DOI: 10.3745/KIPSTD.2009.16D.6.825

낸드 플래시 메모리를 위한 CLOCK 알고리즘 기반의 효율적인 버퍼 교체 전략

김 종 선[†] · 손 진 현[††] · 이 동 호[†††]

요 약

최근에 낸드 플래시 메모리는 빠른 접근속도, 저 전력 소모, 높은 내구성 등의 특성으로 인하여 차세대 대용량 저장 매체로 각광 받고 있다. 그러나 디스크 기반의 저장 장치와는 달리 비대칭적인 읽기, 쓰기, 소거 연산의 처리 속도를 가지고 있고 제자리 갱신이 불가능한 특성을 가지고 있다. 따라서 디스크 기반 시스템의 버퍼 교체 정책은 플래시 메모리 기반의 시스템에서 좋은 성능을 보이지 않을 수 있다. 이러한 문제를 해결하기 위해 플래시 메모리의 특성을 고려한 새로운 플래시 메모리 기반의 버퍼 교체 정책이 제안되어 왔다.

본 논문에서는 디스크 기반의 저장 장치에서 우수한 성능을 보인 CLOCK-Pro를 낸드 플래시 메모리의 특성을 고려하여 개선한 CLOCK-NAND를 제안한다. CLOCK-NAND는 CLOCK-Pro의 알고리즘에 기반하며, 추가적으로 페이지 접근 정보를 효율적으로 활용하기 위한 새로운 핫 페이지 변경을 한다. 또한, 더티인 핫 페이지에 대해 콜드 변경 지연 정책을 사용하여 쓰기 연산을 지연하며, 이러한 새로운 정책들로 인하여 낸드 플래시 메모리에서 쓰기 연산 횟수를 효율적으로 줄이는 우수한 성능을 보인다.

키워드 : 플래시 메모리, 버퍼 교체 정책, 클락프로, 핫 페이지, 더티 페이지

An Efficient Buffer Replacement Policy based on CLOCK Algorithm for NAND Flash Memory

Jong-sun Kim[†] · Jin Hyun Son[††] · Dong-Ho Lee[†††]

ABSTRACT

Recently, NAND flash memory has been popular for a next-generation storage device because of its rapid access speed, low-power consumption, shock-resistant. However, unlike a disk based storage device, it has distinct characteristics such as the asymmetric cost of read, write and erase operation, and no in-place update. thus, disk based buffer management policies may not yield a good performance on NAND flash memory based system. To solve this problem, flash-aware buffer management policies which consider the characteristics of NAND flash memory have been proposed.

In this paper, we propose a novel flash-aware buffer replacement policy called CLOCK-NAND that is an enhancement of the well-known disk-based buffer replacement policy, CLOCK-Pro. Although CLOCK-NAND is basically based on the algorithm of CLOCK-Pro, it exploits a new hot page change policy to use page access information efficiently. Also, it delays write operation for dirty and hot page by using cold change delay policy and it yield a good performance on NAND flash memory according to these new policies.

Keywords : Flash Memory, Buffer Management Policy, Clock-Pro, Hot Page, Dirty Page

1. 서 론

오늘날 휴대용 기기의 사용이 일반화되면서 저 전력 소모와 작고, 가벼운 특성을 가진 낸드 플래시 메모리가 휴대용 기기의 주 저장 매체로 활용되고 있다. 또한, 기술의 급격한 발전으로 용량이 현저히 증가하여 PDA나 노트북 등에서 하드 디스크를 대체하는 저장 장치로 사용되고 있다.

일반적으로 하드 디스크와 같은 디스크 기반의 저장 장치는 디스크에 대한 접근 방식이 기계적인 동작으로 구성되어 있다. 따라서 요구되는 데이터가 기록되어 있는 트랙으로 헤드가 접근하는데 많은 탐색 시간을 필요로 한다. 현재까지 디스크 기반의 저장 장치를 사용한 전통적인 운영체제

※ 이 논문 또는 저서는 2006년 정부(교육과학기술부)의 재원으로 한국연구재단(KRF-2006-521-D00457) 및 2007년 정부(교육과학기술부)의 재원으로 한국연구재단의 지원을 받아 수행된 연구임(KRF-2007-313-D00757).
† 정 회 원 : LIG 넥스원 항공연구센터 연구원
†† 종신회원 : 한양대학교 컴퓨터공학과 부교수
††† 정 회 원 : 한양대학교 컴퓨터공학과 부교수(교신저자)
논문접수 : 2009년 4월 7일
수 정 일 : 1차 2009년 8월 18일
심사완료 : 2009년 8월 31일

시스템은 저장 장치의 데이터에 대한 접근 성능을 최적화하기 위해 다양한 방법을 사용해왔다. 대부분의 운영체제는 CPU와 저장 장치의 속도 차이를 극복하기 위해 버퍼에 요구 페이지를 저장하고, 페이지 요구 시 버퍼에 저장된 페이지를 처리하여 성능 향상을 얻고자 하였다. 특히, 버퍼 공간이 모두 사용 중일 때 새로운 페이지의 저장 공간을 만들기 위해 버퍼의 페이지를 교체하는 다양한 버퍼 교체 정책들이 사용되어 왔다. 대표적인 버퍼 교체 정책으로는 가장 오래된 페이지를 교체하는 LRU(Least Recently Used)와 가장 적은 참조수를 가진 페이지를 교체하는 LFU(Least Frequently Used)가 있다[1]. 그러나 이러한 디스크 기반의 정책들은 플래시 메모리의 특성을 고려한 방법이 아니므로 플래시 메모리 기반의 저장 장치에서 좋은 성능을 보이지 않을 수 있다.

디스크 기반의 저장 장치와 구별되는 플래시 메모리의 가장 큰 특성은 제자리 갱신(no inplace update)이 불가능하다는 것이다. 플래시 메모리는 제자리 갱신이 발생했을 때 해당 블록을 소거한 후, 쓰기 연산을 수행(erase-before-write)[2, 3, 5]하는 구조로 되어 있다. 이러한 구조 때문에 플래시 메모리는 읽기·쓰기 연산 외에도 소거 연산을 제공한다. 플래시 메모리의 또 다른 특성은 읽기, 쓰기, 소거 연산 비용이 비대칭이라는 것이다. 특히, 표 1에서와 같이 쓰기 및 소거 연산이 읽기 연산에 비해 비용이 크기 때문에 쓰기 및 소거 연산의 횟수를 줄이는 것이 플래시 메모리 기반 저장 장치의 성능을 향상시키는 핵심 요소가 된다[6, 7].

본 논문은 플래시 메모리를 저장 장치로 사용하는 임베디드 시스템의 버퍼 교체 정책에 초점을 맞추고 있다. 디스크 기반의 저장 장치에서 보편적으로 사용되는 버퍼 교체 정책들은 버퍼 적중률(Hit ratio)에 초점을 맞춘 방법이다. 그러나 플래시 메모리 기반의 저장 장치에서는 버퍼 적중률뿐만 아니라 비대칭적인 연산 비용으로 인해 페이지 교체 비용도 고려해야 한다.

