache System)

류갑상(Gab-Sang Ryu)[1], 정수목(Soo-Mok Jung)[2]

요 약

처리기와 디스크의 속도 차가 커지고 있어 I/O subsystem이 컴퓨터 시스템의 성능향상에 병목 현상을 일으키게 된다. 이러한 처리기와 디스크와의 속도 차를 극복하기 위한 한 방법으로 캐쉬가 사용되고 있다. 캐쉬를 사용하면 디스크 블록에 대한 접근 횟수를 줄일 수 있어 전체 시스템의 성능을 향상시킬 수 있다. 본 논문에서는 버퍼 캐쉬와 디스크 캐쉬를 가지는 시스템에서 서로 독립적으로 캐쉬가 관리되어 다수의 디스크 블록이 중복되게 유지되는 문제를 해결하기 위하여 디스크 블록의 중복을 최소화함으로 시스템의 성능을 개선하는 캐쉬 관리 기법을 제안하였다. 시뮬레이션을 통하여 제안된 기법을 적용하였을 경우 디스크 블록에 대한 평균 접근 지연시간이 감소됨을 확인하였다.

ABSTRACT

The speed gap between processors and disks is a serious problem. So, I/O sub-system limits the performance of computer system. To overcome the speed gap, caches have been used in computer system. By using cache, the access times to disk blocks can be reduced and the performance of computer system can be improved. In this paper, we proposed an efficient cache management algorithm for computer system which have buffer cache and disk cache. The proposed algorithm can minimize the duplicated blocks between buffer cache and disk cache. We evaluate the proposed algorithm by trace-driven simulation. The simulation results show that the proposed algorithm can reduce the mean access time to disk blocks.

논문접수 : 2004. 1. 5.
심사완료 : 2004. 1. 12.

1) 정회원 : 동신대학교 교수
2) 정회원 : 삼육대학교 컴퓨터과학과 부교수

1. 서론

VLSI기술의 발전으로 처리기는 고속화되고 있으나 기계적인 동작에 기초한 디스크의 속도는 처리기의 속도증가율을 따르지 못하고 있어 처리기와 디스크의 속도 차가 갈수록 커지고 있다. 이러한 속도 차는 컴퓨터시스템의 전체적인 성능을 저하시키는 요인이 되고 있다. 이러한 문제를 해결하기 위하여 Dynamic RAM 메모리로 구성되는 디스크 캐쉬를 사용하고 있다. [1,2].

디스크 캐쉬의 예로 UNIX운영체제 내부의 버퍼 캐쉬(buffer cache)를 들 수 있는데 이것은 운영체제 커널 상에 유지되는 버퍼이다. 버퍼 캐쉬에는 최근에 참조된 디스크 블록들이 유지된다. 처리기에서 블록에 대한 요청이 있을 때 먼저 버퍼 캐쉬 내에서 해당 데이터 블록이 있는지 찾게 된다. 버퍼 캐쉬 내에 원하는 데이터 블록이 있는 경우 이 데이터 블록이서비스 되고 버퍼 캐쉬 내에 원하는 데이터 블록이 없으면 디스크를 접근하여 데이터 블록을 읽어 버퍼 캐쉬에 저장한 후 처리기에 서비스한다. 따라서 버퍼 캐쉬 내에 원하는 디스크 블록이 유지되어 있는 경우 디스크로부터 디스크 블록 읽는데 따르는 부하를 줄일 수 있게 된다.[3]

처리기의 제어를 받아 입출력이 이루어지는 버퍼 캐쉬와 별도로 디스크 제어기에 자체적인 디스크 캐쉬와 I/O처리기를 가지고 있는 시스템이 개발되었다. 이러한 시스템의 디스크 제어기는 디스크 캐쉬와 디스크 입출력 처리기를 가가고 있어 처리기의 간섭 없이 처리기와 병렬적으로 입출력을 할 수 있다. 디스크 제어기에 존재하는 디스크 캐쉬는 운영체제와 관계없이 독자적으로 디스크 시스템의 성능을 향상시킨다[4,5]

그림1과 같이 버퍼 캐쉬와 디스크 캐쉬를 동시에 가지는 시스템에서 사용자 프로세스가 입출력 요구 시 UNIX운영체제는 먼저 버퍼 캐쉬를 검사하여 원하는 디스크 블록이 있는지 확인하다. 원하는 디스크 블록이 있는 경우에는 이를 사용하고 원하는 디스크 블록이 버퍼 캐쉬에 없는 경우에는 디스크 제어기에게 디스크 블록을 요청하게 된다.

