ISSN 1598-0170 (Print)
ISSN 2287-1136 (Online)
http://www.jksii.or.kr

Hybrid SSD 시스템을 위한 재사용 간격 기반 블록 교체 기법☆

Block Replacement Scheme based on Reuse Interval for Hybrid SSD System

유 상 현[1] 김 경 태[1] 윤 희 용[1*]
Sanghyun Yoo Kyung Tae Kim Hee Yong Youn

요 약

최근 SSD(Solid State Drive)는 빠른 읽기/쓰기, 저전력 등 다양한 장점을 가지고 있어 스마트폰, 노트북, 서버 등의 저장장치로 사용 영역이 확대되고 있다. 하지만, 플래시 메모리의 읽기 및 쓰기의 비대칭적 성능과 제한된 쓰기 횟수가 SSD의 수명을 단축시키는 문제가 있어서 캐쉬(cache)로 사용되는 SSD의 내용을 변경시키는 블록 교체 기법(block replacement policy)이 매우 중요하다. Hybrid SSD의 수명을 향상 시킬 수 있는 방법 중 하나로 LARC 기법이 있으나, LARC는 SSD블록 관리를 위해 기존 LRU알고리즘을 사용하기 때문에 빈번히 참조되는 블록이 오래된 블록 대신 교체되어 SSD 미스율을 증가시킴으로써 시스템의 성능이 저하되는 문제점이 발생한다. 따라서, 본 논문에서는 다양한 데이터 읽기, 쓰기 환경에 효과적으로 대응하기 위해 블록의 재사용 간격을 고려한 새로운 블록 교체 기법을 제안한다. 제안된 기법은 블록 재사용 간격(Reuse interval)과 Age를 기반으로 최근성(Recency)을 추출하고 참조빈도(Frequency)를 같이 고려하여 블록을 교체한다. Workload 기반 Trace를 이용한 실험결과, 제안하는 기법은 여러가지의 기존 블록 교체 기법 및 LARC 알고리즘과 비교하여 쓰기 횟수 감소와 히트율 향상을 통해 시스템 성능과 SSD의 수명을 연장시킨다.

☞ 주제어 : Solid State Drive, 블록 교체, 재사용 간격, 블록 우선순위, 고스트 큐

ABSTRACT

Due to the advantages of fast read/write operation and low power consumption, SSD(Solid State Drive) is now widely adopted as storage device of smart phone, laptop computer, server, etc. However, the shortcomings of SSD such as limited number of write operations and asymmetric read/write operation lead to the problem of shortened life span of SSD. Therefore, the block replacement policy of SSD used as cache for HDD is very important. The existing solutions for improving the lifespan of SSD including the LARC scheme typically employ the LRU algorithm to manage the SSD blocks, which may increase the miss rate in SSD due to the replacement of frequently used block instead of rarely used block. In this paper we propose a novel block replacement scheme which considers the block reuse interval to effectively handle various data read/write patterns. The proposed scheme replaces the block in SSD based on the recency decided by reuse interval and age along with hit ratio. Computer simulation using workload trace files reveals that the proposed scheme consistently improves the performance and lifespan of SSD by increasing the hit ratio and decreasing the number of write operations compared to the existing schemes including LARC.

☞ keyword : Solid State Drive, Block Replacement, Reuse distance, Block priority, Ghost queue

1. 서 론

지난 수십 년간 하드디스크를 중심으로 한 스토리지 시스템은 이동장치를 포함한 모든 컴퓨터 시스템의 성능 향상을 위한 중요한 대상이었다. 최근 NAND 플래시 메모리 기반의 SSD(Solid State Drive)[1-3]의 등장으로 인해 하드디스크 중심의 저장장치 패러다임이 바뀌고 있다. SSD는 기존의 하드디스크를 대체할 수 있는 저장장치로 HDD의 기계적 요소를 모두 제거하여 반응 속도를 비약적으로 향상시켰다. 또한, SSD는 기존의 하드디스크와 비교하여, 무작위 요청 시 빠른 데이터 제공이 가능하며, 저전력, 저소음, 진동과 고온에서의 물리적 내성이 있어 데이터 안정성이 뛰어나며, NAND 플래시 메모리 소자 기반이기 때문에, 소형화가 가능하다는 이점이 있다. 그