일반적으로 페이지 교체 비용은 버퍼의 더티(Dirty) 페이지가 교체되면 발생한다. 더티 페이지는 버퍼에서 수정이 일어난 페이지로 교체 시에 저장 장치에 쓰기 연산을 발생시킨다. 반면에, 클린(Clean) 페이지는 수정이 일어나지 않은 페이지로 교체 시에 단순히 드롭(drop)된다. 앞서 언급했듯이 플래시 메모리에 대한 쓰기 연산은 읽기 연산보다 큰 비용을 필요로 하며 제자리 갱신이 불가능하여 추가적인 고비용의 소거 연산을 발생시킬 수 있다. 따라서 플래시 메모리의 비대칭 연산 비용의 특성을 고려하지 않은 버퍼 교체 정책은 플래시 메모리 기반의 저장 장치에서 좋은 성능을 기대할 수 없다. 이러한 문제를 해결하기 위해 플래시 메모리의 특성을 고려한 CFLRU[8]와 LIRS-WSR[9]이 제안되었다. 그러나 이 정책들은 페이지의 참조 정보를 효율적으로 이용하지 못하여 버퍼 적중률이 저하되거나, 리스트 간 중복된 엔트리로 인해 공간 활용이 비효율적일 수 있다.

〈표 1〉 플래시 메모리의 연산 속도[4, 5]

연산(단위) / 블록	읽기(페이지)	쓰기(페이지)	소거(블록)
소블록(64M x 8Bits)	15μs	200μs	2ms
대블록(2G,4G,8G x 8Bit)	25μs	200μs	1.5ms

본 논문에서는 CLOCK-Pro[10]를 낸드 플래시 메모리에 적합하게 개선한 CLOCK-NAND를 제안한다. 디스크 기반의 버퍼 교체 정책인 CLOCK-Pro는 CLOCK 알고리즘과 유사한 방식으로 동작하고 LIRS[11]로부터 필요한 성능상의 이점을 채택하여 우수한 성능을 보여준다. CLOCK-NAND는 이러한 CLOCK-Pro를 기본 모델로 정하여 추가적으로 페이지 접근 정보를 효과적으로 활용하고, 더티 페이지에 대해 효율적인 지연쓰기를 하도록 설계하였다.

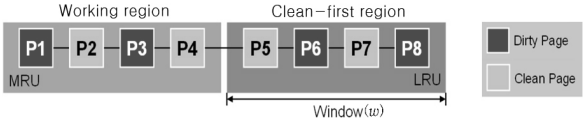
본 논문의 구성은 다음과 같다. 2장은 관련 연구로 CFLRU, LIRS-WSR과 CLOCK-NAND의 기본이 되는 디스크 기반의 버퍼 교체 정책인 CLOCK-Pro에 대해서 기술한다. 3장은 본 논문에서 제안하는 플래시 메모리의 특성을 고려한 버퍼 교체 전략인 CLOCK-NAND에 대해서 자세히 기술한다. 4장에서는 시뮬레이션을 기반으로 성능 평가를 보이고, 5장에서 결론으로 마무리한다.

2. 관련 연구

2.1 CFLRU

CFLRU는 플래시 메모리의 특성을 고려하여 LRU를 개선한 정책이다. CFLRU는 버퍼의 페이지를 더티 페이지와 클린 페이지로 구별하고, LRU의 알고리즘에 기반하여 관리한다. CFLRU의 구조는 (그림 1)과 같이 빈번하게 접근되는 페이지를 포함하고 있는 작업 영역(Woring region)과 페이지 교체가 일어나는 미수정 우선 영역(Clean-first region)으로 구성되어 있다. 페이지 교체 시에 더티 페이지가 교체되면 플래시 메모리에 쓰기 연산을 발생시키므로 CFLRU는 우선 미수정 우선 영역의 클린 페이지 중에서 LRU 방식으로 교체될 페이지를 선택하고, 클린 페이지가 존재하지 않을 경우에 LRU 방식으로 더티 페이지를 선택한다.

(그림 1)의 예제는 전체 버퍼의 크기를 8, 미수정 우선 영역의 윈도우(Window)의 크기(w)는 4로 가정한 경우이다. 페이지 교체 시에 LRU에서는 페이지 반출 순서가 P8, P7, P6, P5인 반면에, CFLRU에서는 클린 페이지인 P7, P5를 먼저 반출한 후에, 더티 페이지인 P8, P6을 반출하여 P7, P5, P8, P6의 순서로 반출이 이루어진다. CFLRU는 이러한 방법으로 더티 페이지를 버퍼에 오랫동안 유지하여 지연쓰기를 유도한다. 그러나 접근 빈도에 관계없이 클린 페이지보다 더티 페이지를 버퍼에 오랫동안 유지하므로 메모리 적중률



(그림 1) CFLRU의 구조

이 저하되어 많은 읽기 연산이 발생할 수 있다. 또한, 쓰기 요구(혹은 읽기 요구)만 존재하는 환경에서는 버퍼에 더티 페이지나 클린 페이지 중 한 종류만 존재하므로 LRU와 동일하게 동작한다.

2.2 LIRS-WSR

LIRS는 IRR(Inter-Reference Recency)의 원리를 사용한 디스크 기반의 버퍼 교체 정책으로 IRR은 어떤 페이지가 접근되었다가 재접근되는 사이에 접근된 중복되지 않은 다른 페이지들의 수이다. 그리고 Recency는 어떤 페이지가 마지막으로 접근된 이후부터 현재 사이에 접근된 중복되지 않은 다른 페이지의 수이다. LIRS는 IRR이 작은 페이지를 LIR(Low Inter-reference Recency) 페이지로, IRR이 큰 페이지를 HIR(High Inter-reference Recency) 페이지로 구별하여 관리하며, 페이지 교체시에 버퍼에 상주(resident)하는 HIR 페이지 중 최대 IRR을 가진 페이지를 교체한다. 교체된 페이지는 메타데이터를 포함하는 버퍼 비상주(non-resident) HIR 페이지가 되어 리스트에 남게 된다.

LIRS-WSR은 WSR(Write Sequence Reordering)을 LIRS에 채택한 정책으로 WSR은 콜드 플래그를 사용하여 콜드 페이지가 아닌 더티 페이지가 교체되는 것을 지연시키기 위한 방법이다.

LIRS-WSR은 (그림 2)와 같이 LIRS 스택과 ListQ로 구성되어 있다. LIRS 스택에는 LIR 페이지, HIR 페이지의 엔트리가 존재하고 ListQ에는 버퍼에 상주하는 HIR 페이지의 엔트리만 존재한다. LIRS-WSR의 기본 알고리즘은 다음과 같다.

