이기종 저장 장치 환경을 위한 버퍼 캐시 관리 기법

(An Efficient Buffer Cache Management Scheme for Heterogeneous Storage Environments)

이 세 환 † 고 건 †† 반 효 경 †††
(Sehwan Lee) (Kern Koh) (Hyokyung Bahn)

요 약 플래시 메모리는 하드 디스크에 비해 크기가 작고 물리적 충격에 강하며 전력 소모가 적은 점 등 많은 장점을 가지고 있지만 아직까지 단위 공간당 가격이 높아 하드 디스크를 전면 대체하기는 어려운 실정이다. 최근 노트북 컴퓨터 등 일부 모바일 장치에서는 하드 디스크와 플래시 메모리를 함께 사용하여 두 매체의 장점을 극대화하려는 시도가 이루어지고 있다. 하지만 기존 운영체제는 이기종 저장 장치 환경이 아닌 단일 저장 장치 환경에 최적화되어 이러한 장점을 충분히 살리지 못하고 있다. 본 논문에서는 이를 해결하기 위해 세 가지 기법을 이용하는 새로운 버퍼 캐시 관리 기법을 제안한다. 첫째, 입출력 접근 패턴을 탐지하고 블록의 저장 위치 별 성능 특성을 분석한 후 동적 한계 효용에 근거하여 버퍼 캐시 공간을 할당한다. 둘째, 입출력 접근 패턴과 저장 장치 특성에 따라서 선택적으로 선반입 기법을 적용한다. 셋째, 버퍼 캐시에서 저장 장치로 쫓겨날 때 해당 블록의 접근 패턴에 따라 하드 디스크와 플래시 메모리 중 더 적합한 매체를 결정하고 그 매체에 블록이 저장되도록 한다. 제안하는 기법들을 트레이스 기반 시뮬레이션으로 검증한 결과 기존 기법에 비해 버퍼 캐시 적중률은 29.9%, 총 실행시간은 49.5% 향상되었다.

키워드 : 하드 디스크, 플래시 메모리, 모바일 컴퓨터, 버퍼 캐시 교체 알고리즘

Abstract Flash memory has many good features such as small size, shock-resistance, and low power consumption, but the cost of flash memory is still high to substitute for hard disk entirely. Recently, some mobile devices, such as laptops, attempt to use both flash memory and hard disk together for taking advantages of merits of them. However, existing OSs (Operating Systems) are not optimized to use the heterogeneous storage media. This paper presents a new buffer cache management scheme. First, we allocate buffer cache space according to access patterns of block references and the characteristics of storage media. Second, we prefetch data blocks selectively according to the location of them and access patterns of them. Third, we moves destaged data from buffer cache to hard disk or flash memory considering the access patterns of block references. Trace-driven simulation shows that the proposed schemes enhance the buffer cache hit ratio by up to 29.9% and reduce the total I/O elapsed time by up to 49.5%.

Key words : hard disk, flash memory, mobile computer, buffer cache replacement algorithm

· 본 논문은 2009 정부(교육과학기술부)의 재원으로 한국연구재단의 지원을 받아 수행된 연구임(과제책임자 고건: No.2009-0076335)
(과제책임자 반효경: No.2009-0077659)
· 이 논문은 2010 동계 워크샵에서 '이기종 저장장치 환경을 위한 버퍼캐쉬 관리 기법'의 제목으로 발표된 논문을 확장한 것임

† 학생회원 : 서울대학교 컴퓨터공학부
sehwan.lee@oslab.snu.ac.kr
†† 종신회원 : 서울대학교 컴퓨터공학부 교수
kernkoh@oslab.snu.ac.kr
††† 종신회원 : 이화여자대학교 컴퓨터공학전공 교수
bahn@ewha.ac.kr
(Corresponding author임)

논문접수 : 2010년 4월 24일
심사완료 : 2010년 6월 21일

정보과학회논문지: 시스템 및 이론 제37권 제5호(2010.10)

1. 서 론

플래시 메모리는 가볍고 물리적 충격에 강하며 전력 소모가 적고 접근 속도가 빨라 모바일 기기용 저장 장치로 널리 사용되고 있다. 최근에는 플래시 메모리의 집적도 기술 향상으로 모바일 기기뿐 아니라 대용량 서버, 데스크톱 PC 등의 환경에서도 플래시 메모리로 구성된 SSD(Solid State Disks)를 사용하려는 시도가 늘고 있는 추세이다. 하지만 플래시 메모리만으로 저장 장치를 구성하기에는 단위 공간당 가격이 높고 쓰기 연산의 성능이 균일하지 않은 약점이 있어 하드 디스크와 플래시 메모리를 함께 사용하여 서로의 장점을 취하려는 시도가 이루어지고 있다[1].