[1] College of Information Communication and Engineering, Sungkyunkwan University, Suwon, Korea.
* Corresponding author (youn7147@skku.edu)
[Received 18 August 2015, Reviewed 22 August 2015, Accepted 18 September 2015]
☆ 본 연구는 BK21+사업, 한국연구재단 기초연구사업 (2012R1A1A2040257), (2013R1A1A2060398), 삼성전자, 미래창조과학부 및 정보통신기술연구진흥센터의 정보통신,방송 연구개발사업 (1391105003)의 일환으로 수행되었음

러므로 소형화가 필요한 다양한 임베디드 장치와 이동장치에 빠르게 적용되고 있다. 하지만, 아직까지 SSD의 단위 공간 당 가격이 HDD에 비해 높은 실정이다. 따라서 최근 NAND 플래시메모리 기반의 SSD와 HDD를 동시에 사용하는 시도가 늘고 있다 [14]. 이와 같은 환경에서 저장시스템의 목표는 이질적인 저장매체의 장점을 극대화하는 것이다. 즉, 스토리지 용량은 HDD처럼 사용하면서 전력소모나 접근 속도 측면에서는 플래시메모리처럼 사용하는 것이다.

SSD와 HDD를 동시에 사용하는 Hybrid 저장시스템에서 HDD에 있는 블록이 한 번 요청되면, 앞으로 자주 사용 될 블록으로 판단하고 SSD로 할당한다. SSD를 사용하면 시스템이 HDD를 참조하기 위해 소요되는 참조 지연시간(Access latency)을 최소화할 수 있다. 하지만, 필요한 블록이 SSD에 없는 경우, 이를 캐시 미스(miss)가 발생했다고 하는데, 캐시 미스로 인한 시스템의 성능저하는 수백 사이클(cycle)에 이르기 때문에 빈번히 사용되는 블록을 최대한 SSD에 유지하는 것이 SSD의 효율을 극대화하기 위해 매우 중요하다.

LRU(최소 최근 사용)[4] 교체 기법은, 캐시 교체 알고리즘으로써 가장 많이 사용되고 있다. 하지만 LRU는 데이터의 최근성만 반영하기 때문에 데이터가 시간적으로 빈번히 사용되지 않는 시스템 환경에서는 적합하지 않은 문제점이 있다. 따라서 LRU외에 LFU[5,6], 2Q[7], LIRS[8], ARC[9,15]등의 다양한 교체 알고리즘이 연구되었다[10,11]. 위의 교체 알고리즘들을 Hybrid 저장시스템에 적용한다면, 향상된 SSD의 히트율을 제공함으로써 블록 참조 지연시간을 최소화함으로써 시스템의 성능을 향상시킬 수 있지만, 쓰기 연산이 많이 발생해 SSD의 수명이 단축되는 문제점이 있다. 무작위 파일 쓰기 연산에 집중된 Workload은 기존대비 100배 이상 SSD 수명을 단축시킨다. 따라서 SSD의 수명을 유지시키면서 시스템의 성능을 향상시키는 LARC[1](Lazy Adaptive Cache Replacement)알고리즘이 제안되었다. LARC는 SSD를 HDD의 캐시로 사용하여 SSD의 수명과 시스템의 성능을 향상시킨다.

LARC는 엑세스된 블록이 SSD에 할당되기 전에 블록의 식별자만 따로 저장하는 LRU 알고리즘을 이용하는 Ghost Queue를 제안하여 한 번 엑세스된 블록을 별도로 관리하도록 하였다. Ghost Queue에 저장된 블록이 한 번 더 호출되면, 이때 SSD로 해당 블록을 할당한다. 이를 통해 불필요한 SSD쓰기 연산을 비약적으로 줄이고, 일정 시간 내에 두 번 엑세스된 블록을 SSD에 할당시킴으로써, 기존 알고리즘보다 수명과 성능 모두 향상시킨다. 하지만, LARC알고리즘은 여전히 SSD에 LRU정책을 이용하기 때문에, 다양한 데이터 읽기, 쓰기 환경에 효과적으로 대응하지 못하는 문제점이 있어 성능 향상에 한계가 있다.

