NAND 플래시 메모리 파일 시스템을 위한 더블 캐시를 활용한 페이지 관리 정책

(A Policy of Page Management Using Double Cache for NAND Flash Memory File System)

박 명 규 † 김 성 조 ††
(Myung Kyu Park) (Sung Jo Kim)

요 약 NAND 플래시 메모리는 특성상 덮어쓰기 연산이 불가능하기 때문에 지움 연산이 선행되어야 하므로 I/O 처리 속도가 느려지게 되어 성능저하의 원인이 된다. 또한 지움 횟수가 제한적이어서 지움 연산이 빈번히 발생하게 되면, NAND 플래시 메모리의 수명이 줄어든다. 이러한 문제점을 해결하기 위해 NAND 플래시 메모리의 특성을 고려한 쓰기 지연 기법을 사용하면, 쓰기 횟수가 줄어들어 I/O 성능 향상에 도움이 되지만, 캐시 적중률이 낮아진다. 본 논문은 NAND 플래시 메모리 파일 시스템을 위한 더블 캐시를 활용한 페이지 관리 정책을 제안한다. 더블 캐시는 실질적인 캐시인 Real Cache와 참조 페이지의 패턴을 관찰하기 위한 Ghost Cache로 구성된다. 이 정책은 Ghost Cache에서 쓰기를 지연함으로써 Real Cache에서의 적중률을 유지할 수 있고, Ghost Cache를 Dirty 리스트와 Clean 리스트로 구성하여 Dirty 페이지에 대한 탐색 시간을 줄임으로써 쓰기 연산 성능을 높인다. 기존 정책들과의 성능을 비교한 결과 제안된 정책이 기존 정책들에 비해 평균적으로 적중률은 20.57%, 그리고 I/O 성능은 20.59% 우수했고, 쓰기 횟수는 30.75% 줄었다.

키워드 : NAND 플래시 메모리, 버퍼 교체 알고리즘, 캐시 정책

Abstract Due to the physical characteristics of NAND flash memory, overwrite operations are not permitted at the same location, and therefore erase operations are required prior to rewriting. These extra operations cause performance degradation of NAND flash memory file system. Since it also has an upper limit to the number of erase operations for a specific location, frequent erases should reduce the lifetime of NAND flash memory. These problems can be resolved by delaying write operations in order to improve I/O performance; however, it will lower the cache hit ratio. This paper proposes a policy of page management using double cache for NAND flash memory file system. Double cache consists of Real cache and Ghost cache to analyze page reference patterns. This policy attempts to delay write operations in Ghost cache to maintain the hit ratio in Real cache. It can also improve write performance by reducing the search time for dirty pages, since Ghost cache consists of Dirty and Clean list. We find that the hit ratio and I/O performance of our policy are improved by 20.57% and 20.59% in average, respectively, when comparing them with the existing policies. The number of write operations is also reduced by 30.75% in average, compared with of the existing policies.

Key words : NAND Flash Memory, Buffer Replacement Algorithm, Cache Policy

· 본 연구는 2008년 중소기업 산학협력지원사업 지원으로 수행되었음

† 학생회원 : 중앙대학교 컴퓨터공학과
blackaqua@konan.cse.cau.ac.kr
†† 종신회원 : 중앙대학교 컴퓨터공학과 교수
sjkim@cau.ac.kr

논문접수 : 2008년 10월 23일
심사완료 : 2009년 6월 26일

정보과학회논문지: 시스템 및 이론 제36권 제5호(2009.10)

1. 서 론

컴퓨터 하드웨어 기술의 발전으로 프로세서와 기계적인 동작이 요구되는 하드 디스크와의 데이터 처리속도 차이는 더욱 커지고 있다. NAND 플래시 메모리는 비휘발성 기억장치로서 하드 디스크에 비해 탐색시간(seek time)이 빠르기 때문에 데이터 처리속도가 우수하며, 외부의 충격에 강하고, 크기가 하드 디스크에 비해 매우 작고, 무게도 가벼워 최근 모바일 기기나 이동식 저장매체, MP3 플레이어 등의 디지털 미디어 기기에서 광범위하게 사용되고 있다.

NAND 플래시 메모리의 읽기, 쓰기 속도는 하드 디스크에 비해 우수하다[1]. 하지만 이미 데이터가 기록된 장소에 덮어쓰기가 불가능하기 때문에 다시 기록하기 위해서는 블록 단위로 지우기 연산이 수행되어야 하며, 같은 주소 공간에 쓰기 연산이 제한된 횟수 이상 반복될 경우 해당 블록을 사용되지 못하게 되는 단점을 가진다. 캐시는 I/O 수행 시 저장장치 접근 횟수를 크게 줄일 수 있어 NAND 플래시 메모리의 지우기 연산 횟수를 획기적으로 줄일 수 있다.

기존 캐시 정책 중 대표적인 것으로는 LRU(Least Recently Used)[2]와 LFU(Least Frequently Used)[2]가 있다. LRU는 버퍼 안에서 가장 오랫동안 참조되지 않은 페이지를 교체하는 정책으로 참조 페이지의 패턴이 캐시의 크기보다 큰 순환참조일 때는 성능이 낮지만, 구현과 동작과정이 간단하고, 적중률이 높다는 점에서 가장 보편적으로 사용되는 캐시 정책이다. LFU는 페이지에 대한 적중률이 참조성에 영향을 받는 다는 것을 고려하여 버퍼 안에서 참조 횟수가 가장 적은 페이지를 교체하는 정책으로 짧은 시간 동안 참조가 많이 된 페이지가 앞으로 참조되지 않게 된다면, 캐시에서 제거되지 않고, 남아 있게 되어 캐시 공간의 효율성을 떨어뜨리는 단점을 가지고 있지만, 반복 참조 패턴에 대해서는 좋은 성능을 보인다.

플래시 메모리는 덮어쓰기가 불가능하기 때문에 쓰기 연산 오버헤드가 매우 크다. 또한 I/O 성능은 쓰기 연산에 앞서 시행되어야 하는 지우기 연산으로 인해 많은 영향을 받게 된다. LRU[2]와 LFU[2] 같은 캐시 정책은 NAND 플래시 메모리의 특성을 고려하지 않기 때문에 Dirty 페이지가 자주 교체하게 되면, 느린 쓰기 연산으로 인해 I/O 성능이 낮아지게 되고, 반복되는 쓰기 연산으로 인하여 NAND 플래시 메모리 수명이 단축될 수 있다. 이러한 단점을 보완하기 위해서 쓰기 연산 횟수를 줄이는 쓰기 지연 기법을 적용한 CF-LRU(Clean First LRU)[3], Dirty-Last[4], LRU-DPL-CD(LRU-Dirty Page Later Cold write page Detection)[5], FLRU (Flash memory LRU)[6]와 같은 NAND 플래시 메모리 기반 캐시 정책이 연구되었다. 본 논문에서는 NAND 플래시 메모리의 특성을 고려한 기존 정책들에서 Dirty 페이지의 쓰기 지연 때문에 적중률이 낮아지는 문제를 보완하기 위해 더블 캐시 정책을 이용하여 적중률과 I/O 성능 향상, NAND 플래시 메모리의 수명 연장을 위하여 캐시 안의 페이지를 Dirty와 Clean으로 구분하여 관리하는 정책을 제안한다. 이 정책은 Ghost Cache (Ghost Buffer)에서 페이지 폴트로 인해 교체할 페이지가 Dirty일 경우 쓰기 지연을 하여 쓰기 횟수를 줄여 I/O 성능을 높이고, Dirty와 Clean 페이지를 구분함으로써 Dirty 페이지 탐색 시간이 불필요하다. 또한 Dirty 페이지를 교체 시 순차 쓰기가 가능해져 쓰기 성능을 높인다.