기존 운영체제는 하드 디스크를 저장 장치로 사용하는 것에 최적화되었다. 이러한 최적화는 하드 디스크가 저장 장치로 사용되는 환경에서는 적합한 접근이지만 이기종 저장 장치가 함께 사용되는 환경에서는 성능 상의 장점을 충분히 이끌어내지 못하는 접근이 되고 있다. 최근에는 플래시 메모리를 저장 장치로 사용하는 경우 운영체제의 각 계층에서 이루어져야 할 변화에 대해 활발한 연구가 진행되고 있지만 하드 디스크와 함께 사용되는 이기종 저장 장치 환경에 관한 연구는 아직 충분히 이루어지지 못하고 있다.

본 연구는 플래시 메모리와 하드 디스크를 함께 저장 장치로 사용하는 환경에서 데이터 이동을 지원하는 새로운 버퍼 캐시 관리 기법을 제안한다. 제안하는 기법은 세 가지 큰 특징을 가지고 있다. 첫째, 데이터 블록의 입출력 패턴과 저장 장치의 특성을 분석하고 이를 기반으로 블록의 한계효용(marginal gain)을 계산한 뒤 버퍼 캐시 공간을 한계효용에 근거하여 동적으로 할당한다. 둘째, 제안하는 기법은 선택적으로 데이터 블록을 선반입 한다. 단일 저장 장치 환경에서는 선반입 기법이 항상 효과적이지만 이기종 저장 장치 환경에서는 경우에 따라 선반입으로 인해 성능이 저하될 수 있기 때문에 이를 방지하기 위해 본 논문에서는 데이터 블록의 인출 비용을 고려하여 선반입을 선택적으로 수행한다. 셋째, 교체 알고리즘에 의해 버퍼 캐시에서 저장 장치로 쫓겨나는 블록을 참조 패턴에 따라 하드 디스크와 플래시 메모리 중 더 적합한 곳으로 위치시킨다. 제안하는 기법은 순차 참조 패턴을 보이는 데이터를 하드 디스크에, 임의 참조 패턴을 보이는 데이터를 플래시 메모리에 저장하여 미래 참조가 효과적으로 처리될 수 있도록 한다.

이상의 세가지 개선을 통하여 기존 기법 중 참조 패턴 별 한계효용을 활용하는 대표적인 기법인 UBM에 비해 버퍼 캐시 적중률은 29.9%, 총 실행시간은 49.5% 개선됨을 보인다.

2. 관련 연구

2.1 하드 디스크를 위한 버퍼 캐시 관리 기법

하드 디스크를 위한 버퍼 캐시 교체 알고리즘은 캐시 적중률을 높여 디스크 입출력 횟수를 줄이는 것을 목표로 하고 있다. LRU 알고리즘이 이러한 환경을 위한 대표적인 알고리즘이라 할 수 있다. LRU 알고리즘은 블록 참조의 최근성(recency)을 조사한 뒤, 가장 오래 전에 참조된 블록을 교체한다. 이에 대한 효율적인 구현을 위해 LRU는 캐싱된 블록들을 리스트로 관리하며 블록 참조가 일어날 때마다 참조된 블록을 리스트의 끝으로 이동시켜 접근 시간 순서로 리스트가 유지되도록 한다. LRU는 모든 참조 시점에 리스트 조작이 필요하다는 부담이 있어 캐시 미스가 발생한 경우에만 리스트 조작을 수행하면 되는 CLOCK 알고리즘을 통해 LRU를 근사시키는 방식이 가상메모리 시스템 등에서 널리 활용되고 있다[2].

LFU 알고리즘은 블록 참조의 최근성 대신 참조 빈도(frequency)를 바탕으로 캐시 교체를 수행한다. 참조 빈도와 최근성을 함께 사용하는 알고리즘도 널리 연구돼 왔다. 그 대표적인 알고리즘으로 LRFU[3]가 있다. LRFU는 각 캐시 블록마다 최근성과 참조빈도를 함께 고려하는 평가값을 계산하고 이 값이 가장 작은 블록을 캐시에서 교체한다.

최근성과 참조 빈도가 아닌 데이터의 접근 패턴을 활용하는 연구도 활발히 이루어졌다. UBM[4]의 경우 블록 참조를 순차 참조, 임의 참조, 반복 참조의 세 가지로 구분한다. 순차 참조는 블록들이 순차적으로 한 번 참조 된 후 다시 참조 되지 않는 패턴이고, 반복 참조는 순차 참조가 일정 주기로 반복되는 패턴이고, 임의 참조는 순차 참조에도 반복 참조에도 해당되지 않는 패턴이다. 각 패턴 별로 버퍼 캐시 공간을 할당한 뒤 한계 효용을 계산하여 할당된 크기를 조절하고, 패턴 별로 페이지 교체 알고리즘을 별도로 적용한다.

2.2 플래시 메모리를 위한 버퍼 캐시 관리

플래시 메모리는 하드 디스크와는 다른 고유한 특징을 가지고 있다. 첫째, 쓰기 연산을 수행하기 위해서는 해당 위치에 대한 삭제 연산을 먼저 수행해야 한다. 둘째, 삭제 연산의 단위인 블록과 읽기/쓰기 연산의 단위인 페이지의 크기가 상이하다. 셋째, 읽기 연산과 쓰기 연산의 속도가 다르다. 넷째, 삭제 연산의 횟수가 제한되어 있어 특정 횟수 이상 삭제 연산이 수행된 블록은 더 이상 사용할 수 없다. 플래시 메모리를 위한 버퍼 캐시 관리 기법은 이러한 플래시 메모리의 특성을 보완해 줄 수 있어야 한다.

