Journal of Digital Contents Society Vol. 12 No. 2 Jun. 2011(pp. 141-150)

SSD를 위한 비대칭 버퍼 관리 기법

정호영*, 강수용**, 차재혁***

요 약

최근 다양한 기기에서 플래시 메모리 저장장치인 SSD가 활용되고 있다. SSD 기반 시스템에서 기존 하드 디스크 기반 버퍼 교체 알고리즘은 플래시 메모리의 특성을 고려하지 않고 이는 시스템의 성능 저하의 원인이 된다. 본 논문에서는 SSD의 특성을 고려하여 읽기 버퍼와 쓰기 버퍼를 분리하고 각각의 버퍼에 서로 다른 크기의 교체 단위와 서로 교체 알고리즘을 적용하는 ABM (Asymmetric Buffer Management) 정책을 제안한다. 추가적으로 제안한 정책을 보완하기 위해 쓰기 교체 지연 정책, 동적 크기 적응화 알고리즘을 적용하였다. 제안한 ABM 정책은 효과적으로 성능을 향상시키는 것으로 나타났으며, 특히 여러 알고리즘 중 가장 성능이 좋은 ABM-LRU-CLC의 경우 기존의 LRU에 비해 최대 32% 성능이 향상되는 것으로 나타났다.

An Asymmetric Buffer Management Policy for SSD

Hoyoung Jung*, Sooyong Kang**, Jaehyuk Cha***

Abstract

Recently the Solid State Drive (SSD) is widely used for storage system of various mobile devices. In this case, existing buffer replacement algorithms based on the hard disk do not consider characteristics of flash memory, so it caused performance degradation of the system. This paper proposes a novel buffer replacement policy called ABM (Asymmetric Buffer Management) policy. ABM policy separates read and write buffer space and applies different replacement unit and replacement algorithm for each buffer. In addition, write buffer delay scheme and dynamic size adaptation algorithm is applied for better performance. ABM outperforms other replacement policies, especially ABM-LRU-CLC shows 32% better performance than normal LRU policy.

Keywords : flash memory, buffer replacement, storage system

1. 서론

최근 플래시 메모리는 다양한 기기에서 하드 디스크를 대치하고 있다. 이는 플래시 메모리가 가지고 있는 다양한 장점들 때문이다.

기존의 하드 디스크는 기계적 장치로 인해, 속도의 한계를 가지고 있고 이는 시스템 병목현상의 원인이 되어왔다. 이러한 기존의 하드 디스크의 결점을 보완하기 위해 여러 IO 계층이 존재하며, 대표적인 버퍼 캐시 계층은 디스크 데이터의 일부를 메모리에 저장하여 디스크의 느린 응답시간을 감추는데 사용된다. 버퍼 캐시는 디스크에 비해 크기가 작으므로, 이를 효율적으로 사용하기 위해서 다양한 버퍼 교체 알고리즘이 연구되어 왔다.

최근에는 많은 저장장치가 플래시 메모리 저장장치인 SSD (Solid State Disk)로 대체됨에 따라 하드 디스크의 특성을 고려하던 기존의 버퍼 교체 알고리즘을 대신하여 플래시 메모리의 특성을 고려하는 버퍼 교체 알고리즘에 대한 연구도 활발히 진행되고 있다. 적중률을 성능을 척

※ 제일저자(First Author) : 정호영
접수일자:2011년04월18일, 수정일자:2011년05월13일
심사완료:2011년05월23일
* 한양대학교 박사과정
horong@hanyang.ac.kr
** 한양대학교 정보통신학과 교수
*** 한양대학교 정보통신학과 교수

도로 삼고 이를 향상시키려는 LRU, ARC (Adaptive Replacement Cache)[1] 혹은 LIRS (Low Inter-reference Recency Set)[2] 등 의 하드디스크 기반 버퍼 교체 정책과 달리 플래시 메모리 기반 교체 정책은 플래시 메모리의 느린 쓰기 및 지우기 성능을 우선적으로 고려한다.

SSD를 위한 버퍼 교체 정책으로는 CF-LRU (Clean First LRU)[3], LRU-WSR (LRU Write Sequence Reordering)[4], FAB (Flash Aware Buffer)[5], CLC (Cold and Largest Cluster)[6] 등이 있다. 이들 정책은 각각 다양한 방법을 시도하여 쓰기 / 지우기의 감소를 목표로 하고 있다. 그러나 이들은 읽기 패턴보다는 쓰기 패턴을 집중적으로 고려하여 일부 경우에만 성능이 향상되는 경향을 가지고 있다.

기존의 버퍼 교체 알고리즘들과 이들의 트레이스 데이터를 분석한 결과를 토대로, 본 논문은 버퍼 캐시에서 읽기 버퍼와 쓰기 버퍼를 분리하고 이를 따로 관리하는 Asymmetric Buffer Management policy (ABM)를 제안한다. ABM은 읽기 버퍼와 쓰기 버퍼를 나누고 각각 의 버퍼에 서로 다른 크기의 교체 단위와 교체 알고리즘을 적용하였다. 분리된 읽기 버퍼는 적중률을 중시하는 하드 디스크 기반 버퍼 교체 알고리즘을 적용하고, 쓰기 버퍼는 플래시 메모리 기반 버퍼 교체 알고리즘을 적용하였다. 제안한 정책은 플래시 메모리의 특성을 잘 반영하고, 읽기 및 쓰기 참조 패턴 모두에 효과적이다. 추가적으로 쓰기 교체 지연 정책과 동적 크기 적응화 정책을 적용하여 읽기 쓰기 버퍼 분리에 따라 발생하는 성능저하를 극복하였다. 이를 입증하기 위해 읽기 버퍼와 쓰기 버퍼에 여러 가지 교체 알고리즘을 적용하여 실험하였다. 특히 여러 알고리즘 중 읽기 버퍼에 LRU를 적용하고 쓰기버퍼에 CLC 를 적용한 ABM-LRU-CLC의 경우 다른 모든 알고리즘보다 우수한 성능을 보이며, 기존의 LRU에 비해 최대 32% 성능이 향상되는 것으로 나타났다.