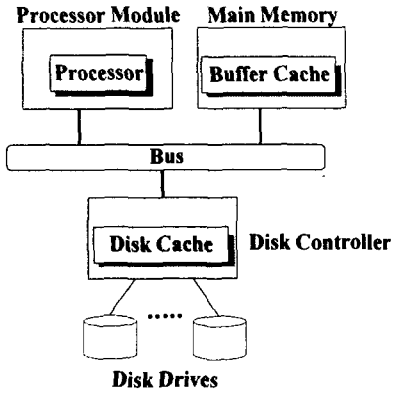


그림 1. Buffer Cache와 Disk Cache를 가지는 시스템

디스크제어기는 운영체제에 의해서 요청된 디스크 블록을 디스크 캐쉬에서 검사하게 된다. 요청된 디스크 블록이 디스크 캐쉬에 존재하면 이 블록은 버퍼 캐쉬에 저장되고 처리기에 서비스되어 사용자 프로세스에 의해서 사용된다. 만약 디스크 캐쉬에서 원하는 디스크 블록이 없을 경우 디스크 입출력을 하게 된다. 처리기와 독립적으로 동작하는 디스크 제어기에 존재하는 디스크 캐쉬는 디스크 블록을 캐싱하고 있어 디스크 시스템의 성능향상에 기여한다. 따라서 디스크 캐쉬는 2차 적인 캐쉬의 역할을 한다. 이러한 시스템에서 버퍼 캐쉬와 디스크 캐쉬는 다수의 디스크 블록들을 중복하여 가지게 되어 캐쉬 활용이 효율적이지 못하게 되고 시스템의 성능이 저하된다[6]. 버퍼 캐쉬와 디스크 캐쉬의 디스크 블록 중복 유지를 제한하여 시스템의 성능향상을 이루는 효율적인 캐쉬 관리기법들이 연구되고 있다. [7-12].

본 논문에서는 디스크 캐쉬와 버퍼 캐쉬를 가지는 시스템에서 프로세스에 의해 요청되는 디스크 블록에 대한 평균 서비스 시간을 줄이기 위하여 디스크 블록의 중복을 제한하는 sLRU(segmented LRU)기법[12]의 성능을 개선시킨 효율적인 캐쉬 관리기법을 제안하였다.

2. sLRU(Segmented Least Recently Used)

사용자 프로세스에 의하여 참조된 디스크 블록은 일정 시간동안에 강한 지역성을 보이고 있다[9]. 따라서 사용자 프로세스에 의하여 디스크 블록이 참조되어 캐쉬로 올라온 후 일정한 시간 안에 두 번째 참조가 발생하지 않으면 이 블록은 캐쉬에서 제거 될 때까지 계속해서 참조가 되지 않는 경향이 있다. 또한 일정 시간 안에 두 번 째 참조가 발생하면 캐쉬에서 제거되기 전까지 여러 번 계속적으로 사용되는 경향이 있다.

sLRU(segmented LRU)[12]에서는 일정 시간 안에 블록에 대한 두 번째 참조가 발생하지 않으면 디스크 블록을 캐쉬에서 제거한다. 따라서 sLRU를 사용하면 순차참조와 같이 한번 사용된 뒤 다시는 참조되지 않는 디스크 블록이 계속 캐쉬로 적재되어 참조될 확률이 큰 다른 디스크 블록을 축출하는 현상을 방지하므로 캐쉬의 적중률을 높일 수 있다. 한번만 참조되는 디스크 블록과 재 참조되어 여러 번 반복적으로 참조될 가능성이 높은 디스크 블록을 구별하기 위하여 sLRU기법을 적용한 sLRU캐쉬에서는 그림 2와 같이 임시 세그먼트(probationary segment)와 보호 세그먼트(protected segment)로 나누어지고, sLRU 캐쉬의 세그먼트들은 단일 연결 리스트(single linked list)로 구성된다. 이 리스트는 금지 세그먼트의 MRU end에서 임시 세그먼트 LRU end까지 연결된다. 캐쉬 미스(cache miss)에 의하여 데이터 블록이 캐쉬로 읽혀오면 임시 세그먼트의 MRU end에 추가된다. 디스크 블록이 임시 세그먼트 내에 있고 캐쉬 히트가 발생할 경우에는 디스크 블록이 임시 세그먼트에서 제거되고 보호 세그먼트의 MRU end에 추가된다. 이를 위하여 보호 세그먼트내의 LRU end에 있는 마지막 디스크 블록이 임시 세그먼트의 MRU end로 이동된다.

