# 비휘발성 메모리 기반 저장장치를 위한 클린 블록 우선 교체 기법의 성능 분석*

양수현, 류연승
명지대학교 컴퓨터공학과
e-mail:ysryu@mju.ac.kr

# Performance Analysis of Clean Block First Replacement Scheme for Non-volatile Memory Based Storage Devices

Soo-Hyun Yang, Yeonseung Ryu
Dept of Computer Engineering, Myongji University

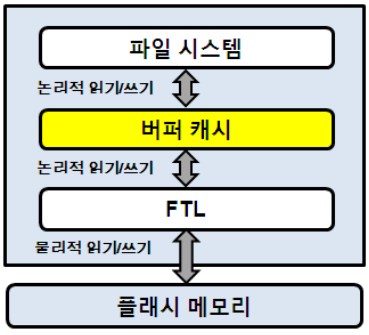
요 약

최근 차세대 저장장치로서 비휘발성 플래시 메모리 기반 저장장치의 사용이 증가하고 있다. 본 논문에서는 플래시 메모리 기반 저장장치의 특성인 삭제 연산의 문제점을 고려하는 새로운 버퍼 캐시 교체 기법을 연구하였다. 제안한 클린 블록 우선(Clean Block First) 기법은 버퍼를 플래시 메모리의 삭제 블록의 리스트로 관리하고 클린 페이지를 가진 블록을 우선적으로 교체하여 플래시 메모리의 삭제 연산 횟수를 줄인다. 트레이스 기반의 시뮬레이션을 수행하여 교체를 위해 검색하는 클린 블록 개수의 변화에 대한 캐시 적중률과 삭제 연산 횟수를 분석하였다.

## 1. 서론

NAND 플래시 메모리는 저 전력소비, 빠른 연산 속도 등의 장점을 가지고 있지만 다음과 같은 특별한 하드웨어 특징을 가지고 있다 [1]. 첫째, 플래시 메모리는 블록(block)들로 구성되며, 각 블록은 고정된 개수의 페이지(page)로 구성된다. 플래시 메모리에 대한 기본 명령어는 읽기, 쓰기, 삭제인데 읽기와 쓰기의 기본 단위는 페이지이며, 삭제는 블록 단위로 수행된다. 둘째, 플래시 메모리는 저장되어 있는 데이터를 바로 변경할 수 없으며 데이터가 저장되어 있는 블록에 대해 삭제 연산을 한 후에 변경할 내용을 쓸 수 있다. 삭제 연산은 읽기와 쓰기 연산에 비해 상대적으로 많은 시간이 소요된다. 셋째, 한 블록에 대한 삭제 연산의 횟수가 제한되어있다 (예, 10,000번). 만약 특정한 블록에 삭제 연산이 집중되면 해당 블록은 쓰이지 못하게 되며 플래시 메모리 장치의 수명에 영향을 미치게 된다.

운영체제는 플래시 메모리를 하드 디스크처럼 사용하기 위해 FTL (Flash Translation Layer)이라는 디바이스 드라이버 소프트웨어를 사용한다 [2-5]. FTL은 결함(bad) 블록 관리, 논리 주소의 물리 주소로의 변환, 에러 검사 등의 기능을 수행한다. 특히, FTL은 플래시 메모리에 대한 삭제 연산 수를 줄이기 위한 논리 주소 변환 기능을 수행한다. 플래시 메모리 상의 데이터의 변경이 요구될 때 데이터의 원래 위치에 변경할 내용을 저장하는 원위치 변경(in-place update) 방식은 블록의 삭제를 요구하기 때문에 쓰기 연산의 성능을 저하시킨다. 이러한 문제점을 해결하기 위해, FTL은 변경할 데이터를 원 위치가 아닌 다른 위치에 저장하고 위치 사상 테이블(address mapping table)을 두어 관리하는 타 위치 변경(out-of-place update) 방식이 널리 쓰이고 있다.