본 논문의 이후 구성은 다음과 같다. 2장에서는 관련 연구를 설명한다. 3장에서는 ABM 정책을 설명한다. 4장에서는 ABM 정책의 성능을 평가하고, 5장에서는 결론과 향후 과제를 기술하였다.

2. 관련 연구

2.1 플래시 메모리 저장장치

NAND 플래시 메모리를 하드 디스크를 대신하는 저장장치로 사용하기 위해서는 같은 위치에 재쓰기(in-place-update)가 불가능한 특성과 지우기의 단위가 페이지의 집합인 블록인 특성을 숨기기 위한 매핑 계층이 필요하다. 여러 가지 매핑 방법 중에서 SSD에서 대표적으로 사용하는 방법은 FTL (Flash Translation Layer)을 사용하는 것이다[7].

FTL은 플래시 메모리 칩과 운영체제의 중간에 위치하며, 플래시 메모리를 디스크 드라이브처럼 보이게 하는 역할을 담당한다. FTL로 인해, 플래시 메모리 저장장치는 논리적으로 재쓰기가 가능한 것처럼 보이고, 지우기는 완전히 숨겨진다. 이를 위해 FTL은 내부적으로 논리 페이지 번호(logical page number)와 이에 대응되는 물리 페이지 번호(physical page number) 를 매핑 테이블에 저장한다. 이러한 구조에서 논리적인 쓰기 연산은 결국 블록 지우기 연산을 발생시키며 이는 다른 연산에 비해 매우 느린 연산이다. 따라서 플래시 메모리 저장장치의 성능 향상을 위해서는 호스트 시스템에서 플래시 메모리로 요청하는 쓰기 연산의 횟수를 감소시키는 것이 필요하다.

FTL의 내부적인 알고리즘 등으로 인해 임의쓰기(random write)는 순차쓰기(sequential write)보다 더 많은 쓰기와 지우기를 발생시키게 되며, 그 결과 SSD는 임의쓰기에 취약점을 가지게 된다. 표1 은 SSD의 성능을 간략히 나타낸 것으로 임의쓰기의 성능이 타 연산에 비해 매우 낮은 것을 알 수 있다[8]. 최근에는 SSD의 단점을 극복하기 위해 멀티 채널과 쓰기 버퍼 등을 활용하고 있으나 SSD는 여전히 랜덤 쓰기에 취약한 성능을 보인다. 따라서 SSD의 성능을 높이기 위해서는 랜덤 쓰기 요청을 줄이는 것이 필요하다.

<표 1> SSD의 성능

접근 방식	MB /s	접근 방식	IOPS
순차읽기	250	4K 읽기	35,000
순차쓰기	170	4K 쓰기	3,300

2.2 하드디스크 기반 버퍼 교체 정책

운영체제에서 버퍼 캐시 정책은 디스크 블록의 일부분을 저장하여 물리적인IO 요청을 감소시키려 한다. 버퍼 캐시의 크기는 디스크에 비해 매우 작기 때문에 IO 성능을 향상시키기 위해 다양한 버퍼 교체 알고리즘이 연구되어 왔다.

ARC 알고리즘은 LRU를 개선한 버퍼 교체 알고리즘이다. ARC는 버퍼에 있는 페이지뿐만 아니라, 과거에 교체된 페이지의 정보를 저장하고 이를 활용한다. 하나의 LRU 리스트를 사용하는 LRU와는 달리 ARC는 T와 B로 불리는 두 개의 리스트를 사용함으로써, 최신성(recency) 뿐만 아니라 참조빈도(frequency)를 고려하여 교체 페이지를 선정한다.

LIRS는 LRU를 개선한 또 다른 버퍼 교체 정책이다. LIRS는 가변길이의 LRU리스트를 사용하여 페이지를 IR (Inter-reference Recency) 값에 의해 두 그룹으로 분류한다. LIR (Low IR) 페이지는 최근에 자주 사용된 페이지이며, HIR (High IR) 페이지는 참조빈도가 낮은 페이지이다. LIRS는 가장 오래된 HIR 페이지를 항상 교체 대상으로 선정한다.

기존의 버퍼 교체 정책은 모두 읽기 / 쓰기 비용이 같은 하드 디스크를 가정하므로 적중률을 높이기 위해 노력한다. 그러나 SSD는 언급한 바와 같이 쓰기가 매우 느리며, 추가적으로 지우기의 발생 가능성을 내포하고 있어, 기존의 적중률만을 고려한 버퍼 교체 정책은 플래시 메모리 저장장치에 적합하지 않다.

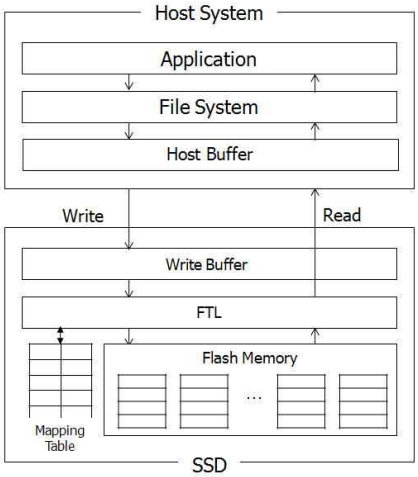
2.3 SSD를 위한 버퍼 교체 알고리즘

그림 1의 시스템 계층에서 플래시 저장장치를 위한 버퍼 교체 정책들은 다음과 같은 사항을 고려해야 한다.