CFLRU[5]는 플래시 메모리의 읽기/쓰기/삭제 연산의

속도가 상이하다는 점을 활용한 페이지 교체기법이다. CFLRU는 기본적으로 LRU 리스트를 통해 캐싱된 블록들을 관리한다. CFLRU는 LRU 리스트를 working region과 clean-first region의 두 영역으로 나누어서 관리한다. 빈 페이지가 필요한 경우 CFLRU는 먼저 clean-first region에서 클린 페이지를 쫓아낸다. 이는 더티 페이지를 쫓아낼 경우 수반되는 플래시 메모리 쓰기 연산을 최대한 줄이기 위한 방법이다.

LRU-WSR[6]은 CFLRU와 같이 더티 페이지에 우선권을 주는 버퍼 캐시 교체 알고리즘이다. LRU 리스트로 캐싱된 페이지들을 관리하면서 더티 페이지가 LRU 위치에 도달한 경우 한번 더 기회를 주어 MRU 위치로 이동시켜 더티 페이지가 쫓겨날 경우 발생하는 쓰기 연산을 최대한 지연시킨다.

FAB[7]은 플래시 메모리를 위하여 DRAM 버퍼를 사용하는 알고리즘이다. 모든 쓰기 요청은 DRAM 버퍼에서 처리가 되고 DRAM 버퍼에서 플래시 메모리로 옮겨질 때는 플래시 메모리의 블록 별로 페이지를 그룹으로 분류하여 기록하는 방식을 사용한다. 이를 통하여 쓰기/삭제 연산의 수를 효과적으로 줄일 수 있다.

BPLRU[8]는 임의 쓰기 요청의 성능을 향상시키기 위한 쓰기 버퍼 관리 알고리즘이다. BPLRU는 DRAM 버퍼를 두고 플래시 메모리로 가는 쓰기 요청을 관리하여 동일한 블록에 있는 페이지들을 한꺼번에 기록함으로써 플래시 쓰기/삭제 연산의 수를 줄인다. 이를 위해 페이지 패딩(page padding)과 LRU 보상(LRU compensation)이 사용된다. 페이지 패딩은 DRAM에서 부분적으로 채워져 있는 블록이 플래시에 기록되어야 할 경우 비워져 있는 페이지를 플래시 메모리에서 읽어온 후 채워서 한 블록으로 만들어주는 기법이다. LRU 보상 기법은 DRAM에서 플래시 메모리로 기록할 때 전부 채워져 있는 블록을 먼저 기록하는 기법이다.

3. 제안하는 기법

3.1 개략적인 구조

본 논문은 하드 디스크와 플래시 메모리로 구성된 이기종 저장 장치 환경을 위한 효율적인 버퍼 캐시 관리 기법을 제안한다. 제안하는 기법의 구조는 그림 1과 같다. 버퍼 캐시를 하드 디스크를 위한 공간과 플래시 메모리를 위한 공간으로 나누어 관리하고 각 공간을 접근 패턴 별로 순차 참조, 반복 참조, 임의 참조로 다시 나누어 관리한다.

본 논문에서는 I/O 성능을 향상시키기 위해 세 가지 기법을 제안하고 있다. 첫째, 각 공간의 크기를 관리하기 위해 한계 효용치를 사용하며 이에 대한 계산은 접근 패턴과 입출력 비용을 함께 고려하여 동적으로 수행한다. 둘째, 선택적으로 선반입 기법을 적용한다. 이기종 저장 장치가 사용되는 경우 버퍼 적중률이 높아진다 하더라도 입출력 비용이 높은 장치에서 캐싱된 데이터가 선반입을 위해 쫓겨날 경우 전체 소요시간이 늘어나는 경우가 발생할 수 있다. 따라서, 본 논문은 이와 같은 상황을 고려하여 소요시간을 줄일 수 있는 경우에만 선반입을 선택적으로 수행한다. 셋째, 캐싱된 블록을 참조 패턴 별로 분류하여 필요한 경우 캐시 교체 시 해당 블록의 저장 위치를 변경할 수 있도록 한다. 이는 대부분의 파일 접근이 과거에 어루어진 패턴과 동일한 패턴으로 이루어지기 때문에 적절한 저장 위치를 결정하여 후속 참조시 입출력 비용을 줄일 수 있도록 하기 위함이