(그림 1) 운영체제 구성

(그림 1)은 플래시 메모리를 저장장치로 사용하는 운영체제의 구조를 보이고 있다. 운영체제는 접근 속도가 느린 저장장치의 성능 향상을 위해 주기억장치를 사용하는 버퍼 캐시 메커니즘을 제공한다. 예를 들어, 운영체제가 응용 프로그램의 읽기요청을 수행할 때, 파일 시스템은 요청받은 데이터를 저장장치에서 버퍼 캐시로 복사한다. 이후의 데이터 접근은 버퍼 캐시에서 수행하여 접근 성능을 향상시킬 수 있다. 그림에서, FTL은 파일 시스템

* 이 논문은 2010년도 정부(교육과학기술부)의 재원으로 한국연구재단의 기초연구사업 지원을 받아 수행된 것임 (2010-0021897)

으로부터 요청되는 논리 주소를 플래시 메모리의 물리 주소로 변환하고, 변환된 물리 주소에 데이터의 입출력을 수행한다.

본 논문에서는 새로운 버퍼 수준 관리 기법인 클린 블록 우선 (Clean Block First : CBF) 교체 기법을 연구하였다. 제안한 CBF 기법은 FTL의 삭제 연산 비용을 고려하여 삭제 블록 단위로 버퍼를 관리하고, 버퍼 교체 시에 클린(clean) 블록을 우선적으로 교체하여 플래시 메모리에 대한 삭제 연산 횟수를 최소화하였다. 버퍼 교체 시에 탐색하는 블록의 개수를 윈도우 크기라고 정의하였으며, 트레이스 기반의 시뮬레이션을 수행하여 윈도우 크기 변화에 따라 캐시 적중률과 삭제 연산 횟수를 분석하였다.

본 논문의 구성은 다음과 같다. 2장에서는 FTL과 기존의 버퍼 캐시 기법을 소개한다. 3장에서는 본 논문에서 제안하는 클린 블럭 우선 교체 기법을 설명하고, 4장에서 제안한 교체 알고리즘의 성능을 평가하고 실험 결과를 기술한다. 5장에서는 본 논문의 결론을 맺는다.

## 2. 관련 연구

### 2.1 FTL

FTL은 파일 시스템으로부터 요청되는 논리 주소를 플래시 메모리의 물리 주소로 변환하며 삭제 연산 문제를 해결하기 위해 타 위치 변경 방식을 사용한다 [3-5]. 가장 널리 사용되는 대표적인 FTL 기법은 데이터의 변경을 원래 데이터 블록에 바로 수행하지 않고 로그 블록에 수행하는 로그 블록 기법이다 [3, 5]. 로그 블록 FTL 기법은 로그 블록이 다 차게되면 원래 데이터 블록의 유효(valid) 페이지와 로그 블록의 유효 페이지를 병합(merge)하여 새 데이터 블록으로 복사하고, 이전 데이터 블록과 로그 블록에 대해 삭제 연상을 수행하여 가용 데이터 블록을 만든다. 이와 같은 병합 연산은 많은 페이지의 복사와 두 블록에 대한 삭제 연산을 수반하기 때문에 비용이 많이 드는 문제점을 갖고 있다. 만약 데이터 블록의 모든 페이지들이 순차적으로 변경된다면 로그 블록에 순차적으로 유효 페이지가 채워지므로 병합 연산 시에 로그 블록을 데이터 블록으로 전환(switch)만 하면 되므로 비용이 최소화된다.

### 2.2 플래시 메모리 기반의 버퍼 캐시 기법

#### 2.2.1 페이지 수준 기법

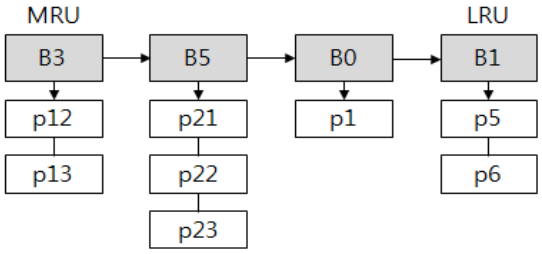
페이지 수준 기법은 버퍼를 플래시 메모리의 페이지 단위로 관리하며, 버퍼 교체 시에 페이지를 선택하여 교체시킨다 [6-9].