(1) 지우기를 감소시키기 위해 쓰기 접근 횟수를 감소시킨다. CF-LRU, WSR이 이에 해당한다. 쓰기 접근의 감소를 통해 간접적으로 지우기를 감소시키는 이유는 그림 1에서 보이는 바와 같이, 호스트 시스템의 버퍼 계층에서는 지우기 가 숨겨지기 때문이다.

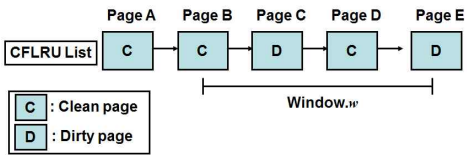
(2) 임의쓰기를 순차 쓰기로 변형하여 FTL이 수행하는 지우기 횟수를 감소시키는 방법, FAB가 이에 해당한다.

또한 그림 1의 SSD에서 쓰기 버퍼를 관리하기 위한 정책도 최근에 제안되었다. CLC가 이에 해당하며, 이들 은 지우기와 쓰기 발생을 최소화하여, 임의쓰기 성능을 향상시키는 데 주력하고 있다.



(그림 1) SSD를 사용하는 시스템

CF-LRU (Clean First LRU) 는 플래시 메모리 쓰기 비용을 고려한 페이지 교체 정책이다. 그림 2는 CF-LRU의 알고리즘을 설명한 것이다. 가장 오래된 페이지는 E이고 가장 최근에 참조된 페이지는 A일 때, LRU 알고리즘은 E를 교체 페이지로 선정한다. 그러나 CF-LRU는 그림 2의 윈도우 w 안의 더티 페이지는 w가 더티(Dirty) 페이지로 가득 찰 때까지, 교체되지 않는다. 그림 2의 경우는 다음으로 가장 오래된 페이지이고 클린(Clean) 페이지인 D가 교체 대상이 된다. 클린 페이지를 더티 페이지보다 우선적으로 교체함에 따라 플래시 메모리에 요청하는 쓰기 횟수가 감소하고 전체 IO 성능은 향상될 수 있다. 그러나 CF-LRU는 적중률의 저하로 인한 성능감소의 문제점을 가지고 있다.



(그림 2) CF-LRU 교체정책 예제[3]

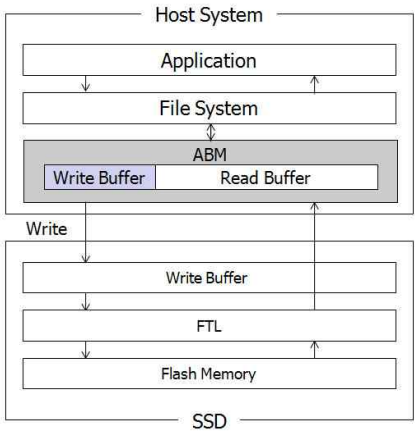
[4] 에서 제안한 LRU-WSR은 적중률의 저하를 막기 위해, 재기회(second chance) 알고리즘에 기반한 콜드 감지 알고리즘을 적용하였다. LRU-WSR은 적용한 콜드 감지 알고리즘을 통해 재참조될 가능성이 있는 더티 페이지들만 교체를 지연함으로써, 효과적으로 쓰기 연산을 감소시키고 적중률의 감소를 방지한다. 그러나 LRU-WSR은 쓰기 시퀀스가 변경됨으로 인해, 랜덤 쓰기가 더 많이 발생하는 문제점을 가지고 있다.

FAB 알고리즘은, 순차접근이 많은 플래시 메모리 저장장치 기반 PMP를 위한 버퍼 교체 알고리즘이다. FAB의 교체 단위는 페이지가 아니라 페이지의 집합인 블록이다. 또한 가장 적은 액세스 가능성을 가진 페이지를 선택하는 LRU와는 달리 가장 많은 페이지를 포함한 블록 전체를 교체 대상으로 선택한다. 만약 교체 대상 블록이 2개 이상일 경우, 이들 중 더 오래 전에 참조된 블록을 교체 대상으로 선정한다.

[6]의 CLC (Cold and Largest Cluster) 알고리즘은 지우기와 블록 병합 연산을 줄이기 위한 플래시 메모리 쓰기 버퍼 교체 알고리즘이다. CLC는 교체 대상 블록으로 오랫동안 참조가 되지 않고, 더 많은 더티 페이지들을 포함한 블록을 교체 대상으로 선정한다. CLC는 블록의 최신성과 클러스터링된 블록의 크기를 모두 고려하여, 효과적으로 지우기 연산과 블록 병합 연산을 감소시켰다.

3. ABM (Asymmetric Buffer Management) 정책

본 연구에서 제안하는 ABM 정책은 SSD를 사용하는 시스템의 버퍼 캐시 계층에 적용되는 정책이다. 그림 3은 ABM이 적용되는 시스템의 구조를 나타낸 것이다. 그림 3처럼 ABM은 읽기 버퍼와 쓰기 버퍼를 분리하고, 각기 다른 버퍼 단위와 교체 정책을 적용하는 정책으로, 하위 계층인 SSD의 쓰기버퍼와 FTL의 특성을 고려하여 시스템의 성능을 향상시키는 것을 목표로 한다.