본 논문의 구성은 다음과 같다. 2장은 NAND 플래시 메모리의 특성을 고려한 기존 캐시 정책들에 대해서 기술하고, 3장은 본 논문이 제안한 DLD-ARC(Double List for Dirty page - ARC) 캐시 정책을 설명한 다음, 다른 기존 캐시 정책들과의 차이점을 기술한다. 4장은 본 논문에서 제안한 캐시 정책과 기존의 캐시 정책들의 적중률 및 I/O 수행속도를 비교 측정한 결과를 분석하고, 5장에서는 본 연구의 결론과 향후 연구에 대해 기술한다.

2. 관련 연구

본 장에서는 NAND 플래시 메모리의 특성을 고려한 기존 캐시 정책인 Dirty-Last, CF-LRU, LRU-DPL-CD, FLRU의 특성과 장단점, 그리고 본 논문에서 개선하고자 하는 더블 캐시 정책 중 하나인 ARC(Adaptive Replacement Cache)[7]에 대하여 기술한다.

2.1 CF-LRU (Clean First LRU)

CF-LRU는 LRU를 플래시 메모리 저장장치에 적합하도록 개선한 알고리즘[3]이다.

그림 1의 CF-LRU 예에서는 가장 오랫동안 참조가 안된 순서대로 캐시 리스트에 6개의 페이지 F, E, D, C, B, A 가 있고, 이 중 D와 F가 Dirty 페이지라고 가정한다. 또한 Dirty 페이지의 쓰기 지연을 위해 윈도우 크기 W를 5페이지로 가정한다. W가 0일 경우 LRU와 동일하게 동작하며, W가 클수록 Dirty 페이지의 쓰기 연산 횟수가 줄어든다. 이러한 조건에서 페이지 폴트가 발생하면 캐시 내에서 W에 속한 5개의 페이지 B, C, D, E, F의 Dirty 여부를 확인한다. 현재 교체 대상 페

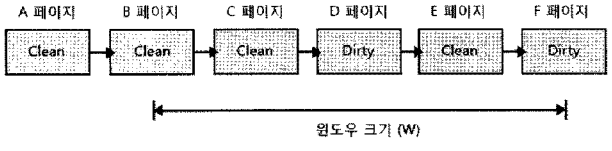


그림 1 CF-LRU 예

이지 F를 교체하게 되면 쓰기 연산이 발생하게 되므로, 첫 번째 Clean 페이지인 E가 교체된다. 만약 B, C, D, E, F가 모두 Dirty일 경우에는 가장 오래된 페이지인 F가 교체된다.

이와 같이 쓰기 지연을 통해 플래시 메모리의 쓰기 횟수가 감소될 수 있어 동일 페이지에 대해 쓰기 연산이 반복적으로 빈번히 발생하는 경우, Dirty 페이지가 캐시에 오래 유지되기 때문에 I/O 성능이 향상될 수 있다. 하지만 쓰기 연산이 빈번하게 발생하지 않는 경우, 자주 사용되지 않은 Dirty 페이지가 캐시 안에 장시간 남아 있을 가능성이 있기 때문에 캐시 공간이 낭비되어 적중률이 낮아진다[5]. 또한 W의 크기에 따라 캐시 성능이 달라지고, W의 크기를 사용자가 정의하기 위해서 많은 테스트를 수행해야 하는 단점도 있다.

2.2 Dirty-Last

Dirty-Last는 CF-LRU와는 달리 다양한 환경의 특성에 따라 LRU뿐만 아니라 MRU, LFU에도 적용될 수 있으며[4], CF-LRU에서의 W를 캐시 전체 공간을 사용하는 것으로 변형한 정책이다.

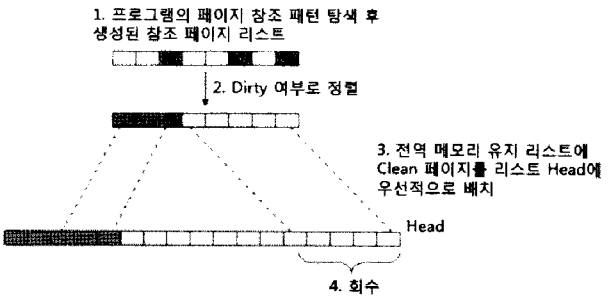


그림 2 Dirty-Last 예

Dirty-Last는 그림 2에서와 같이 참조 페이지 리스트를 생성한 후 패턴 탐색 모듈을 이용하여 프로그램의 페이지 참조 패턴을 미리 탐색하여 Dirty 페이지와 Clean 페이지로 나누어 정렬 한다. Clean 페이지를 캐시 리스트 Head에 배치함으로써 페이지 폴트로 인한 교체 시 Clean 페이지가 우선적으로 교체되도록 하고, 캐시 내 모든 페이지가 Dirty인 경우 Dirty 페이지가 교체되어 쓰기 연산이 수행된다. Dirty-Last는 쓰기 연산을 가능한 지연시킴으로써 I/O 성능이 높아지지만, 패턴 탐색으로 인한 오버헤드가 발생하고, 캐시 안의 모든 페이지가 Dirty 페이지가 되지 않을 경우, 장시간 캐시에 머무르게 됨으로써 공간 효율성과 캐시 적중률이 떨어질 수 있다.

2.3 LRU-DPL-CD(LRU-Dirty Page Later Cold write page Detection)

LRU-DPL-CD는 위에서 언급한 캐시 정책들과 마찬가지로 LRU에서 Dirty 페이지에 대해 쓰기를 지연하는 정책이다.