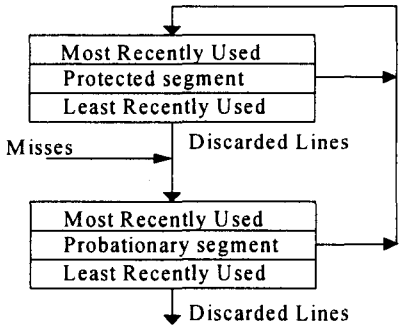


그림 2. sLRU 캐쉬의 논리적인 흐름도

따라서 보호세그먼트의 하나의 디스크 블록이 임시 세그먼트내의 하나의 디스크 블록과 교체된다. 디스크 블록이 보호 세그먼트 내에 있고 캐쉬 히트(cache hit)가 발생할 경우에는 디스크 블록이 보호 세그먼트의 MRU end로 이동하게 된다. 따라서 보호 세그먼트에 있는 디스크 블록은 적어도 한 번 이상은 접근된 것들이다. 캐쉬에서 하나의 데이터 블록이 버려질 경우에는 임시 세그먼트의 LRU end에 있는 하나의 데이터 블록이 버려지게(flush) 된다. 임시 세그먼트와 금지 세그먼트를 구분하기 위하여 boundary pointer를 가지고 있다. 이 포인터는 일반적으로 리스트의 중간 부분을 가리키도록 set된다. 그리고 각 캐쉬라인은 1 bit flag를 가지게 되는데 이는 어느 세그먼트에 있는 지를 나타내는데 사용된다. 캐쉬 미스에 의해서 새로운 디스크 블록이 임시 세그먼트의 MRU end와 보호 세그먼트의 LRU end 사이에 삽입되고 포인터는 새로운 디스크 블록을 가리키도록 조정되고 flag는 임시 세그먼트에 있음을 나타내도록 세트된다. 이와 같이 캐쉬 미스에 의하여 임시 세그먼트에 새로운 디스크 블록이 삽입되는 경우 임시 세그먼트내의 디스크 블록은 새로운 디스크 블록을 위하여 캐쉬로부터 제거되어야 하는데 임시 세그먼트의 LRU end의 디스크 블록이 선택되어 버려지게(flush) 된다. 이는 frequency-based LRU정책의 간단한 변형에 해당한다.

디스크 캐쉬와 버퍼 캐쉬를 가지는 시스템에서 프로그램의 참조 패턴에 의한 디스크 블록

분배 방법이 제안되었다[13]. 이 기법에서는 (그림 3)과 같이 버퍼 캐쉬는 sLRU 캐쉬이고 디스크 캐쉬는 단순 연결리스트로 구성되어 있다. 처음으로 참조되어 캐쉬로 올라온 디스크 블록은 버퍼 캐쉬의 임시 세그먼트와 디스크 캐쉬로 동시에 진입한다(그림 3a와 b에서 화살표 1). 블록이 임시 세그먼트에서 머무르는 시간(Δ_t)안에 두 번째 비연관 참조가 발생하면 이 블록은 보호 세그먼트로 이동하고 디스크 캐쉬에서는 제거된다(그림 3a와 b에서 화살표 2).