(그림 3) ABM이 적용된 SSD 구조

3.1 읽기/ 쓰기 버퍼 분리

ABM은 그림 3과 같이 읽기/ 쓰기 버퍼를 분리하며, 이는 다음과 같은 이유 때문이다. 첫째로, 그림 3의 디스크에 존재하는 FTL과 쓰기 버퍼를 효과적으로 활용하기 위해서 버퍼 계층에서 내보내는 쓰기 요청은 최대한 순차적이어야 한다. 표 1에서 보는 것처럼 임의쓰기의 성능이 매우 나쁜 SSD의 성능을 향상시키기 위해서 임의쓰기를 순차쓰기로 변형하는 것은 SSD의 버퍼 교체 정책에는 필수적이다. 둘째로, 과도한 적중률 저하를 막아야 한다. 현재 존재하는 FAB와 같은 버퍼 교체 알고리즘은 쓰기와 지우기 횟수를 감소시키고 임의쓰기를 순차쓰기로 변형했음에도 불구하고 적중률의 저하로 인해 성능이 감소하는 경우가 발생한다. 이는 읽기 요청에 대해서 SSD 버퍼 알고리즘이 기존의 하드디스크 기반 알고리즘과 달리 적중률을 고려하지 않기 때문이다. 다시 말해서 SSD의 경우 쓰기 버퍼는 순차쓰기를 증가시키기 위해 공간 지역성을 우선적으로 고려해야 하고, 읽기 버퍼는 적중률을 높이기 위해 시간 지역성을 우선적으로 고려해야 한다. 이에 따라 제안되는 ABM의 특성은 다음과 같다.

(1) 읽기 버퍼의 교체단위를 페이지, 쓰기 버퍼의 교체 단위를 페이지의 집합인 블록으로 한다.

(2) 읽기 버퍼는 적중률을 최대로 하기 위해 시간 지역성을 고려하는 하드디스크 기반 버퍼

교체 알고리즘을 적용시킨다.

(3) 쓰기 버퍼는 임의쓰기와 지우기 연산을 최소화하기 위해 공간 지역성을 고려하는 SSD 기반 버퍼 교체 알고리즘을 적용시킨다.

실험을 위해서 읽기 버퍼 알고리즘으로는 가장 널리 사용되는 LRU와 ARC, 그리고 LIRS를 선택하였다. 쓰기 버퍼 알고리즘으로는 블록 단위 버퍼 관리 정책이고, 쓰기 순서의 변경을 통해 순차쓰기를 우선적으로 발생시키는 FAB와 CLC를 선택하였다.

부가적으로 ABM은 읽기 요청에 대해 읽기 버퍼에 빈 공간이 없고 쓰기 버퍼에만 빈 공간이 존재한다면 쓰기 버퍼에서 임시로 공간을 할당받을 수 있다. 해당 페이지는 차후 쓰기 버퍼에 빈 공간이 없을 경우 우선적으로 교체 대상이 된다. 마찬가지로 쓰기 요청에 대해서도 빈 공간이 없을 경우 읽기 버퍼에서 할당받을 수 있다. 언급한 정책은 읽기나 쓰기 한 종류의 요청이 집중적으로 올 경우 다른 버퍼에 여유공간이 있음에도 불구하고, 버퍼가 완전 분리되어 있어 발생하는 성능하락을 막기 위한 방법이다. 아래의 알고리즘은 읽기 요청에 대한 ABM 정책을 유사 코드(pseudo code)로 나타낸 것이다.

```
x: Request page
w : allocated write buffer size
W: max write buffer size

if (x is read request)
  if (free space)
    read x and put it into buffer
  else //buffer is full
  if (w > W) //some read buffer is used for
write buffer
    run write buffer replacement algorithm
  else
    run read buffer replacement algorithm
allocate x read x from flash memory
if(x is write request)
…
```

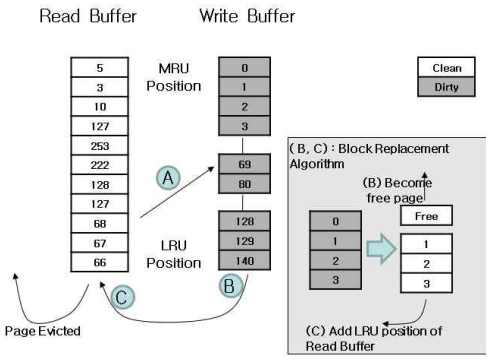
3.2 ABM-LRU-FAB

ABM-LRU-FAB는 ABM의 기본적인 특징을 가장 잘 설명해주는 알고리즘이다. 그림 4와 같이 ABM-LRU-FAB의 읽기 버퍼로는 페이지 단위 LRU를 사용하며, 쓰기 버퍼로는 블록 단위 FAB를 사용한다. 읽기버퍼와 쓰기버퍼의 총합은 전체 버퍼 크기와 같으며, 각 버퍼의 크기는 초기화 시점에 정해진다. 읽기 버퍼는 LRU 알고리즘에 의해 최신성을 고려하여 가장 오래 전에 참조된 페이지, 즉 그림 4에서 가장 바닥에 위치한 페이지가 교체된다. 읽기 버퍼에는 변경이 없는 클린(clean) 페이지들만 위치하므로 교체시 부가적인 비용이 발생하지 않는다. 쓰기 버퍼는 FAB 알고리즘에 의해 페이지들을 블록 단위로 묶어서 관리하고, 교체시에 가장 많은 페이지를 포함하는 블록이 교체된다. 따라서 그림 4에서는 4개의 페이지를 포함하는 블록 0번(페이지 0, 1, 2, 3의 집합) 이 교체된다. 부가적으로 읽기와 쓰기 버퍼의 완전 분리로 인한 성능 하락을 막기 위해 그림 4의 A, B, C의 세 가지 메커니즘을 사용하였다. 먼저 읽기 버퍼에 있던 페이지가 쓰기요청으로 인해 더티 페이지가 될 경우, 해당 페이지는 읽기 버퍼에서 쓰기 버퍼로 이동한다 (그림 4의 A).