그림 3에서 볼 수 있는 바와 같이 LRU-DPL-CD는 페이지가 Dirty인지 Clean인지를 구분하는 Dirty Bit와 쓰기 지연 수행 여부를 확인하는 Cold Flag를 사용한다[5]. 먼저 Head가 가리키는 가장 오랫동안 참조 안된 페이지의 Dirty Bit를 확인하여 Clean일 경우 교체 대상이 되며, Dirty일 경우 Cold Flag를 확인한다. 만약 Cold Flag가 세트 되어 있지 않았다면 Cold Flag를 세트하고, 가장 최근에 참조된 페이지를 가리키는 Tail위치로 이동시키고, 세트 되어 있다면 교체 대상 페이지가 된다. 이런 간단한 기법을 통해 하나의 Dirty 페이지에 대한 쓰기 연산을 전체 캐시에 존재할 수 있는 페이지의 수에 해당하는 횟수만큼 지연시킬 수 있다.

LRU-DPL-CD는 Dirty 페이지의 쓰기 지연을 지원하기 위해 필요한 메모리 공간이 적고, 구현이 쉽다[5]. 하지만 리스트의 모든 페이지들이 Dirty이고 Cold Flag가 세트되어 있지 않은 worst case의 경우, 리스트에 속한 모든 페이지들의 Flag를 검사하고, 각 페이지를 Tail위치로 이동해야 하기 때문에 수행속도가 $O(n^2)$으로 느리다. 또한 CF-LRU과 마찬가지로 쓰기 지연으로 인해 캐시 적중률이 저하된다.

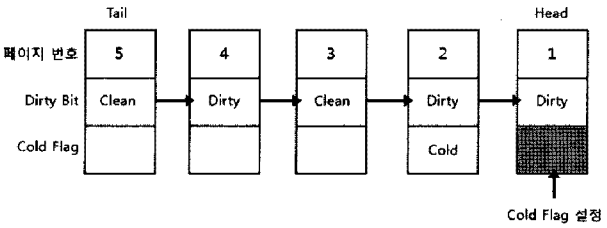


그림 3 LRU-DPL-CD 예

2.4 FLRU (Flash memory LRU)

FLRU 역시 LRU를 기반으로 변형된 캐시 정책으로서 NAND 플래시 메모리의 특성인 읽기, 쓰기, 지우기 비용이 서로 다르고 덮어쓰기를 할 경우 많은 오버헤드가 발생하는 것을 고려하여 각 연산에 대해 가중치(ERC: Expected Replacement Cost)를 적용[6]한다. ERC는 페이지의 교체 예상 비용으로서, 주소 변환 계층인 FTL(Flash Transition Layer)[8-10]에서 플래시 메모리의 물리적 공간 상태를 파악하여 가중치 값으로 표현된다. 쓰기 연산은 읽기 연산의 약 7배 비용이 소모되고, 덮어쓰기 연산은 읽기 연산의 약 65배 비용이 소모[11]되므로 읽기 연산을 기준으로 읽기/쓰기/덮어쓰기 연산의 평균 수행 시간 비율인 1/7/65가 가중치 값으로 이용된다. 교체 대상 페이지 중에서 페이지의 가치 값(C)이 가장 큰 페이지가 교체되며 캐시에 저장되는 시점, 저장되는 시간, 저장 후 참조되는 횟수, 각 연산에

대한 가중치를 기준으로 선정된다.

이 정책은 NAND 플래시 메모리에서의 읽기/쓰기/덮어쓰기 연산 비용에 대한 비율을 이용하여 쓰기를 지연함으로써 페이지 교체 비용을 줄일 수 있지만 페이지 교체가 발생할 때마다 모든 페이지의 참조 시간을 모두 계산해야 하고, 페이지 교체 예상 비용을 계산하기 위해 FTL에 접근해야 하기 때문에 상당한 시간적 오버헤드가 예상된다. 또한 참조 시간과 참조 횟수를 저장하기 위한 공간이 추가로 필요하며, 실행속도 및 공간 요구량이 캐시 크기에 비례하여 더욱 증가하는 단점을 가진다.

2.5 ARC(Adaptive Replacement Cache)

ARC는 이전에 참조된 페이지를 이용하여 캐시 공간을 효율적으로 관리하는 정책으로서 한번 참조된 페이지들을 관리하는 리스트(L_1)와 두 번 이상 참조된 페이지들을 관리하는 리스트(L_2)가 존재한다. 이 정책은 Ghost Cache에서 참조된 페이지의 위치에 따라 L_1, L_2 리스트의 캐시 공간 비율을 점차 변화시킨다.

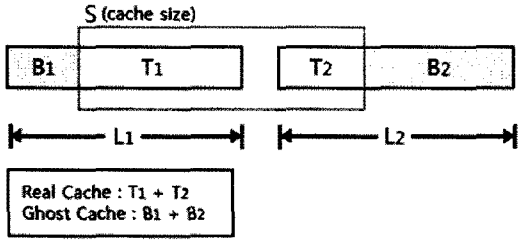


그림 4 ARC 예

그림 4는 실제 캐시 공간인 Real Cache ($T_1 + T_2$)의 크기가 S이고, Ghost Cache ($B_1 + B_2$)의 크기도 S이어서 전체 캐시 크기가 2S인 더블 캐시 정책의 예이다. 페이지가 T_1에서 교체되면 Ghost Cache 공간인 B_1으로, T_2에서 교체가 이루어지면 B_2로 이동한다. 참조 페이지가 Ghost Cache B_1에 존재한다면, 앞으로 참조될 페이지들이 참조 페이지가 속한 L_1의 Real Cache에서 참조될 가능성이 높다고 판단하여 P의 값이 변경되고, P에 의해 리스트 T_1와 T_2의 크기가 변화된다. 여기서 P는 $(0 \le P \le S)$범위를 가지며, T_1과 T_2 리스트 크기가 변화하는데 중요한 역할을 한다[7].

B_1의 크기를 2, B_2의 크기를 4, P의 값이 3으로 가정하고, 참조된 페이지가 B_1에 존재했을 경우, 참조된 페이지에 의해 앞으로 L_1의 페이지 수가 증가할 것이라고 예측하여 P 값은 B_2 / B_1로 계산된 2만큼 증가한다. B_1의 크기가 작고, B_2의 크기가 클수록 B_1에서 참조될 가능성이 낮아지기 때문에 지역성을 고려하여 P의 값을 증가시키기 위함이다. 반대로 B_2에 존재했을 경우, 앞으로 L_2의 페이지 수가 증가할 것이라고 예측하여 P 값은 B_1 / B_2로 계산된 1/2만큼 감소한다. 그리고 변경된 P 값에 의해 각 리스트의 크기가 변경되지 않고, 페이지 폴트 발생 시 1씩 변경된다. 이는 앞으로 참조될 페이지들이 앞서 예측된 결과와 동일하지 않을 수 있기 때문에 차츰 변화하여 캐시 크기를 유연하게 유지할 수 있게 한다. T_1이 T_2보다 커지고 있다면, 앞으로 참조될 페이지들은 T_1에 존재하는 페이지의 특징과 같은 한 번만 참조될 페이지일 경우가 많다는 것을 의미하고, 반대로 T_2가 T_1보다 커지고 있다면, T_2에 존재하는 페이지의 특징과 같은 앞으로 참조될 페이지들은 두 번 이상 참조될 페이지들이 많다는 것을 의미한다.