CFLRU (Clean First LRU)기법은 플래시 메모리의 삭제 연산을 줄이기 위해 제안된 버퍼 교체 알고리즘이다 [6]. CFLRU는 버퍼들을 최근에 참조된 버퍼 순으로 정렬한 LRU 리스트로 관리한다. CFLRU의 리스트는 LRU 리스트에서 상대적으로 최근에 참조된 페이지들을 가지고 있는 작업 영역과 상대적으로 오래 전에 참조된 페이지들을 가지고 있는 클린-우선 영역으로 나누어진다.

클린(clean) 페이지란 내용이 변경되지 않은 페이지를, 더티(dirty) 페이지는 내용이 변경된 페이지를 뜻한다. CFLRU는 버퍼 교체 시에 클린-우선 영역에서 LRU 리스트의 끝에서부터 시작하여 클린 페이지를 찾아 이를 교체한다. 만약 클린-우선 영역에 클린 페이지가 없다면 LRU 리스트의 맨 마지막 페이지를 교체한다. 이 외에도 페이지 수준 교체 기법으로는 CCF (Cold-Clean First) [8], ACR (Adaptive Cost-Aware Replacement) [9] 등이 있다.

#### 2.2.2 블록 수준 기법

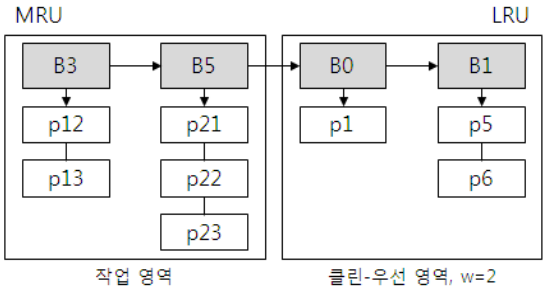
블록 수준 기법은 버퍼를 플래시 메모리의 삭제 블록 단위로 관리하며, 버퍼 교체 시에 블록을 선택하여 소속된 페이지를 모두 교체시킨다 [10,11]. (그림 2)은 블록 수준 기법의 버퍼 리스트의 구조를 보여준다. 블록 리스트는 최근에 참조된 블록 순으로 정렬한 LRU 리스트로 관리된다. 예를 들어, 블록 B3은 가장 최근에 참조된 블록이며 페이지 p12, p13이 참조되어 버퍼에 저장되어 있다.




(그림 2) 블록 수준 기법의 버퍼 리스트 구조

## 3. CBF (Clean Block First) 교체 기법

본 논문에서는 새로운 블록 수준 버퍼 관리 기법인 CBF(Clean-Block First) 교체 기법을 제안한다. 제안한 CBF 기법은 기존 블록 수준 기법처럼 플래시 메모리에서 읽은 페이지를 버퍼에 저장하며 같은 블록에 속한 페이지들을 그룹으로 묶고 블록 단위의 LRU 리스트로 관리한다.




(그림 3) CBF의 버퍼 리스트 구조

또한, (그림 3)에서 보는 것처럼 LRU 리스트를 작업 영역(working region)과 클린 우선 영역(clean-first region)

으로 나눈다. 작업 영역은 상대적으로 최근에 참조된 블록들을 가지고 있고, 클린 우선 영역은 상대적으로 오래 전에 참조된 블록들을 가지고 있다.