또한 쓰기 버퍼에서 많은 페이지가 교체되어 쓰기 버퍼 적중률이 저하하는 현상을 막기 위해 쓰기 버퍼 교체 알고리즘을 적용하였다 (그림 4의 B와 C). 먼저 쓰기 교체가 발생시 0번 블록이 교체될 때는, 4개의 더티 페이지가 교체되면서 순차쓰기가 발생하며, 이중 한 페이지만 프리(free) 페이지가 된다(그림 4의 B). 마지막으로 나머지 3 개의 페이지는 읽기 버퍼의 LRU 위치에 추가되는데 이는 작은 버퍼 공간을 최대한 활용하여 재참조를 높이기 위함이다(그림 4의 C). 기술한 알고리즘을 통해서 ABM-LRU-FAB는 클린 페이지들에 대해서는 LRU 와 같은 히트율을 보장하고, 반면 더티 페이지들에 대해서는 순차쓰기를 보장한다.

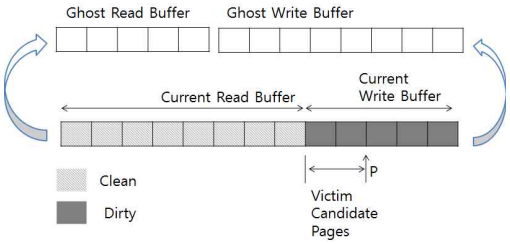
3.3 동적 크기 적응화 알고리즘

읽기 버퍼와 쓰기 버퍼를 분리하는 ABM은 이를 위해 시스템에서 최적 읽기/ 쓰기 버퍼를 수동으로 지정해 주어야 하는 문제점을 가지고 있다. 이 문제를 개선하기 위해 본 논문에서는 동적 크기 적응화 알고리즘을 제안하였다. 아래 그림 5는 동적 크기 적응화 알고리즘을 위한 자료 구조를 나타낸다.



(그림 4) ABM-LRU-FAB의 구조

그림 5와 같이 동적 적응화를 위해 고스트 읽기/쓰기 버퍼와 포인터 P를 추가하였다. 고스트 버퍼는 실제 데이터가 아닌 교체된 페이지의 정보만을 저장하는 자료구조로 매우 작은 비용으로 교체된 페이지의 정보를 저장한다. 포인터 P는 교체 대상 버퍼를 가리키는 포인터로 그림에서는 쓰기 버퍼가 교체 대상 버퍼가 된다. P는 고스트 버퍼 히트시에 그 위치가 단계별로 이동하는데, 읽기 고스트 버퍼 히트시에는 쓰기 버퍼 쪽으로 이동하게 되고, 쓰기 고스트 버퍼 히트시에는 읽기 버퍼 쪽으로 이동한다. 포인트의 위치 변화로 인해 ABM은 응용의 특성에 따라 각 버퍼의 크기가 동적으로 변화한다.



(그림 5) 동적 크기 적응화 알고리즘

4. 성능 평가

본 장에서는 각각의 ABM 알고리즘들에 대한 적중률, 쓰기 작업의 횟수와 수행시간을 비교하였다. 비교를 위해, 세 종류의 트레이스를 수집하고 이를 기반으로 시뮬레이션을 수행하였다. 각각의 트레이스는 순차접근, 임의 접근, 반복 패턴의 조합으로 이루어져 있다. 또한, 각 트레이스의 쓰기 로컬리티 값 또한 이는 쓰기 로컬리티가 플래시 메모리 저장장치의 성능에 큰 영향을 끼치기 때문이다.

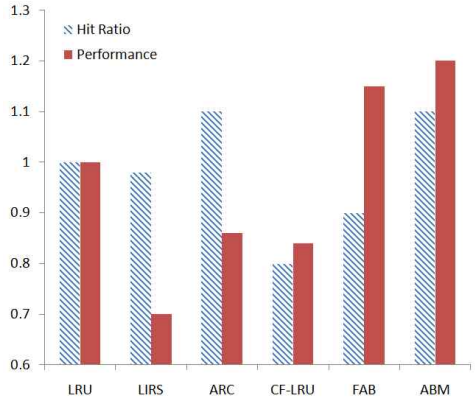
4.1 시뮬레이션 워크로드

실험에 사용된 GCC, Viewperf와 Cscope 트레이스는 보다 다양한 실험을 하기 위해 본 실험에서 사용한 트레이스들이다. 이 트레이스는 리눅스의 strace 프로그램을 변형하여 추출된 것이다[9]. GCC 트레이스는 리눅스 커널 2.4.20 소스를 구축하면서 얻은 트레이스이며 반복적인 짧은 읽기, 쓰기와 긴 순차읽기로 이루어져 있다. Viewperf 트레이스는 그래픽 성능을 측정하는 SPEC 벤치마크의 트레이스이다. Viewpef 트레이스의 경우 쓰기 접근의 비율이 매우 낮은데, 이를 통해 WSR이 쓰기가 작은 경우의 성능을 확인하고자 실험하였다. Cscope 트레이스는 리눅스 커널 소스를 검사하여 얻은 트레이스로 매우 높은 읽기 / 쓰기 지역성을 가지고 있다.

기존의 버퍼교체정책은 쓰기 지역성과 접근 방식에 따라 각각 다른 성능을 보여준다. 본 연구에서는 다양한 접근 패턴과 다양한 쓰기 로컬리티에 대해 실험을 수행함으로써, 제안한 ABM 정책이 기존의 정책과는 달리 거의 모든 경우에 효과적으로 동작함을 보였다.