한 번 참조된 페이지들과 두 번 이상 참조된 페이지들을 구분하여 관리하는 특징을 보이는 ARC에서 L_1은 오래된 페이지를 교체 대상 페이지로 선정하는 LRU와 유사하고, L_2는 참조횟수가 가장 적은 페이지를 교체 대상 페이지로 선정하는 LFU와 유사한 특징을 보인다. LRU의 단점인 순환참조 시 성능저하 문제는 LFU를 이용하여 보완하고, LFU의 단점인 과거 참조가 많이 되었지만 앞으로 사용되지 않는 페이지임에도 캐시 공간을 지속적으로 차지하는 문제는 LRU를 이용하여 보완하였다.

ARC는 Ghost Cache를 사용함으로써 미래에 참조될 페이지들의 참조 패턴이 한 번만 참조될 페이지인지 지속적으로 참조될 가능성이 있는 페이지인지를 예측하여 Real Cache의 T_1과 T_2의 크기를 유동적으로 변화함으로써 다양한 참조 패턴에 대해 높은 성능을 유지할 수 있다[7]. 하지만 기존의 NAND 플래시 메모리의 특성을 고려한 정책들[3-6]과는 다르게 Dirty 페이지에 대한 쓰기 지연 기법을 적용하지 않았기 때문에 NAND 플래시 메모리에 적용할 경우, 페이지 교체 시 잦은 쓰기 연산으로 인한 지움 연산의 증가로 I/O 성능이 낮아지고, NAND 플래시 메모리의 수명이 줄어들게 된다. 이러한 Ghost Cache를 사용하는 더블 캐시 정책에는 2Q[12]와 적응형 LRFU(Least Recently Used and Least Frequently Used)[13] 등이 있다.

3. DLD-ARC 캐시 관리 정책

본 장에서는 NAND 플래시 메모리 기반 파일 시스템의 성능 향상을 위해 기존의 ARC를 개선한 DLD-ARC (Double List for Dirty page - ARC) 정책을 제안한다.

3.1 DLD-ARC 구조

DLD-ARC는 Ghost Cache를 사용하는 더블 캐시 정책 중 하나인 ARC를 NAND 플래시 메모리의 특성에 맞게 쓰기 지연을 고려하여 개선한 정책이다. 기존의 캐시 정책들은 I/O 성능 향상을 위해 캐시 공간에서 Dirty 페이지에 대한 쓰기 연산을 지연하였다. 하지만 쓰기 연산이 요구되는 Dirty 페이지들은 캐시 공간에

오래 남게 되어 캐시 적중률을 떨어뜨리는 원인이 된다. 더블 캐시는 실질적인 캐시인 Real Cache와 참조 페이지의 패턴을 관찰하기 위한 Ghost Cache로 구성되어 있어, Real Cache에서 쓰기 지연을 하지 않고, Ghost Cache에서 쓰기 지연을 적용함으로써 적중률이 떨어지는 문제를 해결할 수 있다.

DLD-ARC는 ARC와 같이 Real Cache와 Ghost Cache로 구성되고, Real Cache는 최근 한번 참조된 페이지를 관리하는 T_1과 최근 두 번 이상 참조된 페이지를 관리하는 T_2로 구성된다. Real Cache의 T_1과 T_2는 ARC와 같이 Ghost Cache를 이용하여 참조되는 페이지의 패턴에 의해 유동적으로 리스트의 크기가 변경되며, Ghost Cache는 Real Cache에서 이동된 페이지들로 구성된다. 그림 5에서 볼 수 있는 바와 같이 DLD-ARC에서는 Dirty 페이지의 효율적 관리를 위해 ARC에서의 B_1을 Dirty 페이지가 유지되는 리스트 D_1과 Clean 페이지가 유지되는 리스트 C_1로 구분하고, 같은 방식으로 B_2를 D_2와 C_2로 구분한다.

3.2 Dirty 리스트와 Clean 리스트

NAND 플래시 메모리의 특성을 고려한 기존 캐시 정책 중 성능이 우수한 LRU-DPL-CD 정책은 페이지 탐색 시간과 Dirty 페이지의 쓰기 지연을 위한 페이지 이동이 요구된다. 이러한 문제의 원인은 가장 오랫동안 참조 안된 페이지를 제거할 경우 Dirty 여부 확인 후 쓰기 지연 여부를 알기 위해 Cold Flag를 체크하는데, 만약 쓰기 지연이 되지 않은 페이지일 경우 해당 페이지를 리스트의 Tail로 이동하기 때문에 Dirty 페이지 탐색 시간과 페이지 이동 오버헤드가 발생하는데 있다. DLD-ARC 정책에서는 Ghost Cache를 Dirty 리스트와 Clean 리스트로 구분하여 이 문제를 해결하였다. 즉, 쓰기 연산이 필요한 페이지는 Dirty 리스트에 존재하기 때문에 교체할 페이지를 Clean 리스트의 Tail에서 선택함으로써 불필요한 탐색 시간과 페이지 이동 없이도 페이지를 빠르게 교체할 수 있다.

3.3 교체된 페이지의 쓰기 연산

Clean 리스트에 속한 페이지는 쓰기 연산이 필요 없기 때문에 즉시 교체되고, Dirty 리스트에 속한 페이지

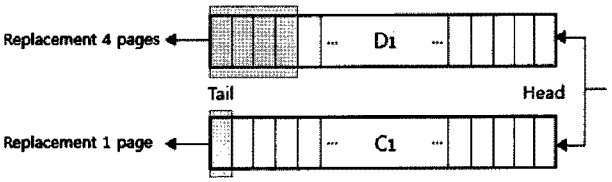


그림 6 Ghost Cache 공간에서 페이지 교체의 예

는 한 블록에 해당하는 페이지 수만큼 쓰기 연산을 수행한다. NAND 플래시 메모리의 구조상 파일을 순차적으로 읽기, 쓰기 연산을 수행할 경우, 임의 읽기, 쓰기 연산 보다 성능이 20~30% 좋아진다[14]. 앞서 언급한 바와 같이 Ghost Cache를 Dirty와 Clean 페이지로 구분하여 관리하게 되면 리스트에서 Dirty 페이지 여부 확인 및 탐색이 필요 없게 되어, 블록 단위로 순차 쓰기 연산을 할 때 큰 이점을 가진다.