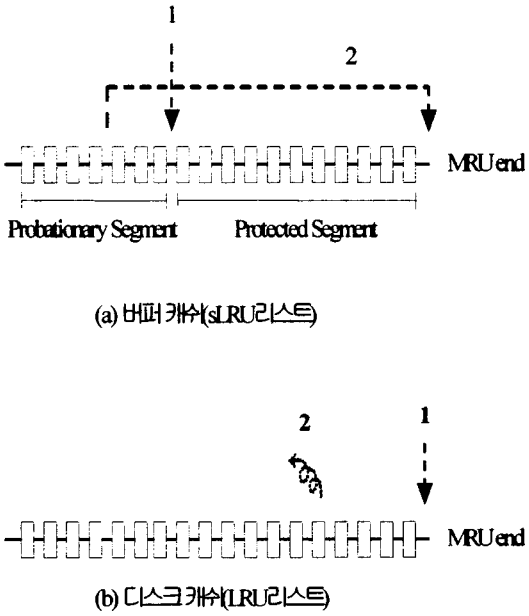


그림 3. 프로그램의 참조 패턴에 의한 분배 기법에서 캐쉬의 구조

Δ_t동안에 비연관 참조가 발생하지 아니하면 버퍼 캐쉬내의 디스크 블록은 임시 세그먼트의 MRU end에서 LRU end로 이동되어 결국에는 버퍼 캐쉬에서 제거된다. 그러나 디스크 캐쉬에서는 MRU end에서 LRU end로 이동하고 LRU end에 도착 후 캐쉬 미스에 의하여 다른 디스크 블록이 디스크에서 읽혀져 오면 LRU end에 있던 디스크 블록이 제거된다. Δ_t는 임시 세그먼트의 크기와 디스크 블록의 캐쉬 진입율에 의해 정해진다.

이 기법에서는 첫째, 처음으로 참조된 디스크 블록이 버퍼캐쉬의 임시 세그먼트 MRU end와 디스크 캐쉬의 MRU end에 함께 삽입된다. 이때 재 참조되어 보호 세그먼트로 진입하기 전까지 혹은 재 참조되지 않아 임시 세그먼트에서 제거 될 때까지 디스크 블록은 버퍼 캐쉬와 디스크 캐쉬에 동시에 존재하게 되는 단점이 있다. 둘째, 디스크로부터 참조된 디스크 블록이 일정한 시간(Δ_t)안에 참조되지 않으면 향후 더 이상 참조될 가능성이 적음에도 디스크 캐쉬에 상당한 기간동안 유지되고 있다.

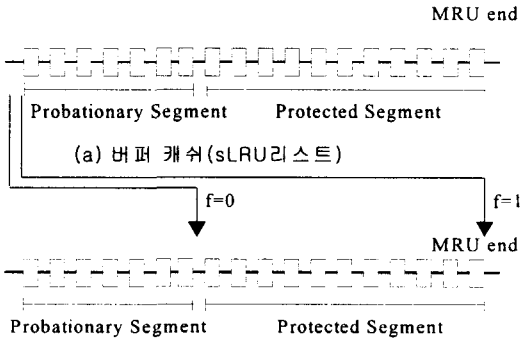
3. 디스크 캐쉬와 버퍼 캐쉬 사이의 중복을 최소화하는 캐쉬 관리 기법

사용자 프로세스에 의하여 참조된 디스크 블록은 일정 시간동안에 강한 지역성을 보이는 가지고 있다.[9]을 효과적으로 . 따라서 일정 시간안에 두 번 째 참조가 발생하면 캐쉬에서 제거되기 전까지 여러 번 계속적으로 사용되는 경향이 있다. 이러한 성질을 효과적으로 이용하기 위하여 본 논문에서는 버퍼 캐쉬와 디스크 캐쉬를 전역적으로 sLRU에 의해서 관리되도록 하는 새로운 캐쉬 관리 기법(gsLRU: global segmented LRU)을 제안하였다. gsLRU 캐쉬는 그림 4와 같이 버퍼 캐쉬와 디스크 캐쉬가 모두 sLRU로 구성되고 관리된다.