4.2 적중률

그림 6은 GCC 트레이스 상에서 버퍼 크기 1500 (12MB) 인 경우 기존의 알고리즘과 ABM-LRU-CLC의 적중률과 성능을 나타낸 것이다. 적중률과 성능 모두 LRU를 기준으로 정규화되었다. 그림 6에서 볼 수 있는 것처럼, SSD는 하드디스크와 달리 적중률과 성능이 비례하지 않는다. 이 결과를 통해 알 수 있는 것은 적중률은 SSD를 위한 버퍼 교체 알고리즘의 성능 평가의 척도로 사용되기 적합하지 않다는 사실이다. 본 논문에서는 따라서 각 알고리즘의 정확한 성능을 측정하기 위해 총 수행시간을 사용하였다. 수행시간의 경우, 읽기와 쓰기, 지우기의 비용을 모두 고려한 결과가 측정되므로 적중률보다 정확한 성능평가의 척도라 할 수 있다.



(그림 6) 정규화된 적중률과 성능

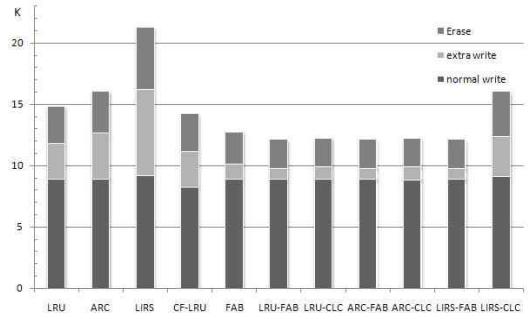
4.3 쓰기 및 지우기 비용

그림 7, 8, 9는 여러 트레이스 상에서 버퍼 교체 알고리즘들의 쓰기와 지우기에 소모된 비용을 나타낸 것이다.

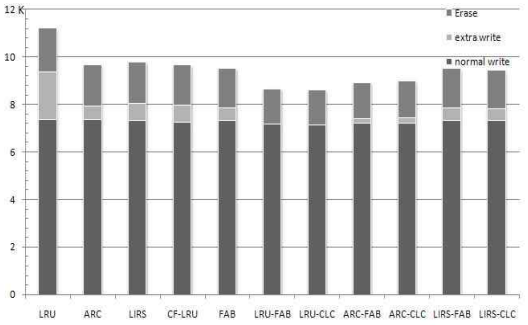
그림에서 쓰기 비용은 더티 페이지가 버퍼에서 교체되어 나갈 때 발생하는 쓰기의 비용을 합산한 것이고, 추가 쓰기 비용은 플래시 메모리 저장장치 내부에서 별도의 쓰기가 추가적으로 발생할 때 발생하는 비용을 나타낸 것이다. 지우기 비용은 모든 쓰기 과정에서 발생하는 지우기 비용을 합산하여 나타내었다.

그림 7, 8, 9에서 전체적으로 볼 수 있는 것처럼, ABM알고리즘(오른쪽 6개)들은 기존의 알고리즘들(왼쪽 5개)에 비해 효과적으로 쓰기 비용이 감소하는 것으로 나타났다. 이들은 쓰기 페이지에 대한 재참조로 인해 쓰기 요청 횟수를 줄였을 뿐만 아니라, 임의쓰기의 감소 및 순차쓰기의 증가로 지우기와 추가쓰기 비용도 감소됐음을 각 그래프를 통해 볼 수 있다.

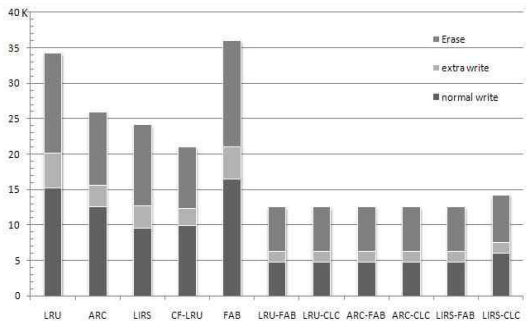
그림 7, 8 ,9의 그래프를 통해 ABM 알고리즘이 효과적으로 쓰기와 지우기를 감소시키며, 추가쓰기 비용도 감소시키는 것을 볼 수 있다. 또한 그림 8과 같이 쓰기 비율이 읽기에 비해 매우 적은 경우에도 쓰기 비용 이득이 다소 있는 것으로 나타났다. 쓰기 비용의 감소 측면으로 봤을 경우에 가장 효과적인 알고리즘은 ABM-LRU-FAB인 것으로 나타났으며 기존의 LRU에 비교해 봤을 경우 81%의 쓰기 감소를 보인다.



(그림 7) GCC의 트레이스 쓰기 및 지우기



(그림 8) Viewperf 쓰기 및 지우기

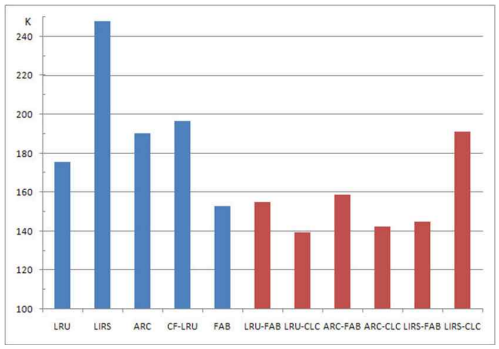


(그림 9) Cscope 쓰기 및 지우기

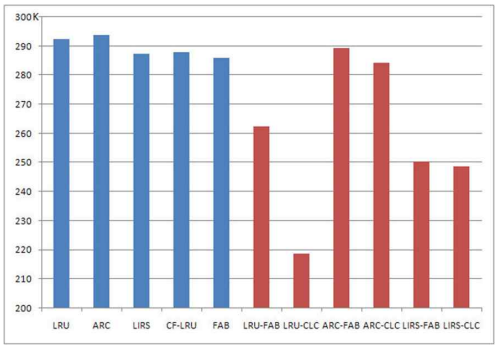
4.4 수행시간

그림 10, 11, 12는 각 알고리즘에 대한 수행시간을 나타낸 것이다. 수행시간은 읽기, 쓰기, 지우기의 발생 횟수를 측정한 후 각각의 비용과 횟수를 곱하여 이를 합산한 것이다. 앞서 언급한 것처럼 수행시간은 읽기/쓰기 횟수만 반영되는 적중률보다 SSD의 평가 척도로 더 적절하다. 그림에서 알 수 있듯이, 각각의 ABM 알고리즘은