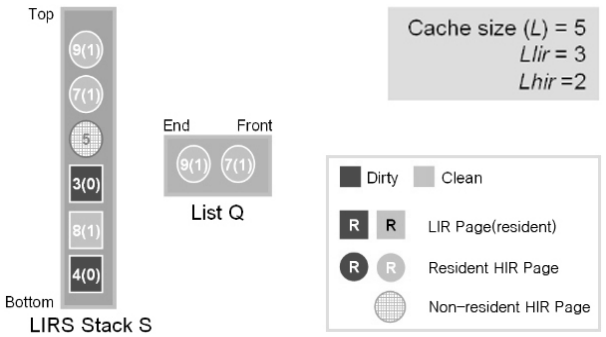
LIR 페이지가 접근되면 콜드 플래그가 0이 되어 스택 위로 이동한다. HIR 페이지가 접근되는 경우는 버퍼 상주와 비상주의 경우로 나뉜다. 버퍼 상주 HIR 페이지가 접근 시 스택에 있으면 LIR 페이지가 되어 스택 위로 이동하고, ListQ에 있는 동일한 HIR 페이지는 제거된다. 그리고 스택 바닥의 LIR 페이지는 HIR 페이지로 변경되어 ListQ의 끝으로 이동한다. 이 때 스택 바닥의 LIR 페이지가 더티이고 콜드플래그가 0이면, 콜드플래그만 1로 변경되고 스택 위로 이동하여 HIR 페이지로 변경이 연기된다. 변경 혹은 연기가 된 후에는 스택프루닝(stack pruning)이 수행되어 스택 바닥에 LIR 페이지가 오게 된다. 스택에 없는 버퍼 상주 HIR 페이지에 접근이 일어나면 HIR 페이지는 ListQ 끝으로 이동된다. 버퍼 비상주 HIR 페이지가 접근된 경우는 페이지 폴트(page fault)로 ListQ 앞의 HIR 페이지를 교체하여 페이지를 읽어오고 버퍼 비상주 HIR 페이지는 LIR 페이지로 변경된다. 그리고 스택 바닥의 LIR 페이지는 HIR 페이지로 변경되어 ListQ 끝에 삽입되고 마찬가지로 스택프루닝이 수행된다. (그림 2)의 예제는 전체 버퍼의 크기를 5, LIR 페이지의 최대 수를 3, HIR 페이지의 최대 수를 2로 가정하였다. 버퍼 비상주 HIR 페이지 5가 접근되면, 버퍼 공간을 할당하기 위해 버퍼에 상주하는 HIR 페이지 중 최대 IRR을 가진 페이지 7이 교체되고, HIR 페이지 5는 LIR 페이지가 되어 스택 위로 이동한다. 그리고 스택 바닥의 LIR 페이지가 HIR 페이지로 변경되어야 한다. 그러나 스택 바닥의 LIR 페이지 4가 더티 페이지이고 콜드 플래그가 0이므로 콜드 플래그만 1로 변경되어 스택 위로 이동하고 LIR 페이지 8을 검사한다. LIR 페이지 8은 콜드 플래그가 1이고 더티 페이지가 아니므로 HIR 페이지가 되어 ListQ로 이동한다.

LIRS-WSR은 플래시 메모리의 특성을 고려한 지연쓰기로 쓰기 연산 횟수를 줄인다. 그러나 교체된 HIR 페이지를 스택푸르닝 전까지 버퍼 비상주 HIR 페이지로 유지하고, LIRS 스택과 ListQ에 중복된 버퍼 상주 HIR 페이지 엔트리를 가지므로 버퍼 공간의 사용이 비효율적일 수 있다.

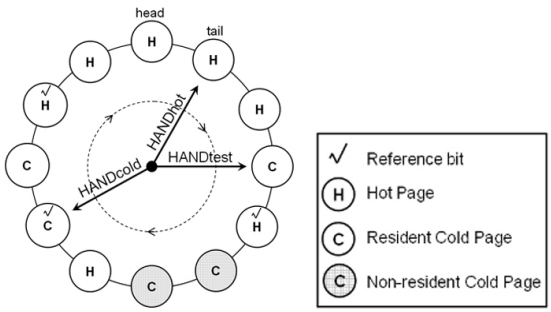
2.3 CLOCK-Pro

CLOCK-Pro는 IRR과 동일한 재사용 거리(reuse-distance)에 기반하여 재사용 거리가 짧은 페이지를 핫 페이지로, 재사용 거리가 긴 페이지를 콜드 페이지로 구별하여 관리한다. 모든 페이지는 기본 값이 0이고 접근 시 1이 되는 참조 비트를 가지며, 각 콜드 페이지에는 페이지가 리스트의 끝을 지날 때까지의 시간과 같은 테스트 기간(test period)이 주어져 이 기간 중에 참조되면 핫 페이지로 변경될 수 있다.

CLOCK-Pro는 순환리스트 구조로 시계방향으로 이동하는 세개의 클락 핸드를 가지고 있다. HANDcold는 페이지 폴트 시에 교체할 콜드 페이지를 찾기 위한 시작 위치인 버퍼 상주 콜드 페이지 중 최대 재사용 거리를 가진 페이지를 가리킨다. HANDhot은 최대 recency를 가진 핫 페이지를 가리키며, HANDhot이 지나가면 참조비트가 0인 핫 페이지는 콜드 페이지로 변경된다. (그림 3)에서와 같이 HANDhot



(그림 2) LIRS-WSR의 구조



(그림 3) CLOCK-Pro의 구조

은 항상 리스트 끝을 가리키고 시계 방향으로 HANDhot을 바로 뒤따르는 페이지는 리스트 앞이다. 즉, HANDhot의 이동은 가리키는 페이지를 리스트 앞으로 보내는 것이다. HANDtest는 리스트의 끝에서 가장 가까운 콜드 페이지를 가리키며, HANDtest가 지나간 콜드 페이지의 테스트 기간은 종료되고 버퍼 비상주 콜드 페이지는 리스트에서 삭제된다.

CLOCK-Pro의 페이지 교체는 클락 핸드들의 동작을 조합하여 이루어진다. 우선, HANDcold는 참조비트가 0인 버퍼 상주 콜드 페이지를 찾아 교체한다. 교체된 콜드 페이지는 메타데이터를 가진 버퍼 비상주 콜드 페이지가 되어 남은 테스트 기간 동안 리스트에 유지된다. 그러나 버퍼 상주 콜드 페이지의 참조비트가 1이고 테스트 기간 중에 있지 않으면 참조비트만 0으로 변경하여 리스트 앞으로 이동시키고, 다시 교체할 페이지를 찾아 이동한다. 만약, HANDcold가 만나는 콜드 페이지가 테스트 기간 중에 있고 참조비트가 1이면 핫 페이지로 변경 되어 리스트 앞으로 이동하고 참조비트는 0이 된다. 콜드 페이지가 핫 페이지로 변경된 후에는, HANDhot이 가리키는 리스트 끝의 핫 페이지는 콜드 페이지가 되어 리스트 앞으로 이동된다. 그러나 HANDhot이 가리키는 핫 페이지의 참조비트가 1이면 참조비트만 0으로 변경되고 HANDhot은 다시 참조비트가 0인 핫 페이지를 찾아 이동한다. 이 과정에서 HANDhot은 만나는 콜드 페이지의 테스트 기간을 종료시키고, 버퍼 비상주 콜드 페이지를 제거한다. 실제로 HANDhot이 HANDtest의 역할을 대신하여 동작한다.