파일 시스템으로부터 읽기 요청이 도착하면, CBF는 해당 페이지가 버퍼에 있는 지를 검사하고 있다면 버퍼에서 읽기를 수행한다. 만약 해당 페이지가 버퍼에 없다면 새로 버퍼를 할당하고 플래시 메모리에서 페이지를 읽어 버퍼에 저장한 후 읽기를 수행한다. 새로 할당된 버퍼는 버퍼에 저장되는 페이지의 소속 블록의 리스트에 들어간다. 예를 들어, (그림 3)에서 페이지 p2가 새로 읽혀진 경우, p2를 저장한 버퍼는 p2의 소속 블록인 B0의 리스트에 들어간다. 또한, 참조된 블록 B0는 LRU 리스트의 맨 앞으로 이동하게 되며, 블록에 소속된 페이지는 함께 이동한다.

파일 시스템으로부터 쓰기 요청이 도착하면, CBF는 해당 페이지가 버퍼에 있는 지를 검사하고 있다면 버퍼에 쓰기를 수행한다. 만약 해당 페이지가 버퍼에 없다면 새로 버퍼를 할당하고 버퍼에 쓰기를 수행한다. 새로 할당된 버퍼는 버퍼에 저장되는 페이지의 소속 블록의 리스트에 들어가며, 참조된 블록은 LRU의 리스트의 맨 앞으로 이동하게 된다.

버퍼 캐시가 다 소모되면, CBF는 클린 우선 영역에서 LRU 위치부터 탐색하여 클린 블록을 찾아 이를 희생 블록으로 선택한다. 만약 클린 우선 영역에 클린 블록이 없다면 페이지가 가장 많은 블록을 희생 블록으로 선택한다. 클린 블록은 버퍼에 있는 소속 페이지가 모두 클린 페이지인 블록을 의미한다.

클린 블록이 희생 블록으로 결정된 경우, 블록의 페이지가 저장된 버퍼는 플래시 메모리에 쓰기 연산을 하지 않고 바로 가용 버퍼로 사용되기 때문에 플래시 메모리의 쓰기 연산 및 삭제 연산 횟수를 줄일 수 있다. 희생 블록이 클린 블록이 아닌 경우, CBF는 BPLRU와 같이 페이지 패딩 기법을 사용하여 하부의 FTL에서 야기되는 병합 비용을 줄인다.

LRU 리스트에서 클린 우선 영역에 속한 블록의 개수를 윈도우 크기라 정의하고 이를 w로 표기한다. 예를 들어, (그림 3)에서 클린 우선 영역에 속한 블록은 두 개(B0, B1)이므로 w는 2가 된다. w의 크기는 알고리즘의 성능에 영향을 미친다. 즉, w의 크기가 증가하면 캐시 적중률이 감소하고, w의 크기가 감소하면 클린 블록을 찾을 확률이 낮아져 플래시 메모리에 대한 삭제 연산 횟수가 증가할 수 있다. 따라서 적절한 윈도우의 크기를 사용하는 것이 중요하다.

## 4. 실험 결과

### 4.1 실험 환경

본 논문에서 제안한 CBF의 성능을 검증하기 위하여 트레이스 기반의 시뮬레이터를 구현했다. 시뮬레이터에서 사용한 주요 파라미터는 <표1>과 같다.

<표 1> 시뮬레이터의 파라미터

| 파라미터 | 총 참조 수 |
|---|---|
| 페이지 크기 | 4 KB |
| 블록 당 페이지 수 | 64 |
| 블록 개수 | 65536 |

트레이스 수집을 위해 마이크로소프트 윈도우 기반의 노트북 PC에서 웹브라우저, 미디어 플레이어와 같은 여러 응용 프로그램을 1주일간 실행하며 2개의 트레이스 T1과 T2를 수집하였다. <표 2>는 트레이스 T1과 T2의 특징을 보여주고 있다.

표에서 읽기/쓰기 비율“x%/y%”은 전체 연산중에서 x%는 읽기 연산을 나머지 y%는 쓰기 연산을 수행하는 것을 의미한다.