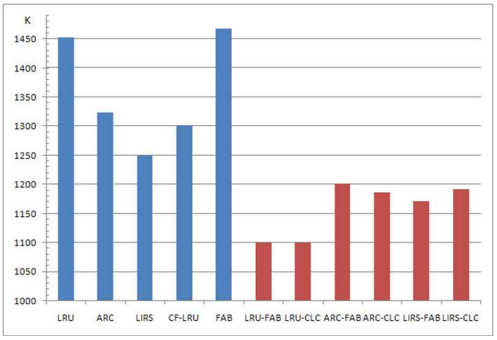
본래 알고리즘들 보다 우수한 성능을 보이는 것으로 나타났다. 그림10, 11, 12에서 보이는 것처럼 대부분의 수행시간은 그림 7, 8, 9에서 보이는 쓰기의 비용과 밀접한 연관이 있는 것을 알 수 있다. 그러나 가장 높은 성능을 보이는 것은 모든 경우에 쓰기 비용이 가장 적은 ABM-LRU-FAB가 아니라 ABM-LRU-CLC인 것으로 나타났는데 이는 CLC 알고리즘이 시간 지역성을 고려하고 이로 인해 적중률이 증가하기 때문이다. ABM-LRU-CLC의 경우 쓰기뿐만 아니라 읽기 또한 효과적으로 감소하고 이래 인해 여러 알고리즘 중에 가장 좋은 성능을 보이는 것으로 나타났다. ABM-LRU-CLC의 경우 기존의 LRU와 비교했을 경우 최대 32%의 성능향상을 보인다. 또한 이 실험으로 인한 결과는 일반적인 경우 가장 적중률이 높은 LIRS나 ARC 알고리즘의 성능이 다른 알고리즘에 비해 좋지 않음을 알 수 있으며, SSD를 위한 버퍼 교체 알고리즘은 디스크 기반 알고리즘과 달리 적중률로는 평가할 수 없음을 시사해 준다.



(그림 10) GCC 트레이스 수행시간



(그림 11) Viewperf 수행시간



(그림 12) Cscope 수행시간

5. 결론

SSD에서 쓰기는 읽기보다 매우 느리고, 지우기는 쓰기보다 매우 느리다. 이러한 SSD의 특성을 개선하기 위해 여러 버퍼 교체 알고리즘들이 제안되었지만, 이들은 적중률 감소와 지우기 및 추가 쓰기를 고려하지 않아 성능이 저하되는 경우가 발생하였다. 본 논문에서 제안하는 ABM 정책은 읽기와 쓰기 버퍼를 분리하고, 읽기 버퍼에는 페이지 단위의 시간 지역성을 고려하는 알고리즘을 적용하고 쓰기 버퍼에는 블록 단위의 공간 지역성을 고려하는 알고리즘을 적용하였다. 제안하는 ABM 정책은 전체적인 IO 횟수를 감소시키고 순차쓰기 횟수를 증가시킴으로 인해 SSD의 지우기와 추가 쓰기를 효과적으로 감소시키고, 이에 따라 시스템의 성능이 향상되는 것으로 나타났다.

시뮬레이션 기반 성능평가를 통해 ABM 정책을 적용한 여러 알고리즘들이 기존의 알고리즘보다 뛰어난 성능을 보이는 것을 확인하였다. 특히 ABM-LRU-CLC의 경우 최대 32 %의 성능향상을 보이는 것으로 나타났다.

참 고 문 헌

[1] Nimrod Megiddo, Dharmendra Modha, “ARC: A Self-Tuning, Low Overhead Replacement Cache”, Proc. 2nd USENIX Conference on File and Storage Technologies (FAST 03), 2003

[2] Song Jiang, Xiaodong Zhang, “LIRS: an efficient low inter-reference recency set replacement policy to improve buffer cache performance” , ACM SIGMETRI

CS Performance Evaluation Review archive Vol. 30, No 1, pp. 31-42, 2002

[3] Chanik Park, Jeong-Uk Kang, Seon-Yeong Park, Jin-Soo Kim, "Energy-aware demand paging on NAND flash-based embedded storages", Proc. of the 2004 Intl. Symposium on Low Power Electronics and Design, pp 338 - 343, 2004.

[4] Hoyoung Jung, Hyoki Shim, Sungmin Park, Sooyong Kang, Jaehyuk Cha, "LRU-WSR: Integration of LIRS and Writes Sequence Reordering for Flash Memory", IEEE Trans. Consumer Electron., Vol. 54, No. 3, pp. 1215 - 1223, 2008.

[5] Heesung Jo, Jin-Soo Kim et al., "FAB: Flash-Aware Buffer Management Policy for Portable Media Players", IEEE Trans. Consumer Electron., Vol. 52, No. 2, pp. 485-493, May. 2006.