CLOCK-Pro는 디스크 기반의 저장장치에서 좋은 성능을 보이지만 플래시 메모리의 비대칭 연산 비용의 특성을 고려하고 있지 않아 플래시 메모리 기반의 저장 장치에서 사용하기에는 적절하지 못하다. 또한, 테스트 기간 중에 있는 콜드 페이지에 다수의 접근이 발생해도 HANDcold를 만나기 전까지 핫 페이지로 변경되지 못하므로 페이지 접근 정보의 활용이 효율적이지 못하다.

3. CLOCK-NAND

3.1 CLOCK-NAND의 기본 아이디어

CLOCK-NAND는 CLOCK-Pro의 알고리즘을 기본으로 하고 있지만 낸드 플래시 메모리 기반의 환경에서 우수한 성능을 보일 수 있도록 낸드 플래시 메모리가 가진 고유의 특성을 고려하여 설계하였다. CLOCK-NAND의 목적은 페이지 교체가 요구될 때 버퍼에서 자주 참조되는 더티 페이지가 플래시 메모리로 반출되는 수를 줄임으로써 고비용의 쓰기 연산 횟수를 줄이는 데 있다. 이러한 목적을 위해 CLOCK-NAND는 추가적으로 설계한 새로운 두가지 정책을 사용한다. 첫 번째 정책은 더티이면서 빈번하게 참조되는 페이지가 버퍼에서 반출되는 것을 지연하기 위해 핫 페이지가 콜드 페이지로 변경되는 것을 지연하는 콜드 변경 지연 정책이다. 두 번째 정책은 참조비트에 의해 표현되는 각 페이지의 참조 정보를 보다 효율적으로 활용하여 자주 참조되는 페이지를 버퍼에 오래 유지하기 위한 빠른 핫 페이지 변경 정책이다. 이러한 정책을 사용하여 CLOCK-NAND의 버퍼 적중률이 다소 저하되고 읽기 연산이 추가적으로 발생할 수 있다. 하지만 이러한 방법은 효과적으로 쓰기 연산 횟수를 감소시키고 결과적으로 플래시 메모리 기반의 시스템에서 전체적인 성능을 향상시킨다.

3.2 콜드 변경 지연 정책

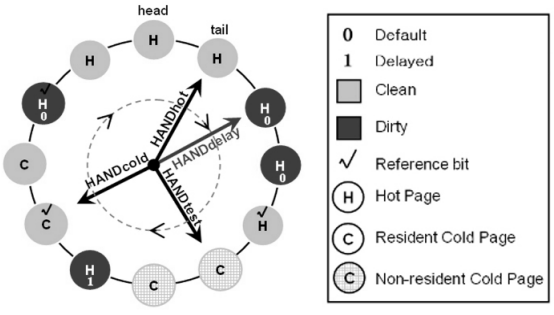
본 논문에서 제안하는 CLOCK-NAND의 콜드 변경 지연 정책은 플래시 메모리의 비대칭적 연산 비용을 고려하기 위한 것으로 빈번하게 접근되는 더티 페이지의 교체를 연기하기 위한 방법이다. 그리고 CLOCK-Pro의 기존 세가지 클락 핸드 외에 (그림 4)와 같이 HANDdelay를 추가로 사용한다.

CLOCK-NAND의 콜드 변경 지연 정책의 기본 알고리즘은 다음과 같다.

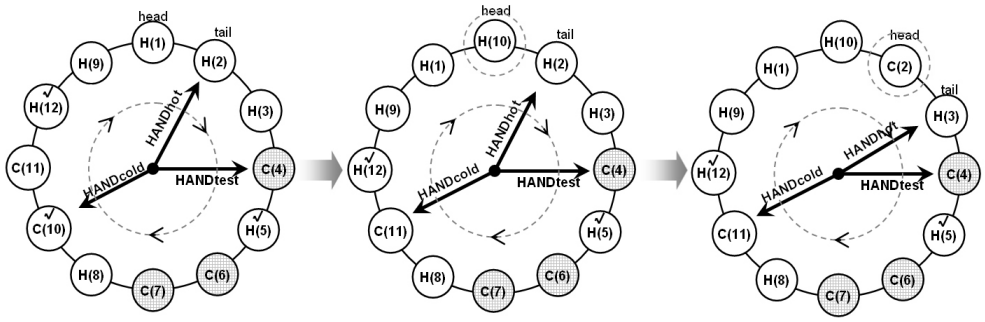
1) 각 페이지에 수정 여부를 구분하기 위해 더티비트라는 비트 플래그를 추가하였다. 더티비트 0은 클린 페이지를 나타내고 더티비트 1은 더티 페이지를 나타낸다.

2) 더티인 핫 페이지는 (그림 4)와 같이 기본값이 0으로 설정된 콜드 지연비트를 가진다. 콜드 지연비트는 핫 페이지에서 콜드 페이지로의 변경 지연여부를 나타내는 것으로 콜드 페이지로의 변경이 지연된 핫 페이지는 콜드 지연비트가 1이 된다.

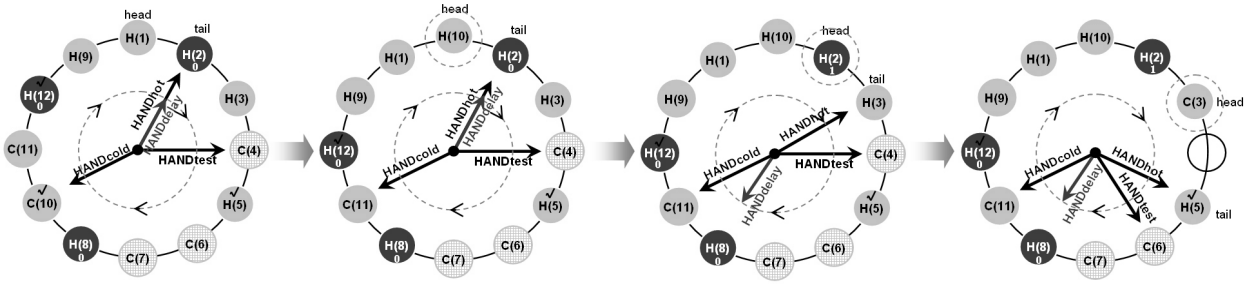
3) HANDdelay는 더티인 핫 페이지 중 콜드 지연비트가 0이고 리스트의 끝에서 최소 거리에 있는 페이지를 가리킨다. 핫 페이지를 콜드 페이지로 변경하는 경우에 HANDhot이 HANDdelay가 가리키는 핫 페이지를 동일하게 가리키면 HANDdelay에 의해 핫 페이지는 콜드 지연비트만 1로 변경되고 리스트의 앞으로 이동하여 콜드 페이지로의 변경이 연기된다. 단, 핫 페이지의 참조비트가 1인 경우에는 콜드 지연비트를 그대로 유지하고 CLOCK-Pro와 동일하게 참조비트만 0으로 변경되어 리스트의 앞으로 이동한다. 그리고 HANDdelay는 다시 콜드 지연비트가 0인 더티인 핫 페이지를 찾아 이동하고 HANDhot은 콜드 페이지로 변경할 핫 페이지를 찾기 위해 이동한다. 실제로 CLOCK-Pro에서 HANDhot이 HANDtest의 역할을 수행하는 것과 같이 HANDdelay의 역할을 HANDhot이 대신하여 수행하므로 구현에서 HANDdelay의 추가에 따른 오버헤드는 발생하지 않는다.