디스크 캐쉬와 버퍼 캐쉬를 가지는 시스템에서 디스크로부터의 블록 읽기 처리는 디스크 제어기를 통하여 읽혀진 디스크 블록이 디스크 캐쉬에 저장되고 그 후 버퍼 블록으로 전달되어 제어기에 디스크 블록이 서비스되어 사용자 프로세스가 수행된다. 따라서 이러한 시스템에서는 적어도 1개의 블록은 디스크 버퍼와 버퍼 캐쉬사이에 공유된다.

gsLRU 캐쉬에서 참조되는 디스크 블록이 존재하는 위치에 따라 다음과 같은 4가지의 상황이 발생한다. 첫째, 버퍼 캐쉬내의 임시 세그먼트 - 디스크 블록이 처음 읽혀져 온 후 재 참조가 되지 않은 상태에서 버퍼 캐쉬의 임시 세그먼트에 유지되는 경우, 디스크 블록이 읽혀져 온 후 한 번 이상의 재 참조가 발생하여 버퍼 캐쉬의 보

호 세그먼트에 들어갔다가 더 이상 참조가 발생하지 않아 버퍼 캐쉬의 보호 세그먼트에서 임시 세그먼트로 이동한 후 유지되는 경우이다. 디스크 블록이 버퍼 캐쉬의 임시 세그먼트에서 버퍼 캐쉬의 보호 세그먼트로 이동한다. 이동된 디스크 블록에 대한 1bit flag는 1로 세트되어 보호 세그먼트에 들어 왔음을 기록한다. 이때 디스크 블록 교체가 발생하게 되는데 보호 세그먼트의 LRU end의 디스크 블록이 임시 세그먼트의 MRU end로 이동된다.



(b) 디스크 캐쉬(sLRU리스트)

그림 4 gsLRU 캐쉬 구조

둘째, 버퍼 캐쉬내의 보호 세그먼트 - 디스크 블록이 처음 읽혀져 온 후 최소한 한번 이상의 재참조가 발생하여 버퍼 캐쉬의 보호 세그먼트에 유지되는 상황에서 디스크 블록이 참조 되는 경우이다. 디스크 블록은 보호세그먼트 MRU end로 이동하게 된다.

셋째, 디스크 캐쉬의 보호 세그먼트 - 디스크 블록이 디스크에서 읽혀진 후 일정한 시간 안에 재 참조되어 버퍼 캐쉬의 보호 세그먼트에 유지되다가 그 후 재 참조되지 않아 버퍼 캐쉬의 임시 세그먼트를 거쳐 디스크 캐쉬의 보호 세그먼트에 유지되고 있는 경우이다. 디스크 블록은 버퍼 캐쉬의 MRU end로 이동된다. 이러한 동작을 위하여 먼저 버퍼 캐쉬의 보호 세그먼트의 LRU end에 있는 디스크 블록이 버퍼 캐쉬의 임시 세그먼트로 이동하고, 임시 세그먼트의 LRU end에 있는 디스크 블록에 대한 flag가 0이면 flush 되고 1이면 디스크 캐쉬의 보호 세그먼트의 MRU end로 이동한다.

넷째, 디스크 캐쉬의 임시 세그먼트 - 디스크 블록이 디스크에서 읽혀진 후 일정한 시간 안에 재 참조되어 버퍼 캐쉬의 보호 세그먼트에 유지되다가 그 후 재 참조되지 않아 버퍼 캐쉬의 임시 세그먼트를 거치고 디스크 캐쉬의 보호 세그먼트를 거친 후 디스크 캐쉬의 임시 세그먼트에 유지되는 경우가 있을 수 있고, 디스크 블록이 디스크에서 읽혀진 후 일정한 시간 안에 재 참조되지 않아 버퍼 캐쉬의 임시 세그먼트 LRU end에서 디스크 캐쉬의 임시 세그먼트로 이동되어 유지되고 있는 경우이다. 디스크 블록은 버퍼 캐쉬의 MRU end로 이동되고 flag가 1로 세트된다. 이러한 동작이 일어나기 위하여 셋째의 경우와 동일하게 버퍼 캐쉬의 보호세그먼트의 LRU end 디스크 블록이 임시 세그먼트의 MRU end로 이동하고 임시 세그먼트의 LRU end에 있는 디스크 블록은 flag값에 의하여 flush되거나 디스크 캐쉬의 보호 세그먼트 MRU end로 이동한다.

gsLRU기법에서는 디스크 캐쉬에 포인터를 가져야하는 overhead는 있지만 버퍼 캐쉬와 디스크 캐쉬 사이에 디스크 블록의 중복을 1개로 최소화함으로 재사용가능성이 높은 디스크 블록들을 더 많이 유지할 수 있게 하여 디스크 접근 평균 지연시간이 효과적으로 감소되어 시스템의 성능이 향상되게 된다.