<표 2> PC 트레이스

| 유형 | 총 참조 수 | 읽기/쓰기 비율 |
|---|---|---|
| T1 | 706,833 | 67%/33% |
| T2 | 1,481,739 | 55%/45% |

### 4.2 실험 결과

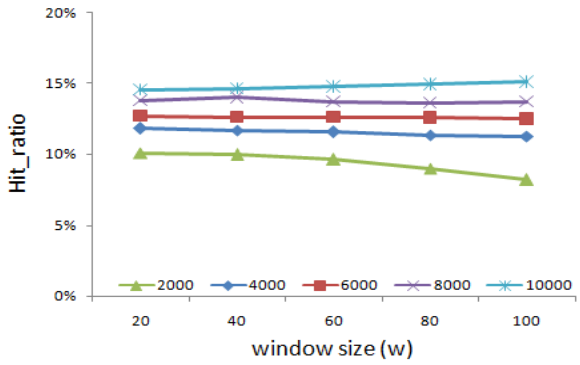
버퍼 캐시의 크기를 2,000개에서 10,000개까지 변화시키면서 버퍼 캐시 적중률과 삭제 연산 횟수를 측정하였다. 또한 윈도우의 크기(w)의 변화에 따른 성능을 확인하기 위하여 w의 크기를 20부터 100까지 변화시키면서 측정하였다.

(그림 4)와 (그림 5)는 각각 트레이스 T1과 T2에 대하여 버퍼 크기와 윈도우의 크기(w) 변화에 따른 버퍼 캐시 적중률과 삭제 연산 횟수를 보여준다.

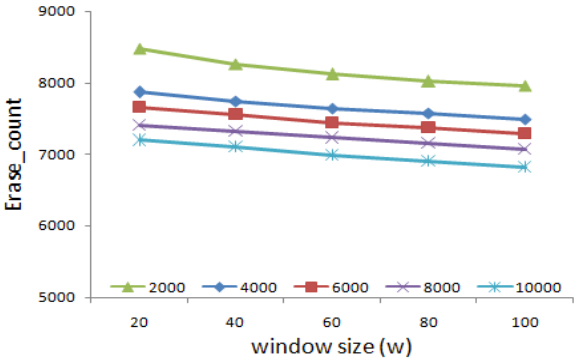
그림의 (a)는 버퍼 캐시 적중률을 보여주고 있다. 버퍼의 크기가 작을 때 (예: 2000), w의 크기가 커질수록 캐시 적중률이 낮아짐을 알 수 있다. 그러나, 버퍼의 크기가 크면 (예: 4000 이상), 캐시 적중률은 w의 크기와 큰 관련이 없음을 알 수 있다. 그림의 (b)는 삭제 연산의 횟수를 보여주고 있다. w 크기가 커질수록 플래시 메모리에 대한 삭제 연산의 횟수가 줄어든다. 종합해보면, 제안한 CBF 기법에서 적절한 윈도우 크기를 정하려면 버퍼 캐시 적중률과 플래시 메모리에 대한 삭제 연산 횟수를 모두 고려하여 결정해야 함을 알 수 있다.

## 5. 결론

본 논문에서는 플래시 메모리 기반 저장장치를 사용하는 운영체제에서 버퍼 캐시의 교체 기법을 연구하였다. 플래시 메모리는 삭제 연산으로 인해 성능이 저하되며 그 횟수가 제한적인 문제점이 있다. 따라서, 버퍼 캐시의 교체 기법은 플래시 메모리의 삭제 연산 횟수를 최소화하는 한편, 버퍼 캐시의 적중률을 최대화해야 한다. 제안한 CBF 기법은 삭제 블록 단위로 버퍼를 관리하고, 버퍼 교