(그림 4) CLOCK-NAND의 구조



(그림 5) CLOCK-Pro의 콜드 페이지 변경 예제



(그림 6) CLOCK-NAND의 콜드 변경 지연 정책

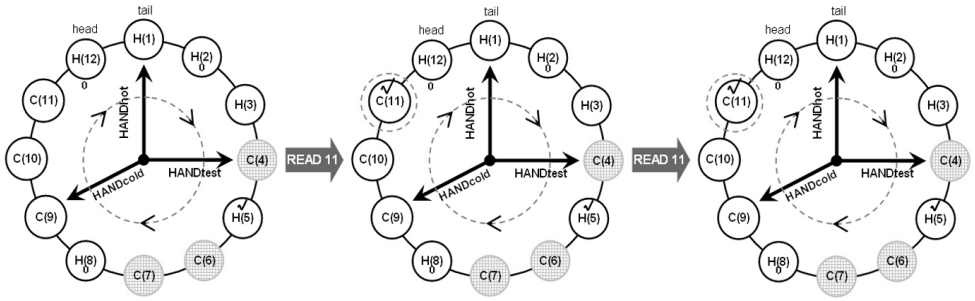
(그림 5)와 (그림 6)은 CLOCK-Pro와 CLOCK-NAND의 핫 페이지의 콜드 페이지 변경 예제로 전체 페이지 수는 9개, 핫 페이지는 7개, 버퍼 상주 콜드 페이지는 2개로 가정하였다. 두 예제에서 테스트 기간 중에 접근되어 참조비트가 1이 된 콜드 페이지 10은 핫 페이지가 되어 리스트의 앞으로 이동한다. 그리고 HANDhot이 가리키는 최대 재사용 거리를 가진 핫 페이지를 콜드 페이지로 변경하게 된다. (그림 5)에서와 같이 CLOCK-Pro는 핫 페이지 2를 콜드 페이지로 변경하여 리스트의 앞으로 보낸다. 그러나 CLOCK-NAND는 플래시 메모리의 비대칭 연산 비용을 고려하여 지연쓰기를 하기 위해 더티인 핫 페이지가 콜드 페이지로 변경되는 것을 연기시킨다. 따라서 (그림 6)에서와 같이 더티인 핫 페이지 2의 콜드 지연비트가 0이므로 HANDdelay에 의해 콜드 지연비트만 1로 변경하고 리스트의 앞으로 보내 콜드 페이지로 변경을 연기시킨다. 그리고 HANDhot은 시계방향으로 이동하여 핫 페이지 3을 검사하고, 핫 페이지 3이 클린 페이지이며 참조비트가 0이므로 콜드 페이지로 변경되어 리스트의 앞으로 이동한다. 마지막으로 HANDhot은 최대 재사용 거리를 가진 핫 페이지를 찾을 때까지 이동하면서 테스트 기간이 종료된 버퍼 비상주 콜드 페이지 4를 삭제하고 최대 재사용 거리를 가진 핫 페이지 5를 찾은 후 정지한다.

3.3 빠른 핫 페이지 변경 정책

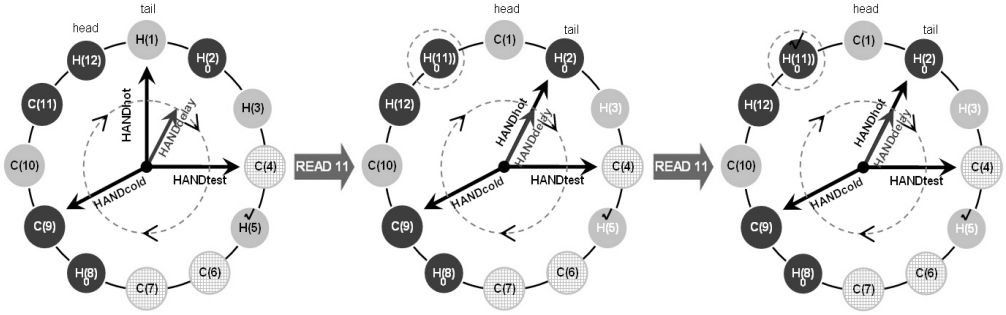
CLOCK-Pro는 현재의 접근 행동을 정확히 반영하는 콜드/핫 상태를 확립하기 위해 콜드 페이지가 핫 페이지로 변경될 수 있는 기회를 제공한다. CLOCK-Pro에서 콜드 페이지가 핫 페이지가 되는 경우는 HANDcold가 교체할 페이지를 찾아 이동하는 과정에서 테스트 기간에 있는 참조비트가 1로 변경된 콜드 페이지를 만나는 경우이다. 따라서 테스트 기간 중에 있는 참조비트가 1인 핫 페이지일지라도 HANDcold를 만나기 전까지는, 그대로 콜드 페이지로 남아 있게 된다. 이러한 방법에서는 한동안 페이지 교체 없이 테스트 기간에 있는 콜드페이지에 빈번하게 접근이 발생하는 경우에 페이지 접근 정보를 제대로 활용하기가 어렵다. 따라서 CLOCK-NAND에서는 페이지 접근 정보를 보다 효율적으로 활용하기 위해 테스트 기간에 있는 콜드 페이지가 접근되어 참조비트가 1이 되면, 즉시 핫 페이지로 변경하는 방법을 사용한다.