4. 실험결과

본 논문에서는 디스크가 없는 20개의 클라이언트 워크스테이션과 4개의 파일서버로 구성된 네트워크 환경에 대하여 sprite 네트워크 파일 시스템 트레이스[14]를 사용하여 제안된 버퍼 캐쉬 및 디스크 캐쉬 관리 기법에 대하여 실험하였다. 트레이스는 파일에 대한 open, close, lseek, 기타 연산과 디렉토리 검색을 포함하도록 하였다. 버퍼 캐쉬와 디스크 캐쉬의 크기를 16MB 일 때 초기화에 소요되는 요청을 제외한 643,721개의 요청으로 실험결과를 정리하였다. C언어를 사용하여 UNIX의 버퍼 캐쉬와 디스크 캐쉬를 구현한 시뮬레이션 프로그램에서 캐쉬

교체기법으로 LRU기법을 채용하였다. 시스템의 성능을 고려하여 버퍼 캐쉬는 write back policy를 채용하였다. 시뮬레이터는 새로운 디스크 블록을 위한 버퍼를 할당하는 경우 UNIX와 동일하게 버퍼 자유리스트에서 변경되지 않은 버퍼를 발견할 때까지 변경된 버퍼에 대하여 쓰기 동작을 반복하여 요청한다. 디스크에 쓰도록 요청된 버퍼는 I/O리스트에 삽입된다. 쓰기 동작은 I/O리스트에 순서에 따라 완료되도록 하였다. 쓰기가 완료된 버퍼는 플래그가 설정된 후 버퍼 자유리스트의 앞쪽(LRU end)으로 삽입 되도록 하였다.

그림 5는 버퍼 캐쉬와 디스크 캐쉬에서의 적중률과 평균 접근시간을 나타낸다. 버퍼 캐쉬의 적중률은 전체 요청 중에서 버퍼 캐쉬에서 만족된 블록의 비율을 표시한다. 디스크 캐쉬 적중률은 버퍼 캐쉬에서 미스(miss)되어 디스크 캐쉬를 검색한 요청 중 디스크 캐쉬에서 만족된 비율을 나타낸다. sLRU기법이 적용된 경우 버퍼 캐쉬와 디스크 캐쉬의 적중률은 60.3%, 8.6%이고 gsLRU기법이 적용된 경우 버퍼 캐쉬와 디스크 캐쉬의 적중률은 64.1% 12.2% 이다. 제안된 gsLRU기법에서는 버퍼 캐쉬와 디스크 캐쉬에서 디스크 블록의 중복을 최소화하기 때문에 버퍼 캐쉬와 디스크 캐쉬가 보다 많은 중복되지 않은 디스크 블록들을 효과적으로 유지 할 수 있어 캐쉬 적중률이 증가 하게 된다.

블록 평균 접근 지연시간은 그림 5(b)와 같이 sLRU기법의 경우 5.15ms, gsLRU기법의 경우 5.04ms이다. 디스크 블록 평균 접근 지연시간의 감소는 2-레벨 캐쉬가 gsLRU기법을 사용하여 보다 많은 디스크 블록들을 유지할 수 있고 또한 참조될 가능성이 높은 디스크 블록을 버퍼 캐쉬에 유지 할 수 있기 때문이다.