DLD-ARC는 LRU-DPL-CD의 반복적 페이지 이동 문제점을 개선하기 위해 Dirty counter($0 \leq$ Dirty counter $\leq S$)를 활용한다. Dirty counter의 초기값은 0이고, 페이지 교체가 이루어지는 리스트에서 쓰기 지연이 발생할 때마다 1씩 증가한다. 만약 Dirty counter가 S와 같고, 페이지 폴트로 인해 교체가 이루어질 때 해당 Dirty 리스트의 Tail 페이지를 교체 대상 페이지로 선택하며, Dirty counter는 다시 초기화된다.

그림 6은 그림 5의 공간 [A] 에 속한 리스트 D_1, C_1에서 한 블록이 4개의 페이지로 구성되어 있다고 가정한다. 만약 Dirty counter가 S보다 작을 경우, C_1의 Tail 위치에서 하나의 페이지를 교체하고, Dirty counter가 S와 같을 경우, D_1의 Tail 위치에서 한 블록에 해당하는 4개의 페이지에 대하여 쓰기 연산을 수행한다. 이 때 Dirty 리스트의 페이지 수가 한 블록을 이루는 페이지 수보다 적을 경우, D_1의 모든 페이지에 대해 쓰기 연산을 수행함으로써 쓰기 연산 성능을 높인다.

3.4 DLD-ARC 알고리즘

그림 7에서 X와 S, $NPoneblock$, i, $|T_i|$, T_i^{Head}, T_i^{Tail} 등은 다음과 같은 의미를 가진다.

- X : 새로 삽입될 참조 페이지
- S : Real Cache 혹은 Ghost Cache의 캐시 크기

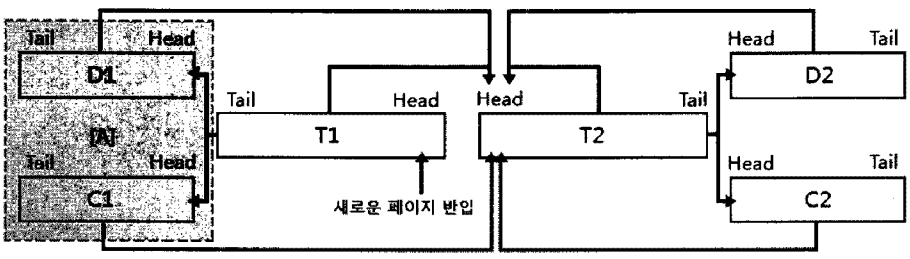


그림 5 DLD-ARC 구조

INITIALIZE
$T_1 = T_2 = C_1 = C_2 = D_1 = D_2 = P = dc_1 = dc_2 = i = 0$;

DLD-ARC(X)
① If (X in T_1 or T_2) then
  Move X to T_2^{Head};
② Else if (X in C_1 or D_1) then
  $P = \min\{S,\ P + \max\{(|C_2|+|D_2|)/(|C_1|+|D_1|),1\}\}$;
  Call Replace(P);
  Move X to T_2^{Head};
③ Else if (X in C_2 or D_2) then
  $P = \max\{0,\ P - \max\{(|C_1|+|D_1|)/(|C_2|+|D_2|),1\}\}$;
  Call Replace(P);
  Move X to T_2^{Head};
④ Else
  If ($|L_1| + |L_2| == 2S$) then
    If ($|L_1| \geq S$) then
      $i = 1$;
    Else
      $i = 2$;
    End if
⑤   If (($dc_i < S$) and ($|C_i| > 0$)) then
      Increment dc_1 and dc_2 by 1;
      Delete C_i^{Tail};
⑥   Else
      Set dc_i to 1;
⑦   Call Write(D_i);
    End if
  End if
⑧ If ($|T_1| + |T_2| == S$) then
    Call Replace(P);
  End if
⑨ Insert X at T_1^{Head};
End if

Function Replace(P)
If (($|T_1| \geq 1$) and (((X in C_2 or D_2) and $|T_1| = P$) or ($|T_1| > P$))) then
  $i = 1$;
Else
  $i = 2$;
End if
⑩ If (T_i^{Tail} is a dirty page) then
  Move T_i^{Tail} to D_i^{Head};
⑪ Else
  Move T_i^{Tail} to C_i^{Head};
End if

Function Write(D_i)
⑫ If ($|D_i| \geq NPoneblock$) then
  Write and Delete one block from the end of D_i;
⑬ Else
  Write and Delete all pages of D_i;
End if

그림 7 DLD-ARC 알고리즘

- $NPoneblock$: 한 블록 당 페이지의 수
- i : 1 혹은 2, 작업을 수행해야 하는 리스트를 선택하기 위해 사용 (예, L_1, L_2)
- $|T_i|$: T_i 리스트에 존재하는 페이지의 수
- T_i^{Head} : T_i 리스트의 Head
- T_i^{Tail} : T_i 리스트의 Tail
- L_i : T_i와 C_i, D_i을 포함하는 공간
- dc_i : L_i와 관련 있는 Dirty counter
- P : $P(0 \le P \le S)$는 ARC에서와 마찬가지로 캐시 내에서 적중된 참조 페이지가 위치해 있는 리스트 내의 다른 페이지들도 지역성으로 인하여 가까운 미래에 참조될 가능성이 높을 것이라는 예측 하에 실시간으로 Real Cache인 T_1과 T_2의 크기를 변화시켜 캐시의 적중률을 향상시키는데 활용

DLD-ARC 알고리즘은 기본적으로 ARC와 유사하며, 캐시에 참조 페이지(X)가 삽입되었을 때, DLD-ARC의 동작 순서는 다음과 같다. 먼저 X가 Real Cache의 T_1이나 T_2에 존재하는지 확인하고(①), 만약 존재한다면, X를 T_2의 Head로 이동시킨다. X가 C_1 또는 D_1에 존재하는 경우(②), P 값을 ARC에서와 동일한 방법으로 L_1의 Ghost Cache 공간 크기와 L_2의 Ghost Cache 공간 크기 비율을 고려하여 P 값을 증가시킨다. 그 다음 함수 *Replace()*를 호출하여 Real Cache에서 Ghost Cache로 이동할 페이지를 선택하여 이동시킨 뒤, X를 T_2의 Head로 이동시킨다. 반대로 C_2 또는 D_2에 X가 존재하는 경우, P 값을 감소시키고, 함수 *Replace()*를 호출한 뒤, X를 T_2의 Head로 이동시킨다(③).