(그림 7)과 (그림 8)의 예제는 테스트 기간 중에 있는 참조비트가 0인 콜드 페이지 11에 연속으로 두번의 읽기가 발생했다고 가정한 경우이다. HANDcold는 최대 재사용 거리를 가진 참조비트가 0인 콜드 페이지를 찾아서 교체하는데 예제에서는 모두 페이지 교체가 요구되지 않아 HANDcold는 정지한 상태로 있게 된다. 따라서 (그림 7)의 CLOCK-Pro 예제에서 연속적인 2회의 접근이 발생한 콜드 페이지 11은 HANDcold를 만나지 못해 참조비트만 1로 변경되고 핫 페이지가 되지 못한다. 그러나 (그림 8)의 CLOCK-NAND 예제에서 콜드 페이지 11은 첫 번째 접근 시에 참조비트가 0인 핫 페이지가 되고, 두 번째 접근 시에는 참조비트가 1인 핫 페이지가 된다.

이와 같은 방법으로 CLOCK-NAND는 빠른 핫 페이지 변경 정책을 사용하여 테스트 기간 중에 있는 콜드 페이지에 빈번하게 접근이 발생했을 때 CLOCK-Pro보다 페이지 접근 정보를 효율적으로 활용할 수 있다.



(그림 7) CLOCK-Pro의 핫 페이지 변경 예제



(그림 8) CLOCK-NAND의 빠른 핫 페이지 변경 예제

3.4 CLOCK-NAND의 알고리즘

본 장에서는 CLOCK-NAND의 알고리즘에 대해서 서술한다. 알고리즘 1의 의사코드는 요구되는 페이지를 찾기 위해 순환 리스트를 순차적으로 검색하며 페이지의 메타데이터를 변경하는 것을 나타낸다. 순환 리스트 내에 요구되는 페이지가 존재하면, 5줄과 같이 페이지의 참조비트를 1로 변경한다. 이 경우에 6-8줄과 같이 페이지가 비상주 콜드 페이지로 존재하면 페이지를 할당할 공간을 만들기 위해 알고리즘 3을 사용한다. 그리고 10-15줄과 같이 콜드 페이지가 테스트 기간에 있으면 빠른 핫 페이지 변경 정책을 적용하여, 참조비트를 0으로 변경하고 핫 페이지로 변경하여 리스트의 앞으로 이동시킨다. 만약 페이지가 더티이면 13줄과 같이 콜드 지연비트를 0으로 설정한다. 그리고 16줄과 같이 알고리즘 2를 사용하여 리스트 끝의 핫 페이지를 콜드 페이지로 변경한다.

알고리즘 2는 리스트 끝의 핫 페이지를 콜드페이지로 변경하는 의사코드이다. HANDhot은 가리키는 핫 페이지의 참조비트가 1인 경우 5줄과 같이 참조비트를 0으로 변경 후 이동하고, 참조비트가 0인 경우에는 7-8줄에서와 같이 콜드 지연비트를 검사하여 0이면 1로 변경한다. 즉, HANDhot은 10-11번째 줄과 같이 참조비트가 0이고 콜드 지연비트가 1인 핫 페이지를 찾을 때까지 이동하여 콜드 페이지로 변경하고, 15-17줄과 같이 이동하는 과정에서 만나는 비상주 콜드 페이지는 순환 리스트에서 삭제한다.

알고리즘 3의 의사코드는 순환 리스트에서 HANDcold가 가리키는 가장 큰 재사용 거리를 가진 콜드 페이지를 교체하는 것을 나타낸다. HANDcold는 이동하면서 3-8줄과 같이 참조비트가 0인 버퍼 상주 콜드 페이지를 찾아 교체하여 비상주 콜드 페이지로 만들고, 10줄과 같이 버퍼 비상주 콜드 페이지를 만나면 다음 페이지로 이동한다. 그리고 핫 페이지를 만나는 경우에도 13번째 줄과 같이 다음 페이지로 이동한다.

Algorithm 1. *SearchPage (lsn)*

Input : lsn
Output : isExist (0=default, 1=exist)

```
1  isExist = unset
2  for current_page = list_head to list_tail
3     if lsn == lsn of current_page then
4           set isExist
5           set reference bit in current_page
6           if coldflag in current_page == set then
7                 if resident bit in current_page == unset then
8                       ReplaceColdPage()
9                 end if
10                if current_page is in test period then
11                      unset reference bit in current_page
12                      if dirty bit in current_page == set then
13                            unset cold_delay bit in current_page
14                      end if
15                      unset coldflag in current_page move
                        current_page to list_head
16                      ChangeHottoCold()
17                end if
18          end if
19          break
20    else
21          current_page = next page in a counterclockwise
            direction
22    end if
23 end for
```

Algorithm2. *ChangeHottoCold ()*

Input : none
Output : isChanged (0=default, 1=changed)

```
1   while isChanged == 0
2       Handhot_page = page which is currently pointed by
        HANDhot
3       if coldflag in Handhot_page = unset then
4           if reference bit in Handhot_page == set then
5               unset reference bit in Handhot_page
6           else
7               if cold_delay bit in Handhot_page == unset
                then
8                   set cold_delay bit of Handhot_page
9               else
10                  set coldflag in Handhot_page
11                  set isChanged
12              end if
13          end if
14      else
15          if resident bit in Handhot_page == unset then
16              delete Handhot_page from circular list
17          end if
18      end if
19      HANDhot moves in a clockwise direction as a page
20  end while
```

Algorithm3. *ReplaceColdPage ()*

Input : none
Output : none

```
1   while isReplaced == 0
2       Handcold_page = page which is currently pointed by
        HANDcold
3       if coldflag in Handcold_page == set then
4           if resident bit in Handcold_page == set then
5               replace Handcold_page from circular list
6               unset resident bit in Handcold_page
7               isReplaced = 1
8               HANDcold keeps moving until it meets page
                which has setted coldflag
9           else
10              HANDcold moves in a clockwise direction as
                a page
11          end if
12      else
13          HANDcold moves in a clockwise direction as a
            page
14      end if
15  end while
```

4. 실험 및 성능 평가

본 논문에서 제안한 버퍼 교체 정책의 성능을 평가하기 위해 기존의 버퍼 교체 정책인 CFLRU와 LIRS-WSR을 시뮬레이터로 구현하여 비교하였다. 시뮬레이션은 인텔 제온 3.2GHz 듀얼 프로세스와 2기가의 메인 메모리를 가진 서버 장비와 리눅스 OS커널 버전 2.6.18상에서 수행하였다.