본 논문에서는 블록의 재사용 간격을 고려한 캐시 교체 정책인 BRI(Block Replacement Algorithm with Interval Reuse)알고리즘을 제안한다. 제안하는 알고리즘은 LARC와 동일하게 SSD를 캐시로 사용하는 Hybrid 저장시스템 환경에서 블록 재사용 간격(Reuse interval)에 기반을 둔 최근성(Recency)과 히트율(Hit ratio)을 기반으로 한 참조빈도(Frequency)를 고려해 블록을 교체한다. 이 알고리즘을 통해 자주 쓰이는 블록과 그렇지 않은 블록을 더욱 정확하게 나눌 수 있으며, 자주 쓰이지 않는 블록은 HDD로 보내고, 자주 쓰이는 블록은 SSD에 할당한다. 본 논문에서 제안한 기법을 검증하기 위해 두 가지의 실제 Workload 기반의 Trace[13]를 이용하여 실험하였으며, 기존에 연구된 다섯 가지 알고리즘인 LRU, LFU, 2Q, ARC, LARC과 성능 비교하였다. 실험 결과, 제안하는 기법은 기존의 다섯 가지의 교체 알고리즘보다 향상된 히트율(Hit ratio)과 낮은 쓰기율(Write ratio)을 보여줌으로써, 기존 알고리즘보다 Hybrid 저장시스템의 입출력 성능과 SSD의 수명을 향상시켰다.

본 논문은 다음과 같이 구성된다. 2장에서는 기존의 블록교체알고리즘에 대해 알아본다. 3장에서는 제안하는 BRI 알고리즘에 대해 상세히 기술하고, 4장에서는 실험을 통해 제안한 기법의 성능 및 실험 결과를 보인다. 마지막으로, 5장에서는 결론을 맺는다.

2. 관련연구

2.1 기존의 블록 교체 알고리즘

LRU 알고리즘[4]은 가장 많이 사용되는 블록 교체 알고리즘으로서 HDD의 캐시로 SSD를 사용하는 환경에서 널리 사용되고 있다. LRU 알고리즘은 블록의 최근성만 고려하는 알고리즘이기 때문에, 최근성이 낮은 Workload에서는 비교적 낮은 성능을 보여준다.

이에 반해, LFU[5]알고리즘은 과거의 사용 정보를 참고하여 교체할 블록을 결정하는 알고리즘이다. LRU-K알고리즘은 과거사용 정보를 반영하여 캐시를 교체하지만, 구현하고자 하는 캐시의 크기 당 비용이 많이 드는 문제점이 있다.

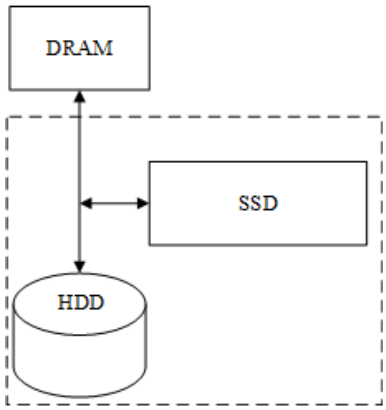
LRFU알고리즘[6]은 LRU와 LFU알고리즘을 결합한 형태로 사용된 페이지의 과거 기록을 참조하여 가중치를 결정한 뒤 교체를 결정하는 알고리즘이다. 2Q알고리즘[7]은 최근 사용 기록을 기반으로 방출할 블록을 결정하며, 방출된 페이지는 별도의 스택에서 방출된 순서대로 저장하는 알고리즘이다. 2Q알고리즘은, FIFO스택, LRU 스택과, FIFO history 스택으로 총 3종류의 고정된 스택으로 구성된다.

ARC(Adaptive Replacement Cache)알고리즘[9]은 최근에 사용된 블록을 관리하는 스택과 자주 사용된 블록을 저장하는 스택으로 나누고 스택의 사이즈를 동적으로 조정한다. ARC 알고리즘은 또한 최근에 사용된 블록과 자주 사용된 블록의 기록도 스택으로 별도 관리한다.

기존의 LRU, LFU, LRFU, 2Q, ARC 등의 블록교체 기법들은 SSD를 HDD의 캐시로 사용하는 Hybrid 저장 시스템에서 적용되어 널리 사용되고 있다. 그러나 이러한 기법들은 주로 RAM과 CPU 캐시(Cache)교체를 위해 고안된 기법이다. 따라서 제한된 쓰기 연산 횟수를 갖고 있는 SSD의 물리적 특징을 충분히 고려하지 않은 위 알고리즘을 Hybrid 저장시스템에 적용한다면, 쓰기 연산이 많이 발생해 SSD 수명이 감소하는 문제점이 있다. 이러한 문제점을 해결하기 위하여 SSD의 쓰기 연산을 최소화하면서, 시스템 성능을 향상시키기 위해 LARC 알고리즘이 제안되었으며, 2.2장에서 LARC알고리즘에 대해 상세히 기술한다.