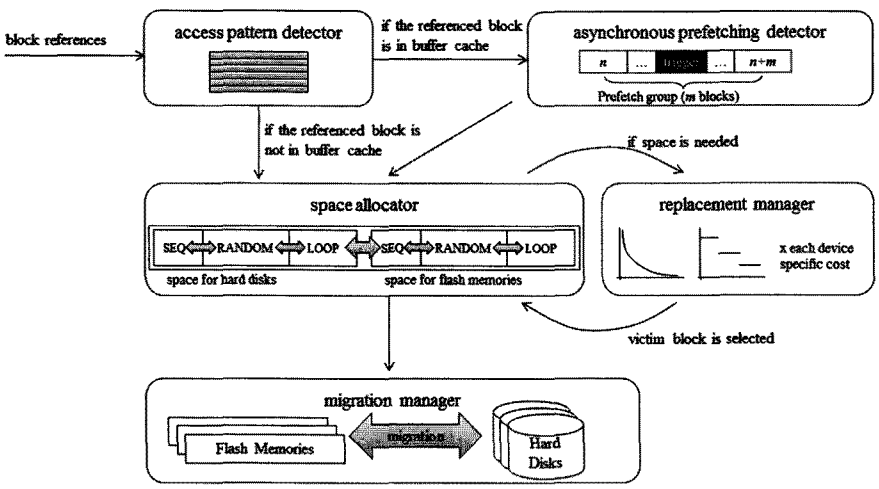


그림 1 제안하는 알고리즘의 개략적 구조

다. 플래시 메모리는 하드 디스크와 비교하여 임의 접근에 비교 우위가 있고 하드 디스크는 플래시 메모리와 비교하여 순차 접근에 비교 우위가 있다. 따라서 버퍼 캐시에서 교체될 때 순차 패턴을 보이는 데이터 블록을 하드 디스크에 저장하고 임의 패턴을 보이는 데이터를 플래시 메모리에 저장하도록 한다.

3.2 동적 한계 효용 계산 기법

한계 효용으로 공간의 크기를 결정하는 것은 UBM 기법에서 이미 제안된 바 있다[4]. UBM 기법은 하드 디스크를 위한 버퍼 캐시 관리 기법으로 버퍼 캐시 적중률만 고려하여 한계 효용을 계산한다. 즉, 캐시 크기가 n이라 할 때 $MG(n) = Hit(n) - Hit(n-1)$ 으로 정의하고 각 참조 패턴 별 한계 효용을 다음과 같이 정의하였다. 순차 참조 패턴의 경우 다시 참조가 일어나지 않기 때문에 $MG_{seq}(n) = 0$, 임의 참조 패턴의 경우 Belady의 lifetime 함수에 의해 $MG_{rand}(n) = c(n-1)^{-k} - cn^{-k}$ (c, k는 조절 변수), 반복 참조 패턴의 경우 $MG(n) = 1/p$ (p는 순환 주기)로 정의된다[9].

이러한 정의는 하드 디스크만을 저장 장치로 사용하는 환경에는 적합하지만 이기종 저장 장치가 사용되는 환경에는 적합하지 않다. 왜냐하면 이기종 저장 장치의 경우 각 장치 별로 입출력 비용이 다르기 때문에 단순한 버퍼 캐시 적중률로는 합리적인 한계효용을 계산할 수 없기 때문이다. 본 논문에서는 한계효용 함수를 각 장치의 입출력 비용과 블록 소비율을 고려하여 새롭게 정의하였다.

첫째, 각 장치의 입출력 비용을 고려하기 위하여 각 장치별 입출력에 소요되는 시간을 고려하였다. 표 1과 같이 플래시 메모리와 하드디스크는 임의 참조 패턴과 순차 참조 패턴의 경우 상이한 소요시간을 갖게 된다. 이를 고려하기 위하여 기존 UBM 알고리즘을 확장하여 해당 블록이 어떤 장치에서 올라왔는지를 고려하도록 한다.

둘째, 순차 패턴의 한계 효용값을 새롭게 정의하였다. UBM 에서 순차 패턴의 한계 효용 값은 0 이지만 제안하는 기법은 순차 패턴의 한계 효용 값을 계산하기 위하여 블록 소비율을 정의하였다. 블록 소비율은 다음 선반입이 일어나거나 버퍼 캐시 교체가 일어날 때까지 선반입된 블록 중에서 적중한 블록의 수로 정의된다. 이 값이 1이라고 하면 선반입된 블록이 전부 적중한 것이다. 이 경우 보다 많은 양의 블록을 선반입했다면 더 많은 블록이 적중할 수 있기 때문에 선반입 크기를 키워야 한다. 블록 소비율이 1 보다 작은 경우 두 가지로 나눌 수 있는다. 첫째, 블록 소비율이 매우 작은 경우 선반입된 블록이 적중되기 전에 대부분이 쫓겨난 경우이다. 이 경우 선반입 크기를 줄이고 임의 참조나 반복 참조의 크기를 키우면 더 높은 버퍼 캐시 적중률을 얻을 수 있기 때문에 선반입 크기를 줄여야 한다. 둘째, 블록 소비율이 앞의 두 가지 경우의 중간 크기라면 순차 패턴의 한계 효용 값이 충분히 큰 경우이다. 이 경우 블록 소비율을 바탕으로 다른 패턴들과 한계 효용값을 비교하여 선반입 크기를 키워야 한다.