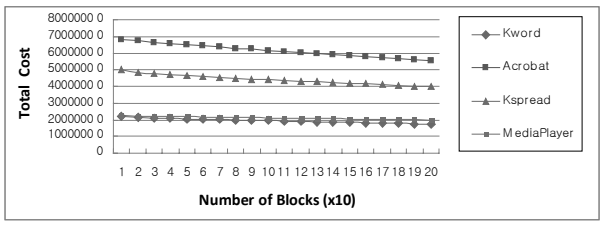
4.1 트레이스 파일 분석 및 환경 설정

시뮬레이션에는 리눅스 기반의 응용 프로그램들에서 추출된 가상 메모리 참조 정보의 트레이스(trace) 파일을 사용하였다. Kwords, Acrobat, Kspread, MediaPlayer의 4종류 응용 프로그램에서 추출되었으며, 각 트레이스(trace) 파일은 표 2에서와 같이 읽기 참조와 쓰기 참조가 구별되어 있는 총 50만개의 참조 정보로 구성되어 있다. Kword 트레이스는 읽기 참조 정보의 28.5%, 쓰기 참조 정보의 44.6%가 중복된 참조 정보이며, 읽기 참조 횟수가 쓰기 참조 횟수의 약 5배 규모이다. Acrobat 트레이스는 읽기 참조 정보의 25.9%, 쓰기 참조 정보의 33.4%가 중복되어 있다. 그리고 쓰기 참조 횟수가 읽기참조 횟수의 약 2배 규모이다. Kspread 트레이스는 읽기 참조 정보의 20.4%, 쓰기 참조 정보의 76.1%가 중복된 참조 정보로 쓰기 참조 정보에 대해 높은 지역성을 보인다. MediaPlayer 트레이스는 읽기 참조 정보의 55.2%, 쓰기 참조 정보의 63.2%가 중복되어 있으며, 읽기 참조 횟수와 쓰기 참조 횟수가 유사하다. 전체적으로 4종류의 트레이스 파일은 읽기 참조 정보보다 쓰기 참조 정보에 대해 높은 지역성을 가지고 있다.

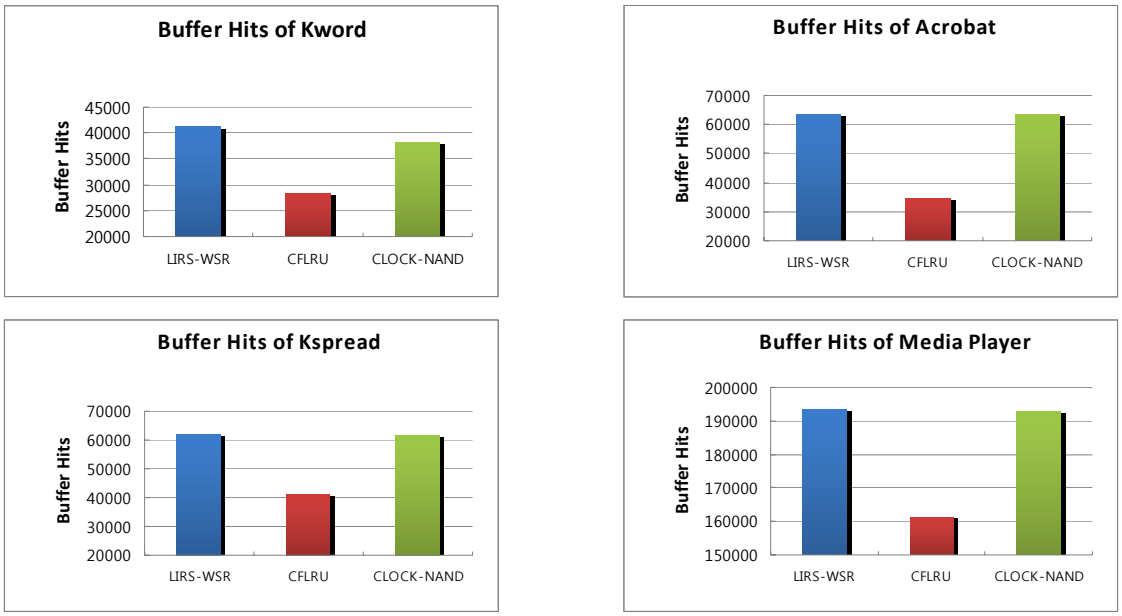
CLOCK-NAND에서 전체 버퍼에 대한 핫 페이지와 콜드 페이지의 최적화된 비율을 구하기 위해 비율을 변경해가며 성능 측정을 하였고, 그 중 최적화된 성능을 보인 80:20의 비율을 실험에 반영하였다. 또한, 최적의 성능을 보여주는 버퍼의 크기를 측정하기 위해 (그림 9)와 같이 소블록 단위로 버퍼의 크기를 변경하면서 총 연산 비용을 측정하였다. 총 연산 비용은 실험에서 발생한 읽기 연산 횟수와 쓰기 연산 횟수를 측정하여 읽기 연산과 쓰기 연산에 표 1의 수치를 적용하여 계산하였다. (그림 9)의 결과에서 버퍼의 크기가 소블록 130개일 때 총 연산 비용 측면에서 최적의 성능을 보여주었으므로 시뮬레이션에서 가상으로 잡은 버퍼의 크기는 소블록 130개의 크기인 약 2MB에 해당한다. CLOCK-NAND와 유사한 환경에서 성능을 측정하기 위해 비교 대상

〈표 2〉 실험에 사용된 트레이스 파일

	Kword	Acrobat	Kspread	MediaPlayer
Total	500000	500000	500000	500000
Read	419582	159142	259465	245370
Write	80418	340858	240535	254630



(그림 9) 버퍼사이즈 변경에 따른 총 비용



(그림 10) 각 트레이스에 대한 버퍼 적중률

인 LIRS-WSR, CFLRU에도 동일한 크기의 버퍼를 할당하였고, 각각의 버퍼 교체 정책의 알고리즘에 따른 페이지 비율도 설정하였다. LIRS-WSR의 경우, LIR 페이지와 HIR 페이지의 개념이 CLOCK-NAND의 핫 페이지와 콜드 페이지의 개념과 유사하므로 LIR 페이지와 HIR 페이지의 비율을 80:20으로 설정하였다. CFLRU는 본 실험에서 사용한 데이터 상에서 미수정 우선 영역의 비율을 고정적으로 80% 할당하였을 때 가변적인 방법이나 다른 비율에 비교하여 최적의 성능을 보여주었기 때문에 작업 영역 : 미수정 우선 영역의 비율을 20:80으로 설정하였다.

4.2 버퍼 적중률 측정

(그림 10)과 같이 버퍼 적중률을 측정하여 비교한 결과, CLOCK-NAND는 접근빈도에 상관없이 더티 페이지를 클린 페이지보다 지연쓰기하는 CFLRU보다 최소 16.2%에서 최대 45.4%의 향상된 적중률을 보였고, LIRS-WSR보다는 최소 0.1%에서 최대 7.2%의 저하된 적중률을 보였다. 4가지 경우가 모두 CFLRU의 버퍼 적중률이 다른 두가지 버퍼 교체 정책에 비해 저하된 것은 CFLRU가 지연쓰기를 하기 위해 미수정 우선 영역에 더티 페이지를 오래 유지하면서 빈번하게 접근되는 클린 페이지를 반출시키는 경우가 발생하

(그림 11) 각 트레이스에 대한 총 비용

기 때문이다. 이 경우, 버퍼 적중률이 떨어지게 되고 추가적인 읽기 연산이 발생하게 되어 전체적인 성능저하로 이어진다. CLOCK-NAND는 더티인 핫 페이지가 콜드 페이지로 변경되는 것을 지연하므로, LIR 페이지 중 재 접근되어 콜드플래그가 0이 된 LIR 페이지가 HIR 페이지로 변경되는 것을 지연하는 LIRS-WSR보다 더욱 지연쓰기를 하게 된다. 따라서 버퍼에 더티 페이지를 더 오래 유지하게 되면서 버퍼 적중률이 상대적으로 떨어진다.