만약 X가 Real Cache와 Ghost Cache에 존재하지 않을 경우(④), 모든 리스트의 길이의 합이 전체 캐시 크기인 $2S$와 같으면, 교체 대상 페이지를 선택해야 한다. 교체 대상이 되는 페이지는 3.3절에서 언급한 Dirty counter를 확인하여 Ghost Cache에서 교체된다. Dirty counter S면(⑤), Dirty counter를 1증가 시키고, Clean 페이지가 존재하는 C_1이나 C_2의 Tail에서 한 페이지를 제거한다. 만약 Dirty counter S면(⑥), Dirty counter를 1로 초기화한 뒤, 함수 *Write()*를 호출한다(⑦). Dirty 리스트의 페이지 수가 한 블록 당 페이지 수보다 많을 경우(⑫), 3.3절에서 언급한 쓰기 연산 성능을 향상시키기 위해 Dirty 페이지가 존재하는 D_1이나 D_2의 Tail에서 한 블록에 해당하는 페이지 수만큼 쓰기 연산을 하지만, 적을 경우 Dirty 리스트의 모든 페이지에 대해 쓰기 연산을 한다(⑬). 마지막으로 X를 삽입하기 전에 Real Cache가 Full이면(⑧), 함수 *Replace()*를 호출하여 Real Cache에서 Ghost Cache로 이동할 페이지를 선택하고, 마지막으로 X를 Real Cache인 T_1의 Head로 삽입한다(⑨).

ARC는 Ghost Cache를 B_1과 B_2로 구분하지만, DLD-ARC는 C_1, D_1, 그리고 C_2, D_2로 구분한다. 따라서 DLD-ARC의 함수 *Replace()*는 페이지를 이동시킬 때, ARC의 *Replace()*와는 달리 이동해야 하는 페이지의 Dirty인지 여부를 확인하여 Dirty일 경우(⑩), 리스트의 특성에 따라 D_i로 이동시키고, Clean일 경우(⑪), C_i로 이동시킨다. 또한 DLD-ARC에서는 D_i에서 교체가 발생하게 되면, 순차 쓰기 연산을 하기 위해 함수 *Write()*가 호출된다.

4. 성능 평가

본 장에서는 성능평가를 위해 DLD-ARC, LRU-DPL-CD, CF-LRU를 동일한 환경에서 IOzone Filesystem Benchmark[15]를 사용하여 적중률과 I/O 수행시간을 측정한다. 단 CF-LRU는 2.1절에서 언급했듯이 w의 크기에 따라 성능이 결정되므로 기존 논문[5]에서와 마찬가지로 W의 크기가 캐시의 10% 및 30%의 경우와 추가로 50%의 경우에 대해 성능을 측정하였다. 그리고 Dirty-Last는 패턴 탐색 모듈이 필요하며, FLRU은 FTL기반의 파일 시스템이 필요하여 동일한 환경에서 테스트가 이루어질 수 없기 때문에 성능 측정 대상에서 제외되었다.

성능은 Linux 2.6.29.3 상에서 1Gbytes의 저장공간을 가정하고 NAND 시뮬레이터[16]를 사용하여 측정하였다. IOzone Filesystem Benchmark는 파일의 크기를 4Kbytes부터 16Mbytes까지 2배씩 증가하며, 순차적으로 Read/Write를 수행한다.

4.1 적중률

그림 8에서 볼 수 있는 바와 같이 DLD-ARC의 적중률은 CF-LRU(10%), CF-LRU(30%), CF-LRU(50%), LRU-DPL-CD와 비교하여 평균적으로 각각 38.97%, 9.06%, 21.07%, 13.67% 정도 높은 성능을 보였다. CF-LRU(10%)는 쓰기 지연을 위한 공간이 전체 크기의 1/10를 차지하기 때문에 모든 구간에서 적중률이 가장 낮게 나타났으며, CF-LRU(30%)는 캐시 크기가 작은 구간에서는 LRU-DPL-CD와 비교하여 적중률이 높았지만, 캐시 크기가 커짐에 따라 유사한 결과를 보였다. CF-LRU는 캐시의 크기가 10%일 때보다 30%일 때 적중률이 좋았지만, 50%일 때는 적중률이 낮았다. 이러한 결과는 쓰기 지연을 위한 공간이 많아지게 되어 쓰기 횟수가 줄어드는 장점은 있지만, 참조되지 않는 dirty 페이지가 캐시에 오랫동안 존재하게 되어 적중률에 영향을 주었기 때문이다.

기존 연구[3,5]의 성능비교에서는 기존 LRU를 바탕으로 쓰기 지연 기법을 적용함으로써 적중률이 감소하였다. DLD-ARC는 기존 ARC를 바탕으로 쓰기 지연 기

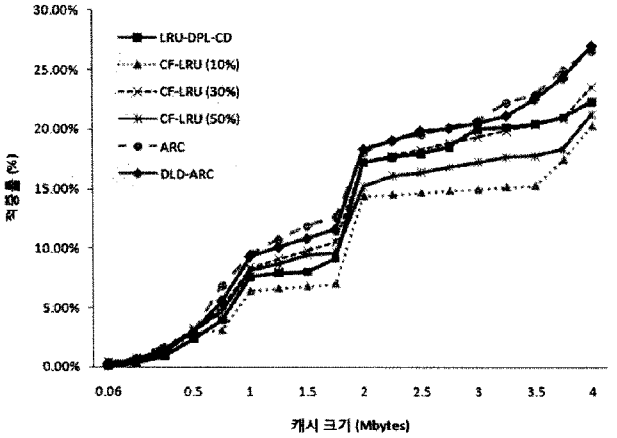


그림 8 적중률 비교

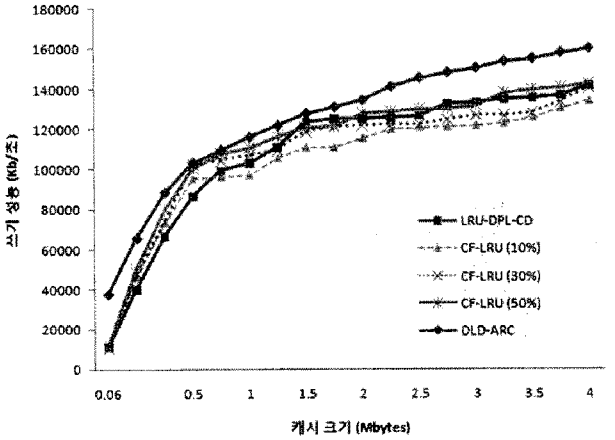


그림 10 쓰기 성능 비교

법을 적용하였기 때문에 적중률이 감소하는지 비교하기 위해 ARC의 적중률을 측정하였다. 그 결과, DLD-ARC는 모든 구간에서 다른 캐시 정책들[3,5]보다 높은 적중률을 보였으며, 기존 ARC와 비교하여도 크게 차이가 없다. 그 이유는 더블 캐시의 구조 중 하나인 Real Cache와 Ghost Cache를 다음과 같이 활용하였기 때문이다. 기존 ARC의 적중률은 Real Cache에서 측정되며, Ghost Cache를 앞으로 참조될 페이지들의 패턴을 예상하는 공간으로만 사용하였지만, DLD-ARC는 Ghost Cache를 쓰기 지연 기법을 적용하기 위한 공간으로도 사용하였다. 따라서 실제 캐시 공간인 Real Cache에서는 쓰기 지연 기법을 적용하지 않고, Ghost Cache를 이용함으로써 적중률이 유지될 수 있다는 것을 보여준다.