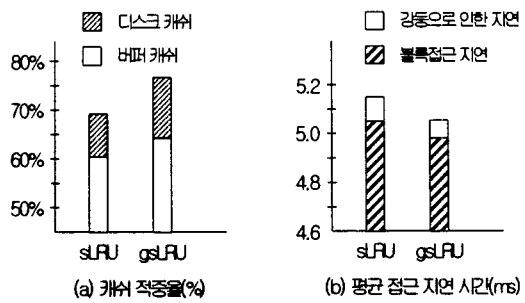


그림5. 각 캐쉬의 적중률과 평균 접근 지연시간

5. 결론

본 논문에서는 버퍼 캐쉬와 디스크 캐쉬를 가지는 2-레벨 계층구조 캐쉬를 가지는 시스템에서 디스크 블록의 중복을 최소화하여 전체 캐쉬 용량을 증가시켜 많은 디스크 블록들이 캐쉬에서 유지되도록 하여 캐쉬 적중률을 높이고, 빈번히 참조하는 디스크 블록을 접근시간이 적은 버퍼 캐쉬에 유지하여 평균적인 접근 지연 시간을 최소화하는 효율적인 캐쉬 관리 알고리즘을 제안하고 트레이스 기반 시뮬레이션을 통하여 제안한 알고리즘의 성능을 측정하였다. 본 논문에서 제안한 gsLRU 기법을 적용하였을 경우 캐쉬 적중률이 버퍼 캐쉬와 디스크 캐쉬에서 각각 3.8%, 3.6% 증가하였고 디스크 블록 평균 접근 지연시간은 sLRU기법에 비하여 0.11ms 감소되었다.

참 고 문 헌

[1] C. P. Grossman, "Cache-DASD storage design for improving system performance," IBM Systems Journal, vol. 24, no. 3/4, pp. 316-334. 1985.

[2] P. J. Jalics and D. R. McIntyre, "Caching and Other Disk Access Avoidance Techniques on Personal Computers," Communications of the ACM, vol. 32, no. 2, pp. 246-255, Feb. 1989.

[3] M. J Bach, "The Design of the UNIX Operating System," Prentice-Hall, Englewood Cliffs, NJ. 1986.

[4] Data General Corporation, "Configuring and Managing a CLARiiON Disk-Array Storage System," 1994.

[5] ETRI, "TICOM-91: Hardware Functional Specification Rev. 1.0," 1989.

[6] B. McNutt, "I/O subsystem configurations for ESA: New roles for processor storage," IBM Systems Journal, vol. 32, no. 2, pp. 252-264, 1993.

[7] D. M. Muntz and P. Honeyman, "Multi-level Caching in Distributed File Systems," Proceedings of the 1992 Winter USENIX Conference, pp. 305-313, 1992.

[8] J. T. Robinson and N. V. Devarakonda, "Data Cache Management Using Frequency-Based Replacement," Proceedings of the ACM SIGMETRICS Conference, pp. 134-142, 1990.

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

[10] P. Cao, E. W. Felton, and K. Li, "Application-Controlled File Caching Policies." Proceedings of the Summer 1994 USENIX Conference, pp. 171-182, Jun. 1994.

[11] M. D. Dahlin, R. Y. Wang. T. E. Anderson, and D. A. Patterson, "Cooperative Caching: Using Remote Client Memory to Improve File System Performance," First Symposium on Operating Systems Design and Implementation, pp. 267-280, Nov. 1994.

[12] R. Karedla, J. S. Love, and B. G. Wherry, "Caching Strategies to Improve Disk System Performance," Computer, vol. 27, no. 3, pp. 38-46, March 1994.

[13] D. H. Lee, S. H. Noh, S. L. Min, and Y. K. Cho, "Efficient Cache Management Schemes for Reducing Duplication Caching between Buffer and Disk Caches," Journal of KISS(A), vol. 22, no. 10, october 1995.

[14] M. N. Nelson, B. B. Welch, and J. K. Ousterhout, "Caching in the Sprite Network File System," ACM Trans. on Computer Systems, vol. 6, no. 1, pp. 134-154, 1988.

류갑상

1983년 전남대학교 전산통계학과(이학사)
1985년 전남대 전산통계학과(이학석사)
1998년 고려대 컴퓨터학과(박사수료)
1991년 정보처리기술사 취득
1985-1996년 한국기계연구원 선임연구원
1997년-현재 한국정보처리학회 편집위원
2000년-현재 (기)유신하이테크 대표
2001년-현재 (사)한국정보통신기술사회 이사
1996년-현재 동신대학교 교수
관심분야: 멀티미디어응용, 전자상거래, 공장자동화

정수목

1984년 경북대학교 전자공학과(공학)
1986년 경북대학교 대학원 전자공학과(공학석사)
2001년 고려대 대학원 컴퓨터학과(이학박사)
1986년-1991년 LG정보통신 연구소 연구원
1988년-현재 삼육대학교 컴퓨터과학과 부교수
관심분야: 멀티미디어, Video coding, Computer architecture