4.3 총 연산비용 측정

CLOCK-NAND는 (그림 11)에서와 같이 총 연산비용 면에서 CFLRU보다 최소 3.9%, 최대 10.8%의 성능 향상을 보였고, LIRS-WSR보다는 최소 1.8%, 최대 5.2%의 성능 향상을 보였다. CFLRU의 총 연산비용이 LIRS-WSR과 CLOCK-NAND보다 큰 것은 앞서 말했듯이 지연쓰기를 위해 더티 페이지를 접근 빈도를 고려하지 않고 미수정 우선 영역에 오랫동안 유지하면서 메모리 적중률이 저하되고 추가적인 다수의 읽기 연산이 발생했기 때문이다. 그리고 CLOCK-NAND는 LIRS-WSR보다 버퍼 적중률이 저하된 만큼 더티 페이지에 대해 지연쓰기를 하여 쓰기 요구를 최소화하므로 총 연산비용이 LIRS-WSR보다 우수한 결과를 보여준다.

5. 결론 및 향후 연구

낸드 플래시 메모리는 고유의 특성으로 인하여 빈번한 쓰기 연산과 추가적인 삭제 연산의 발생은 성능 저하의 원인이 된다. 본 논문에서는 낸드 플래시 메모리의 비대칭적인 연산 속도를 고려하여 CLOCK-NAND라는 새로운 버퍼 교체 정책을 제안하였다. CLOCK-NAND는 디스크 기반의 저장 장치에서 우수한 성능을 보여준 CLOCK-Pro의 기본 알고리즘에 기반하여 있지만 CLOCK-Pro보다 페이지 접근 정보를 효율적으로 활용하기 위해 콜드 페이지에 대해 빠른 핫 페이지 변경 정책을 사용한다. 그리고 빈번하게 참조되는 더티 페이지가 플래시 메모리로 반출되는 것을 지연하기 위해 콜드 변경 지연 정책을 사용한다. 본 논문에서는 CLOCK-NAND의 성능을 입증하기 위해서 다양한 종류의 트레이스 파일을 사용하여 시뮬레이션을 수행하였다. 그리고 시뮬레이션의 결과에서 CLOCK-NAND가 기존의 플래시 메모리 기반의 버퍼 교체 정책들보다 전체적으로 우수한 성능을 보여준다는 것을 증명하였다. 또한, 낸드 플래시 메모리 기반의 환경에서 버퍼 교체 정책들에 의한 다양한 성능을 확인하였다.

향후 연구 과제로써 CLOCK-NAND를 플래시 주소 변환 계층(Flash Translation Layer)[2]과 연계된 실제의 임베디드(embedded) 시스템에 적용하여 실험하고 개선할 계획이다.

참 고 문 헌

[1] Greg Gagne, Abraham Silberschatz. Peter Baer Galvin, “Operating System Concepts sixth edtion”, *Wiley*, 2003.

[2] BAN, A, “Flash file system”, *United States Patent, No. 5,404,485*, April, 1995.

[3] CHUNG, T. S. et al., “System software for flash memory: a survey”, *IFIP International Conference on Embedded And Ubiquitous Computing (EUC 2006)*, 2006

[4] Samsung Electronics, “64M x 8Bits 낸드 Flash Memory (K9F1208X0C)”, 2007.

[5] Samsung Electronics, “2G x 8Bit / 4G x 8Bit / 8G x 8Bit 낸드 Flash Memory (K9XXG08XXM)”, 2007.

[6] JiYong Shin et al., “FTL design exploration in reconfigurable high-performance SSD for server applications”, *Proceedings of the 23rd international conference on Supercomputing*, 2009.

[7] Sang-Won Lee, Bongki Moon, and Chanik Park, “Advances in flash memory SSD technology for enterprise database applications”, *SIGMOD '09: Proceedings of the 35th SIGMOD international conference on Management of data*, June, 2009.

[8] Seon-Yeong Park, Dawoon Jung, Jeong-Uk Kang, Jin-Soo Kim, and Joonwon Lee, “CFLRU: a replacement algorithm for flash memory”, *In CASES '06: Proceedings of the 2006 International Conference on Compilers, Architecture and Synthesis for Embedded Systems*, pp.234-241, 2006.

[9] Hoyoung Jung et al, “LIRS-WSR: Integration of LIRS and writes sequence reordering for flash memory”, *Lecture Notes in Computer Science, 4705*, pp.224-237, 2007.

[10] Song Jiang, Feng Chen, and Xiaodong Zhang, “CLOCK-Pro: an effective improvement of the CLOCK replacement”, *Proceedings of 2005 USENIX Annual Technical Conference (USENIX'05)*, 2005.

[11] Song Jiang and Xiaodong Zhang, “LIRS: an efficient low inter-reference recency set replacement policy to improve buffer cache performance”, *In SIGMETRICS '02: Proceedings of the 2002 ACM SIGMETRICS International Conference on Measurement and Modeling of Computer Systems*, pp.31-42, 2002.

김 종 선

e-mail : sprigan1@hanyang.ac.kr

2006년 한양대학교 컴퓨터공학전공(학사)

2009년 한양대학교 컴퓨터공학과(석사)

2006년~2007년 다날(주)

2009년~현 재 LIG 넥스원 항공연구센터 연구원

관심분야 : 데이터베이스, 플래시메모리용 시스템소프트웨어 등

손 진 현

e-mail : jhson@hanyang.ac.kr
1996년 서강대학교 전산학과(학사)
1998년 한국과학기술원 전산학과(석사)
2001년 한국과학기술원 전자전산학과(박사)
2001년 9월~2002년 8월 한국과학기술원 전자전산학과 박사후 연구원
2002년 9월~현 재 한양대학교 컴퓨터공학과 부교수
관심분야 : 데이터베이스, e-비즈니스, 유비쿼터스 컴퓨팅, 임베디드 시스템

이 동 호

e-mail : dhlee72@hanyang.ac.kr
1995년 홍익대학교 컴퓨터공학과(학사)
1997년 서울대학교 컴퓨터공학과(석사)
2001년 서울대학교 컴퓨터공학과(박사)
2001년~2004년 삼성전자 책임연구원
2004년~현 재 한양대학교 컴퓨터공학과 부교수
관심분야 : 데이터베이스, 멀티미디어 정보검색, 플래시메모리용 시스템소프트웨어 등