4.2 읽기/쓰기 성능

그림 9에서 볼 수 있는 바와 같이 DLD-ARC의 읽기 성능은 CF-LRU(10%), CF-LRU(30%), CF-LRU(50%), LRU-DPL-CD와 비교하여 평균적으로 각각 31.06%, 19.83%, 30.82%, 21.12%정도 높은 성능을 보였다. 읽기 성능은 전체적으로 적중률에 영향을 많이 받기 때문에 적중률이 가장 낮은 CF-LRU(10%)가 가장 낮았고, DLD-ARC가 가장 높았다.

그림 10에서 볼 수 있는 바와 같이 DLD-ARC의 쓰기 성능은 CF-LRU(10%), CF-LRU(30%), CF-LRU(50%) LRU-DPL-CD와 비교하여 평균적으로 각각 20.87%, 15.7%, 11.04%, 14.96%정도 높은 성능을 보였다. 그 이유는 DLD-ARC에서 쓰기 연산의 성능 향상을 위해 Dirty 리스트와 Clean 리스트를 별도로 구성하여 탐색 시간을 줄이고, 여러 개의 Dirty 페이지로 구성된 블록 단위로 순차 쓰기 연산을 하였기 때문이다.

4.3 쓰기 횟수

그림 11은 Dirty 페이지 교체 시 발생하는 쓰기 연산의 횟수를 비교한 결과이다. DLD-ARC의 쓰기 횟수는 CF-LRU(10%), CF-LRU(30%), CF-LRU(50%), LRU-DPL-CD와 비교하여 각각 35.01%, 30.11%, 23.3%, 27.12%정도 줄었다. 예상한 것과 같이 캐시에서 쓰기 지연을 위한 공간이 가장 적은 CF-LRU(10%)가 가장 많았고, 그 다음으로 CF-LRU(30%), LRU-DPL-CD, CF-LRU(50%) 순으로 많았다. 이러한 결과는 LRU가

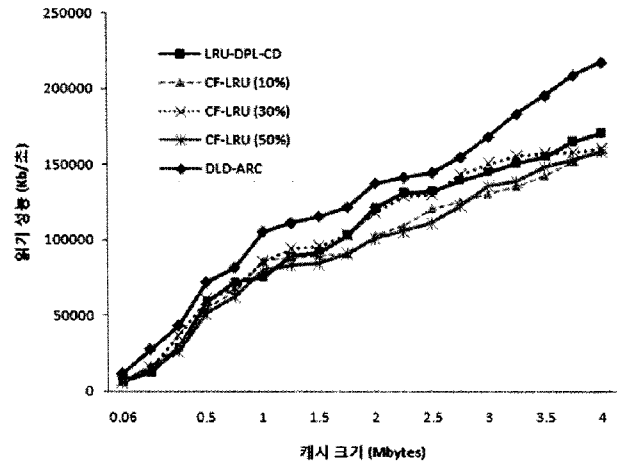


그림 9 읽기 성능 비교

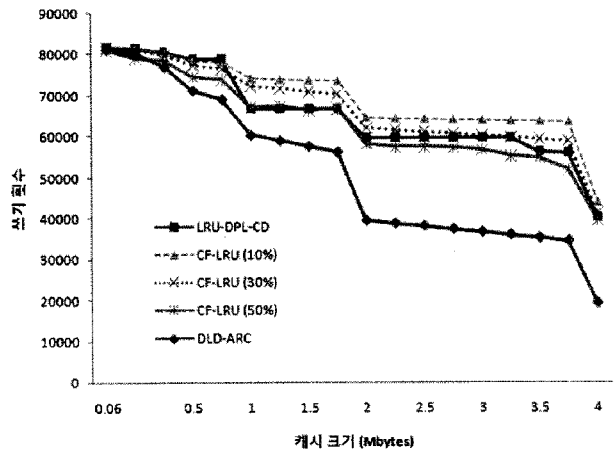


그림 11 쓰기 횟수 비교

순차 쓰기 연산에 대한 비효율성으로 인하여 나타나는 결과이다. 그리고 CF-LRU(50%)의 경우 캐시 크기의 절반을 쓰기 지연을 위한 공간으로 사용하여 LRU-DPL-CD보다 높은 성능을 보였다. DLD-ARC는 다른 캐시 정책들과 비교해서 캐시 크기가 작은 구간에서는 다른 정책들과 유사한 횟수를 보이지만, 캐시 크기가 점차 커지면서 쓰기 횟수가 크게 줄어들었다.

4.4 기존 기법과의 비교

NAND 플래시 메모리의 특성을 고려하여 ARC를 개선한 DLD-ARC는 동적 캐시 정책이다. 따라서 CF-LRU의 경우처럼 정적으로 W을 결정하는 정책보다 다양한 실시간 환경에 적합하다. 또한 이전에 참조된 페이지를 활용하여 앞으로 참조될 페이지들의 패턴을 예측하기 때문에 Dirty-Last와 같이 패턴을 탐색하기 위한 모듈이 불필요하며, Dirty 페이지의 삽입이 빈번히 발생하지 않을 경우 Dirty 페이지를 장시간 보관하게 되어 공간 효율이 낮아지는 단점도 없다. LRU-DPL-CD는 2.3절에서 언급한 바와 같이 기존 LRU에서 cold flag만이 추가적으로 필요하다. 이에 비해 DLD-ARC는 두 개의 dirty counter가 필요하여 공간 요구량이 다소 높다. 하지만 2.3절에서 언급한 바와 같이 LRU-DPL-CD는 worst case에서 $O(n^2)$의 수행속도를 보이지만, DLD-ARC는 모든 상황에서 페이지의 이동 없이 dirty counter를 증가시킴으로써 수행속도가 O(n)로 유지된다. 마지막으로 페이지 교체가 발생할 때마다 모든 페이지에 대한 가치(C)를 계산 하는 FLRU는 캐시에 존재하는 모든 페이지에 대하여 FTL에 접근하여 복잡한 계산 과정을 반복한다. 그리고 DLD-ARC는 두 개의 dirty counter가 필요한 반면, FLRU는 각 페이지의 참조 횟수 및 참조 시간을 유지해야 하기 때문에 많은 공간이 필요하다.