표 1 1 Block을 읽기 위한 소요 시간

	임의 참조 (ms)	순차 참조 (ms)
플래시 메모리	0.5	0.5
하드 디스크	5.6	0.1

(1 Block:8 KB)

3.3 선택적 선반입 기법

단일한 저장 장치를 사용하는 환경의 경우 과도한 선반입으로 미처 사용되기도 전에 선반입된 블록이 쫓겨나는 특수한 상황이 아니라면 선반입 기법이 항상 효과적이다. 하지만 이기종 저장 장치를 사용하는 환경에서는 버퍼 캐시 적중률을 높이는 것에 최적화된 기존의 선반입 기법이 저장 장치별 입출력 비용 차이로 인해 성능을 오히려 저하시킬 수 있다. 예를 들어 플래시 메모리에 저장된 순차 참조 데이터에 대해 선반입을 하는 경우 캐시 적중률을 높일 수는 있지만 선반입의 공간 마련을 위해 쫓겨난 데이터가 하드 디스크로부터 캐싱된 블록이라면 향후 재참조시 많은 입출력 시간이 소요되어 전체적인 입출력 성능은 더 나빠질 수 있다.

선반입 기법은 그림 2에서 볼 수 있듯이 동기식 선반입 기법과 비동기식 선반입 기법으로 나눌 수 있다. 동기식 선반입 기법은 버퍼캐시에 원하는 블록이 없을 때 선반입을 시작하는 방식이다. 비동기식 선반입 기법은 트리거 블록을 설정하여 트리거 블록을 요청할 때 선반입 기법을 시작하는 것이다. 트리거 블록이란 아직 선반입된 순차 참조 블록들이 모두 소진되기 전에 후속 블록들을 미리 읽어와 선반입 크기 제한으로 중도에 발생할 미스를 방지하기 위한 블록으로 선반입 블록들 중 후반부에 선반입 재시작점으로 설정된 블록을 뜻한다.

본 논문에서는 요청이 발생한 블록을 처리할 때 다음과 같이 판단하여 선택적 선반입 기법을 수행한다. 첫

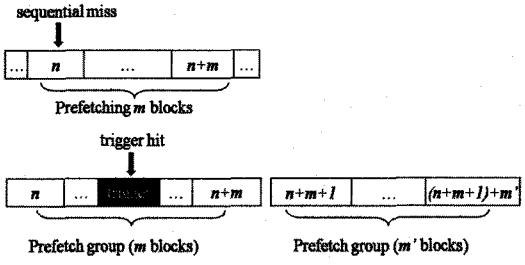


그림 2 동기식 선반입 기법과 비동기식 선반입 기법

째, 요청된 블록의 참조 패턴이 순차 참조 또는 반복 참조인지 확인한다. 임의 참조의 경우 선반입에 의한 효과가 떨어지기 때문에 고려하지 않는다. 둘째, 현재 요청된 블록과 직전에 요청된 블록 사이의 거리가 시퀀스로 묶을 수 있는지 확인한다. 선반입 기법은 물리적으로 인접한 블록에 요청이 올 때 효과가 있다. 만일 두 요청 사이에 거리가 멀다면 그 요청들은 하나의 블록으로 처리하는 것이 아니라 두개의 임의 참조 요청으로 판단해야 한다. 셋째, 요청된 블록이 하드 디스크에 속한 블록인지 확인한다. 플래시 메모리의 경우 입출력시 검색시간이 없기 때문에 선반입으로 인한 효과가 미미하다. 이러한 세 조건을 모두 만족시키면서 요청한 블록이 트리거 블록(trigger block)일 경우 선반입 기법을 수행하고 선반입 크기가 최대 크기에 도달하지 않은 경우 이를 증가시킨다.

3.4 참조 패턴을 고려한 데이터 이동 기법

버퍼 캐시에 빈 공간이 없을 때 캐시에 없는 새로운 블록에 대한 요청이 오면 기존에 캐싱된 블록 중 하나를 쫓아내고 그 자리에 새로운 블록을 저장하게 된다. 이때 희생된 블록은 통상적으로 그 블록이 원래 보관돼 있던 저장 장치로 쫓겨나게 된다. 하지만 본 논문에서는 참조 패턴을 고려하여 어떤 저장 장치로 쫓아낼지를 판단한다.

한 번 참조 패턴이 결정된 블록의 경우 다시 참조가 일어날 때 동일한 패턴으로 참조될 확률이 높다. 이를 이용하기 위하여서 본 논문에서는 임의 참조 패턴으로 판단된 블록이 쫓겨날 때에는 플래시 메모리로, 순차 참조 패턴으로 판단된 블록이 쫓겨날 때에는 하드 디스크로 저장하도록 한다. 이때 필요한 플래시 메모리와 하드 디스크의 공간은 표 2와 같다. 비록 플래시 메모리에 저장되는 임의 참조의 경우 그 시퀀스가 많지만 해당 시퀀스의 길이는 순차 참조와 비교하면 훨씬 짧은 것을 볼 수 있다. 따라서 적은 크기의 플래시 메모리를 이용하여 미래에 있을 참조 비용을 효과적으로 절약할 수 있다.