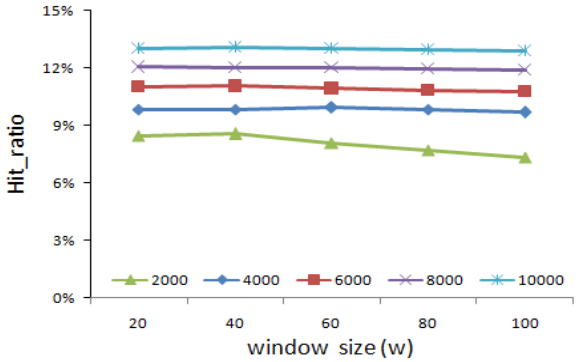



(a) 버퍼 캐시 적중률

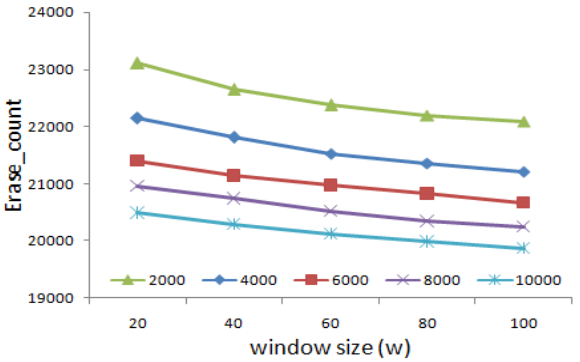



(b) 삭제 연산 횟수

(그림 4) 버퍼 캐시 적중률과 삭제 연산 횟수: T1




(a) 버퍼 캐시 적중률




(b) 삭제 연산 횟수

(그림 5) 버퍼 캐시 적중률과 삭제 연산 횟수: T2

체 시에 클린 블록을 우선적으로 교체하여 플래시 메모리에 대한 삭제 연산 횟수를 최소화하였다. 트레이스 기반의 시뮬레이션을 통해 캐시 적중률과 삭제 연산 횟수를 분석하였다.

## 참고문헌

[1] Samsung Electronics, K9XXG08UXM.1G x 8 Bit/2G x 8 bit NAND Flash Memory

[2] Gal and S. Toledo, "Algorithms and data structures for flash memories," ACM Computing Surveys, vol. 37, no. 2, 2005.

[3] J. Kim, J. Kim, S. Noh, S. Min, and Y. Cho, "A space-efficient flash translation layer for compactflash systems," IEEE Transactions on Consumer Electronics, vol. 48, no. 2, 2002.

[4] Gupta, Y. Kim, and B. Urgaonkar, "DFTL: a flash translation layer employing demand-based selective caching of page-level address mapping," in Proc. of International Conference on Architectural Support for Programming Languages and Operating Systems, 2009, pp.229-240.

[5] Y. Ryu, "SAT: switchable address translation for flash memory storages," in Proc. of IEEE Computer Software and Applications Conference (COMPSAC), Jul. 2010.

[6] S. Park, D. Jung, J. Kang, J. Kim, and J. Lee. "CFLRU: a replacement algorithm for flash memory", in Proc. of the 2006 International Conference on Compilers, Architecture and Synthesis for Embedded Systems, pp. 234-241, 2006.

[7] Y. Yoo, H. Lee, Y. Ryu, and H. Bahn, "Page replacement algorithms for NAND flash memory storages," in Proc. of International Conference on Computational Science and its Applications, pp. 201-212, 2007.

[8] Z. Li, P. Jin, X. Su, K. Cui, and L. Yue, "CCF-LRU: A new buffer replacement algorithm for flash memory," IEEE Transactions on Consumer Electronics, Vol. 55, No. 3, pp. 1351-1359, August, 2009.

[9] X. Tang and X. Meng, "ACR: an Adaptive Cost-Aware Buffer Replacement Algorithm for Flash Storage Devices," IEEE Conference on Mobile Data Management, pp. 33-42, 2010.

[10] H. Jo, J. Kang, S. Park, and J. Lee, "FAB: Flash aware buffer management policy for portable media players," IEEE Transactions on Consumer Electronics, Vol. 48, No. 2, pp. 485-493, 2006.

[11] H. Kim and S. Ahn, "BPLRU: A buffer management Scheme for improving random writes in flash storage," in Proc. of USENIX Conference on File and Storage Technologies(FAST),pp.239-252,2008.