5. 결론 및 향후 계획

NAND 플래시 메모리를 위한 기존의 캐시 정책은 쓰기 지연을 통해 I/O 성능 향상이 가능하였으나, 적중률이 낮아지는 단점이 있다. 이는 성능 향상을 위해 실제 캐시 공간에서 가능한 한 Dirty 페이지의 쓰기 연산을 지연함으로써 오래된 페이지가 캐시에 남아 적중률을 떨어뜨리기 때문이다. 이를 해결하기 위해 본 논문에서는 NAND 플래시 메모리 파일 시스템을 고려하여 ARC 정책을 아래와 같이 개선하여 설계 및 구현하였다. 더블 캐시의 특징인 참조되는 페이지에 대한 예측에 필요한 Ghost Cache 공간을 Dirty 리스트와 Clean 리스트로 구분하였다. 그리고 dirty counter를 이용하여 Dirty 페이지의 이동 시간과 탐색 시간을 줄이고, 쓰기 연산이 발생하는 Dirty 페이지들을 효율적으로 지연하여 쓰기 횟수를 줄임으로써 I/O 성능을 개선하였다. 또한 기존 연구에서 나타나는 쓰기 지연에 의한 적중률의 감소 문제를 Ghost Cache 공간에 적용하여 이를 해결하였으며, Dirty 리스트에서의 순차 쓰기 연산을 통해 쓰기 성능을 높일 수 있도록 개선하였다.

성능 측정 결과 제안한 DLD-ARC의 적중률은 CF-LRU(10%), CF-LRU(30%), CF-LRU(50%), LRU-DPL-CD와 비교하여 각각 39.04%, 9.06%, 21.07%, 13.76% 정도, 읽기 성능은 각각 31.06%, 19.83%, 30.82%, 21.12%정도 높은 성능을 보였다. 쓰기 성능은 각각 14.96%, 20.87%, 11.04%, 15.7%정도 높은 성능을 보였고, 성능 향상에 영향을 미치는 쓰기 횟수 또한 CF-LRU(10%), CF-LRU(30%), CF-LRU50%), LRU-DPL-CD와 비교하여 각각 35.01%, 30.11%, 23.3%, 27.12% 정도 줄어 들었다. 결과적으로 제안된 정책이 가장 높은 적중률과 효과적으로 쓰기 지연 횟수를 줄임으로써 NAND 플래시 메모리 파일 시스템의 성능이 전체적으로 향상되었다.

향후 연구로서는 더블 캐시를 사용하는 다른 캐시 정책에도 NAND 플래시 메모리의 특성에 맞게 Dirty 페이지와 Clean 페이지 관리 기법을 적용하여 적중률 및 I/O 성능을 측정하고, 더블 캐시를 사용하지 않는 캐시 정책에 본 논문에서 제안한 페이지 관리 기법을 적용하기 위한 연구가 진행될 예정이다.

참 고 문 헌

[1] Needham & Company, LCC, *NAND vs. Hard Disk Drives: Hype, Myth and Reality*, October, 2005.

[2] Greg Gagne, Abraham Silberschatz, Peter Baer Galvin, *Operating System Concepts*, 6th edition, Wiley, 2003.

[3] Chanik Park, Jeong-Uk Kang, Seon-Yeong Park, Jin-Soo Kim, "Energy-aware Demand Paging on NAND Flash-based Embedded Storages," *Proceedings of the 2004 International Symposium on Low Power Electronics and Design Table of Contents*, pp.338-343, 2004.

[4] 박상오, 김경산, 김성조, "NAND 플래시 메모리용 파일 시스템 계층에서 프로그램의 페이지 참조 패턴을 고려한 캐시 및 선반입 정책", *정보처리학회논문지 A*, 제 14-A권 제4호, pp.235-244, 2007.

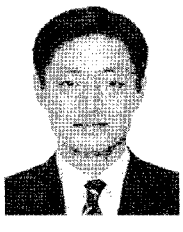
[5] 정호영, 박성민, 차재혁, 강수용, "플래시 메모리를 위한 Not-cold-page 쓰기지연을 통한 LRU 버퍼교체 정책 개선", *정보과학회논문지: 시스템 및 이론*, 제33권 제9호, pp.634-641, 2006.

[6] 박종민, 박동주, "플래시 메모리상에서 시스템 소프트웨어의 효율적인 버퍼 페이지 교체 기법", *정보과학회논문지: 데이터베이스*, 제34권 제2호, pp.133-140, 2007.

[7] N. Megiddo, D.S. Modha, "ARC: A Self-Tuning, Low Overhead Replacement Cache," *Proc. Usenix*

Conf. File and Storage Technologies, Usenix, pp 115-130, 2003.
[8] Li-Pin Chang, Tei-Wei Kuo, "An Efficient Management Scheme for Large-Scale Flash-Memory Storage Systems," In *Proceedings of the ACM Symposium on Applied Computing (SAC)*, pp.862-868, 2004.
[9] Jesung Kim, Jong Min Kim, S. H. Noh, Sang Lyul Min, Yookun Cho, "A Space-Efficient Flash Translation Layer for CompactFlash Systems," *IEEE Transactions on Consumer Electronics*, vol. 48, no.2, pp.366-375, 2002.
[10] Michael Wu, Willy Zwaenepoel, "eNVy: A Non-Volatile, Main Memory Storage System," In *Proceedings of the ACM International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, pp. 86-97, 1994.
[11] J.Kim, J.M. Kim, S.H. Noh, S.L. Min, and Y. Cho, "A Space-Efficient Flash Translation Layer for Compact Flash Systems," *IEEE Transactions on Consumer Electronics*, vol.48 no.2, pp.366-375, 2002.
[12] T. Johnson and D. Shasha, "2Q: A Low Overhead High Performance Buffer Management Replacement Algorithm," in *Proc. VLDB Conf.*, pp.297-306, 1994.
[13] 이종민, 현철승, 이동희, "적응형 LRFU 블록 교체 정책의 설계", *한국컴퓨터종합학술대회논문집*, vol.34, no.1(A), 2007.
[14] CommDesign, Flash *memory 101: An Introduction to NAND Flash*, 2006.
[15] IOzone Organization. "IOzone Filesystem Benchmark," http://www.iozone.org/.
[16] NAND Simulator. "Memory Technology Device (MTD) Subsystem for Linux," http://www.linux-mtd.infradead.org/.

박 명 규
2008년 한림대학교 컴퓨터공학과(공학사). 2008년~현재 중앙대학교 컴퓨터공학과(석사과정). 관심분야는 NAND 플래시 메모리 파일 시스템, 임베디드 시스템



김 성 조
1975년 서울대학교 응용수학과(공학사) 1977년 한국과학기술원 전산과(이학석사) 1977년~1980년 ADD(연구원). 1980년~현재 중앙대학교 컴퓨터공학부 교수 2007년~현재 한국공학교육인증원 부원장 2009년~현재 Seoul Accord 사무 총장 2009년~현재 한국정보기술학술단체연합회 회장. 2009년~현재 한국정보과학회 회장. 관심분야는 이동컴퓨팅, 임베디드 소프트웨어, 유비쿼터스컴퓨팅