데이터를 이동할 때 어떤 데이터를 이동하는지는 이동 기법의 정책에 따라 다르다. 첫째, 참조 패턴이 파악된 블록이 저장장치로 쓰일 때, 해당하는 블록만 기록하는 방식이다. 이 경우 데이터 이동에 의한 이득이 작다는 단점이 있지만 예측이 틀렸을 경우 불이익이 적다. 둘째, 참조 패턴이 파악된 블록이 저장장치로 쓰일 때, 해당하는 블록이 포함된 파일 전체를 옮기는 방식이다. 이것은 적극적으로 데이터 이동을 함으로써 참조비용을 감소시키지만 예측이 틀린 경우, 혹은 블록의 참조 패턴이 바뀐 경우 다시 옮기는데 드는 부담이 크다. 따라서 본 기법에서는 첫번째 방법을 사용하도록 한다.

표 2 저장 장치 별 시퀀스와 평균 크기

	시퀀스 수	평균 길이 (KB)
플래시 메모리	1194	11.96
하드 디스크	28	1356.00

4. 성능 평가

4.1 실험 환경과 데이터

본 장에서는 트레이스 기반 시뮬레이션을 통해 제안하는 기법의 성능을 UBM 기법과 점진적으로 비교한다. 또한, 다양한 워크로드와 캐시 크기 하에서 LRU, UBM[1]과 성능 비교를 통해 제안한 기법의 효과를 검증한다.

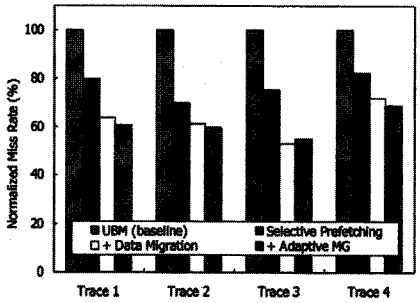
실험에 사용된 트레이스들은 UBM의 성능측정을 위해서 사용된 트레이스로써 FreeBSD에서 여러 프로그램들을 실행시켜서 수집한 것이다. 각 트레이스는 아래 설명된 프로그램들을 동시에 실행시킴으로써 얻어졌다.

- cs: cscope로써 소스 코드를 검색하는 참조를 생성하는 프로그램이다. 약 9MB의 c 소스 코드를 입력으로 받아 참조를 생성하였고 임의 참조 패턴과 반복 참조 패턴을 보인다.
- cpp: GNU C 전처리기이다. 입력으로 약 11MB의 c 소스코드를 받아 처리하였고 순차 참조 패턴과 임의 참조 패턴을 보인다.
- ps: postgress로써 관계형 데이터베이스 프로그램이다. 약 24MB 정도의 각종 쿼리를 처리하였고 순차 참조 패턴, 임의 참조 패턴, 반복 참조 패턴을 모두 보인다.
- sdet: SPEC SDET 벤치마크로써 8명의 사용자가 사용중인 스크립트를 처리하였다.

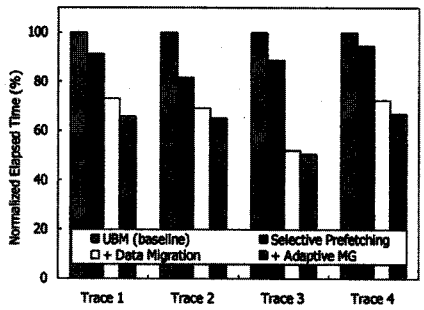
4.2 버퍼 적중률과 실행 시간 비교

그림 3(a)와 3(b)는 버퍼 캐시의 크기를 1,000개의 블록으로 고정시켰을 때 각 트레이스별 버퍼 캐시 적중률과 실행시간을 보여주고 있다. 기존 UBM과 비교할 때 선택적 선반입 기법은 버퍼 캐시 실패율을 17.8-30.1%, 실행 시간을 5.5-18.5%까지 향상시키는 것을 확인할 수 있었다. 선택적 선반입 기법과 패턴에 따른 데이터 이동 기법을 함께 적용한 결과를 보면 기존 UBM과 비교할 때 트레이스에 따라서 버퍼 캐시 실패율은 28.1-46.9%, 실행 시간은 27.2-47.9%의 성능 향상이 있었다. 데이터 이동 기법의 성능만 측정하기 위하여 선택적 선반입 기법과 데이터 이동 기법을 동시에 적용한 실험 결과와 선택적 선반입 기법만 적용한 결과를 비교하면 버퍼 캐시 실패율은 3.6-8.1%, 실행 시간은 15.0-41.0%의 성능 향상이 있었다. 데이터 이동 기법은 버퍼 캐시 적중률을 높이는 것보다 실행 시간을 줄이는 것에 초점이 맞춰져 있기에 버퍼 캐시 적중률의 향상 폭은 적지만 실행 시간은 큰 폭으로 향상시키는 것을 볼 수 있다.