[6] Sungmin Park,Hoyoung Jung, Hyoki Shim,Sooyong Kang, Jaehyuk Cha, "Using Non-Volatile RAM as a Write Buffer for NAND Flash Memory-based Storage Devices", 2008 IEEE International Symposium on Modeling, Analysis & Simulation of Computer & Telecommunication Systems, MASCOTS, Sep., 2008

[7] Intel, "Understanding the Flash Translation Layer (FTL)Specification", White Paper, http://www.embeddedfreebsd.org/Documents/Intel-FTL.pdf, 1998

[8] Intel® X25-E SATA Solid State Drive SSDSA2SH032G1 Advance Product Manual, http://download.intel.com/design/flash/nand/extreme/extreme-sata-ssd-datasheet.pdf

[9] Ali R. Butt, Chris Gniady, Y. Charlie Hu, "The Performance Impact of Kernel Prefetching on Buffer Cache Replacement Algorithms", Proc. of the 2005 ACM SIGMETRICS intl. conference on Measurement and modeling of computer systems, pp. 157 - 168, 2005

[10] A. Sliberschantz et al, "Operating System Concepts", 6th ed., John Wiley & Sons, Inc. 2004

[11] Jim Gray and Fransco Putzolu, " The Five Minutes Rule for Trading Memory for Disk Accesses and The 10 Byte Rule for Trading Memory for CPU Time," Proc. ACM SIGMOD, pp. 395-398, 1987

[12] Dongyoung Seo, Dongkun Shin, "Recently-Evicted-First Buffer Replacement Policy for Flash Storage Devices," IEEE Trans. Consumer Electron., Vol. 54, No. 3, pp. 1228-1235, Aug. 2008

[13] Hyojun Kim and Seongjun Ahn, "BPLRU: A Buffer Management Scheme for Improving Random Writes in Flash Storage", Proc. FAST '08 pp. 239-252, 2008

[14] Jesung Kim, Jong Min Kim, Sam H. Noh, Sang Lyul Min and Yookun Cho, "A Space-Efficient Flash Translation Layer for Compactflash systems", IEEE Trans. Consumer Electron., Vol. 48, No. 2, pp. 366-375, May, 2002.

[15] Song Jiang, Xiaoning Ding, Feng Chen, Enhua Tan, Xiaodong Zhang, "DULO: an Effective Buffer Cache Management Scheme to Exploit both Temporal and Spatial Locality", Proc. FAST'05, Dec., 2005.

[16] Philip E. Brunelle et al, "Demand allocation of read/write buffer partitions favoring sequential read cache", U.S. Patent 5,381,528, Oct., 1992

[17] Binny S. Gill and Dharmendra S. Modha, "WOW: Wise Ordering for Writes - Combining Spatial and Temporal Locality in Non-Volatile Caches ," Proc. FAST'05 pp. 129-142, 2005

[18] Ali R. Butt, Chris Gniady, Y. Charlie Hu, "The Performance Impact of Kernel Prefetching on Buffer Cache Replacement Algorithms", Proc. of 2005 ACM SIGMETRICS intl. conference on Measurement and modeling of computer systems, pp. 157-168, 2005

[19] Suman Nath, Aman Kansal, "FlashDB: Dynamic Self-tuning Database for NAND Flash", Microsoft white paper, MSR-TR-2006-168, 2006

[20] A. Kawaguchi, S. Nishioka, "H. Motoda, A Flash Memory Based File System", Proc. of the USENIX Technical Conference, 1995

[21] M. L. Chiang, C. H. Paul, R. C. Chang, "Manage flash memory in personal communicate devices", Proc. of IEEE Intl. Symposium on Consumer Electronics, 1997

[22] Eran Gal, Sivan Toledo, "Mapping Structures for Flash Memories: Techniques and Open Problems", Proc. of the IEEE Intl. Conference on Software-Science, Technology and Engineering (2005)

[23] Donghee Lee, Jongmoo Choi et al., "LRFU: A Spectrum of Policies that Subsumes the Least Recently Used and Least Frequently Used Policies", IEEE Transactions on Computers, Vol. 50, No. 12, Dec, 2001

[24] Theodore Johnson, Dennis Shasha, "2Q: A Low Overhead High Performance Buffer Management Replacement Algorithm", Proc. of the 20th Intl. Conference on Very Large Databases

[25] S. Jiang, F. Chen and X. Zhang, "CLOCK-Pro: An Effective Improvement of the CLOCK Replacement", Proc. Of USENIX '05, Apr., 2005.

[26] E. J. O'Neil, P. E. O'Neil, and G.Weikum, "The LRU-K Page Replacement Algorithm for Database Disk Buffering" , Proc. of SIGMOD '93, 1993.

정 호 영

2004년 : 한양대학교 공과대학 재료공학부 (공학사)
2006년 : 한양대학교 대학원 정보통신공학과 (공학석사)
2008년 : 한양대학교 대학원 전자컴퓨터통신공학과 (박사 수료)
2011년~현 재 : LG전자 CTO SW Platform (연) 주임연구원
관심분야 : 운영체제, 데이터 베이스, 파일 시스템, 플래시 메모리, 병렬 프로그래밍

강 수 용

1996년 : 서울대학교 수학과 (이학사)
1998년 : 서울대학교 컴퓨터공학과 (공학석사)
2002년 :서울대학교 컴퓨터공학과 (공학박사)
2003년~현 재 : 한양대학교 정보통신대학 교수
관심분야 : 멀티미디어, 인터넷 정보보안, 멀티미디어 데이터 전송, 멀티미디어 데이터저장

차 재 혁

1987년 : 서울대학교 계산통계학과 (이학사)
1991년 : 서울대학교 컴퓨터공학과 (공학석사)
1997년 :서울대학교 컴퓨터공학과 (공학박사)
1998년~2001년 : 한양대학교 컴퓨터교육과 조교수
2002년~현 재 : 한양대학교 컴퓨터공학부 교수
관심분야 : 데이터베이스, 파일시스템, 플래시 메모리, 이러닝, 콘텐츠변환 등