제안하는 기법을 전부 적용시킨 결과를 UBM과 비교하면 버퍼 캐시 실패율은 31.3-45.0%, 실행 시간은 33.0-49.5%의 성능 향상이 있었다. 제안하는 세 가지 기



(a) UBM에 정규화된 버퍼 캐시 실패율



(b) UBM에 정규화된 실행 시간

그림 3 UBM 대비 성능 개선(버퍼캐시의 크기: 1000, Trace 1 = cs+cpp+ps, Trace 2 = cs+cpp+ps+sdet, Trace 3 = sdet-cs+cpp+ps+sdet, Trace 4 = mixed trace)

법 중에서 버퍼 캐시 실패율을 가장 많이 향상 시킨 것은 선택적 선반입 기법이고 실행 시간을 가장 많이 향상 시킨 기법은 패턴에 따른 데이터 이동 기법이었다. 패턴에 따라 데이터를 이동하면 비록 버퍼 캐시 실패율 향상은 적었으나 데이터를 저장장치에서 읽는 시간을 크게 절약함으로써 전체 실행시간을 크게 줄일 수 있었다.

4.3 다른 기법들과 성능 비교

동일한 저장 장치를 사용하는 시스템에서는 버퍼 캐시 적중률을 높이면 가장 빠른 실행 시간을 얻을 수 있기 때문에 지금까지 버퍼 캐시 알고리즘은 적중률을 높이는 것을 목표로 하였다. 하지만 이기종 저장 장치를 사용하는 환경의 경우, 단순하게 버퍼 캐시 적중률을 높이는 것으로는 캐싱 알고리즘의 성능을 정확히 측정할 수 없다. 동일한 블록이라고 할지라도 어떤 저장 장치에서 반입하는가에 따라 실행 시간이 달라지기 때문이다. 그림 4와 5는 버퍼 캐시의 크기를 변화시키면서 각 트레이스 별로 제안하는 기법의 성능을 LRU 기법, UBM 기법과 함께 비교하였다. 그림에서 볼 수 있듯이 제안하는 알고리즘은 버퍼 캐시 크기와 트레이스에 상관 없이 가장 적은 수행시간을 기록하였다.

UBM과 LRU의 경우 적중률과 수행 시간이 버퍼 캐시의 크기에 비례하는 것을 볼 수 있다. 하지만 제안하는 기법은 버퍼 캐시의 크기와 항상 일치하는 것은 아니다. 때로는 버퍼 캐시의 크기가 증가함에도 불구하고 적중률은 감소하는 경우도 있다. 하지만 전체 수행시간은 적중률과는 달리 버퍼 캐시의 크기가 증가함에 따라서 일정하게 감소하는 것을 볼 수 있다. 이러한 결과는 제안하는 기법은 기존 알고리즘들이 버퍼 캐시 적중률을 극대화하는 것을 목표로 했던 것과는 달리 전체 실행 시간을 최소화하는 것을 목표로 버퍼 캐시를 관리하기 때문이다.

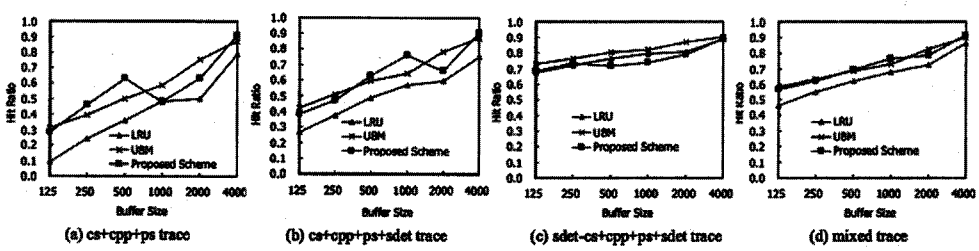


그림 4 버퍼 캐시 적중률에 따른 성능 비교

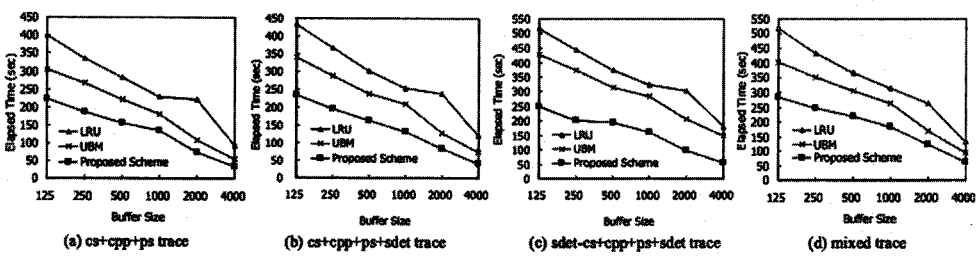


그림 5 전체 수행 시간에 따른 성능 비교

5. 결 론

본 논문에서는 하드 디스크와 플래시 메모리가 함께 사용되는 이기종 저장 장치 환경에서 장치 별 입출력 비용 차이와 블록의 참조 패턴을 고려한 버퍼 캐시 교체 알고리즘을 제안하였다. 제안된 기법은 세가지 요소로 구성된다. 첫째, 입출력 비용을 탐지하고 저장 장치 별 특성을 고려하여 한계 효용을 동적으로 계산한다. 둘째, 저장장치와 블록 참조 패턴을 고려하여 선택적으로 데이터를 선반입한다. 셋째, 참조 패턴을 고려하여 쫓겨나는 데이터 블록을 적합한 저장장치에 저장하도록 한다. 또한 본 논문에서는 버퍼 캐시의 성능을 측정할 때 기존 환경에서 적합한 지표였던 캐시 적중률이 아닌 이기종 저장장치 환경에 적합한 총 실행 시간에 초점을 맞춰야 함을 제시하였다. 트레이스 기반 시뮬레이션 실험을 통해 본 논문에서 제안한 세가지 기법을 실행 시간 관점에서 조사한 결과 UBM에 비해 버퍼 캐시 적중률은 29.9% 증가하였고, 실행 시간은 49.5% 감소되는 것을 확인할 수 있었다.

다음 연구로는 제안한 기법들을 실제 리눅스 상에서 구현하여 성능 평가를 하겠다. 플래시 메모리와 하드디스크간의 데이터 이동을 지원하기 위하여 LVM 방식 위에 구현하여 성능을 평가하겠다.

참 고 문 헌

[1] F. Chen, S. Jiang, and X. Zhang, "SmartSaver: Turning Flash Drive into a Disk Energy Saver for Mobile Computers," in *ISLPED '06: Proceedings of the 2006 International Symposium on Low Power Electronics and Design*, New York, NY, USA: ACM, pp.412-417, 2006.

[2] S. Jiang, F. Chen, and X. Zhang, "Clock-pro: An Effective Improvement of the Clock Replacement," in *ATEC '05: Proceedings of the Annual Conference on USENIX Annual Technical Conference. Berkeley, CA, USA: USENIX Association*, pp.323-336, 2005.

[3] D. Lee, J. Choi, J.-H. Kim, S. Noh, S. L. Min, Y. Cho, and C. S. Kim, "LRFU: A Spectrum of Policies that Subsumes the Least Recently Used and Least Frequently Used Policies," *Computers, IEEE Transaction on*, vol.50, no.12, pp.1352-1361, Dec 2001.

[4] J.M. Kim, J. Choi, J. Kim, S.H. Noh, S.L. Min, Y. Cho, and C.S. Kim, "A Low-overhead High-Performance Unified Buffer Management Scheme that Exploits Sequential and Looping References," In *Proceedings of the 4th conference on Symposium on Operating System Design and Implementation*, 2000.

[5] S.-y. Park, D. Jung, J.-u. Kang, J.-s. Kim, and J. Lee, "CFLRU: A Replacement Algorithm for Flash Memory," in *Proceedings of the 2006 International Conference on Compilers, Architecture and Synthesis for Embedded Systems, ser. CASES '06. New York*, NY, USA: ACM, pp.234-241, 2006.

[6] H. Jung, H. Shim, S. Park, S. Kang, and J. Cha, "LRU-WSR: Integration of LRU and Writes Sequence Reordering for Flash Memory," *Consumer Electronics, IEEE Transactions on*, vol.54, no.3, pp.1215-1223, August 2008.

[7] H. Jo, J.-U. Kang, S.-Y. Park, J.-S. Kim, and J. Lee, "FAB: Flashaware Buffer Management Policy for Portable Media Players," *Consumer Electronics, IEEE Transactions on*, vol.52, no.2, pp.485-493, may 2006.

[8] H. Kim and S. Ahn, "BPLRU: A Buffer Management Scheme for Improving Random Writes in Flash Storage," in *FAST '08: Proceedings of the 6th USENIX Symposium on File and Storage Technologies*, pp.239-252, 2008.

[9] J. Choi, S. Cho, S. Noh, S. Lyul, and Y. Cho, "Analytical Predictionof Buffer Hit Ratios," *Electronics Letters*, vol.36, no.1, pp.10-11, Jan2000.

이 세 환

2004년 서울대학교 컴퓨터 공학부 학사 2004년~현재 서울대학교 컴퓨터 공학부 석·박사 통합과정. 관심분야는 운영체제, 스토리지 시스템, 버퍼캐시, 플래시 메모리 등

고 건

1974년 서울대학교 응용물리학 학사. 1979년 Univ. of Virginia 전산학 석사. 1981년 Univ. of Virginia 전산학 박사. 현재 서울대학교 컴퓨터공학부 교수. 관심분야는 운영체제, 컴퓨터 구조, 컴퓨터 시스템 성능평가, 분산 컴퓨터시스템 등

반 효 경

1997년 서울대학교 계산통계학과 학사. 1999년 서울대학교 전산과학과 석사. 2002년 서울대학교 컴퓨터공학부 박사. 2002년~현재 이화여자대학교 컴퓨터공학과 교수. 관심분야는 운영체제, 스토리지 관리, 저전력 시스템, 메모리 및 캐싱 시스템 등