2.2 LARC

SSD의 쓰기 연산을 최소화하기 위해, Lazy Adaptive Replacement Cache(LARC) 알고리즘은 가장 먼저 SSD를 효율적으로 사용하기 위해 크게 두 가지 방향으로 연구하였다[12]. 첫째는 RAM과 같이 SSD를 시스템 메모리로 사용하는 방향이고, 또 다른 방법은 SSD를 HDD의 캐시로써 쓰는 방법이다. 물론, 첫 번째의 방법이 RAM의 특성상 HDD의 확장된 저장장치로 사용하는 방법보다, 더 빠른 입출력 성능을 제공할 수 있다. 하지만, DRAM은 휘발성 저장장치이며, 이를 구현한다는 것은 일반적으로 운영체제 혹은 응용프로그램 그 자체의 변형을 의미한다. 이 방법은 구현하기에 비용도 많이 들 뿐만 아니라, 데이터의 영속성도 보장하지 못한다. 이런 이유로, LARC 알고리즘은 SSD를 HDD의 캐시로 사용하는 환경에서 연구를 진행하였다. 그림 1은 Hybrid-SSD 환경을 보여준다.



(그림 1) Hybrid-SSD 환경
(Figure 1) The environment of Hybrid-SSD

LARC알고리즘은 자주 요청되지 않는 블록을 Ghost cache를 이용하여 Cache에서 방출한다. Ghost cache는 데이터 블록의 식별자만 저장하는 LRU Queue 이다. HDD에 저장된 블록이 한번 요청되면 SSD로 이동하기 전에 향후 SSD에 할당될 후보블록으로서 Ghost cache에 저장된다. Ghost cache에 있는 블록이 한 번 더 요청될 경우, 자주 쓰일 블록으로 간주하고 SSD로 이동시킨다. SSD 캐시는 LRU 정책을 이용하여 관리한다.

LARC에서 블록이 요청되면 가장 먼저 SSD 블록 Queue에 요청된 블록이 있는지 검색한다. 만약 요청된 블록이 있으면, SSD Queue 의 MRU위치로 요청된 블록을 배치한다. 반면에, SSD에 요청된 블록이 없다면, Ghost Queue에 요청된 블록이 있는지 찾는다. 만약 요청된 블록이 없으면, HDD에서 요청된 블록정보를 가져온다. 여기에서 정적인 크기의 Ghost Queue를 이용할 경우, HDD의 크기만큼 큰 Ghost Queue는 거의 모든 HDD의 블록을 저장하기 때문에, SSD 수명향상이 미비해지며, 너무 작은 Ghost Queues는 LRU정책에 의해 한 번 엑세스된 블록이 체류할 수 있는 시간이 너무 짧아져, SSD의 성능을 극단적으로 감소시키는 문제점이 있다. 따라서 LARC는 이를 해결하기 위해, Ghost Queue의 크기를 동적으로 조절하는 알고리즘을 제안하였다. 요청된 블록이 SSD Queue에 없을 경우에 LARC알고리즘은 HDD에 있는 요청된 블록을 Ghost Queue에 넣는다. 이 동작 후 아래와 같이 크기가 조정된다.

$$G_q = Min(0.9 * S_q, G_q + \frac{S_q}{G_q}) \quad (1)$$

반면에 요청된 블록이 SSD Queue에 있을 경우 Ghost Queue 크기가 아래와 같이 조정된다.

$$G_q = Max(0.1 * S_q, G_q - \frac{S_q}{S_q - G_q}) \quad (2)$$

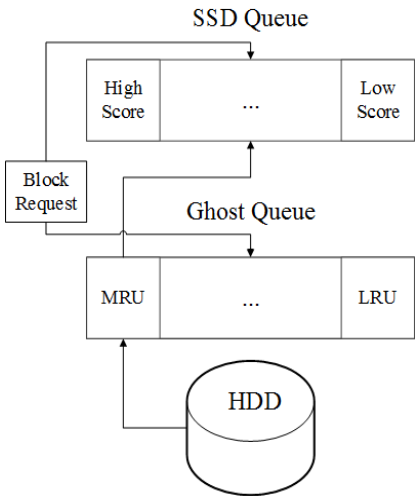
위 알고리즘을 통해 블록의 히트여부에 따라 동적으로 Ghost Queue의 크기를 조정함으로써 효과적으로 SSD 쓰기 횟수를 줄일 수 있어 기존 블록 알고리즘보다 월등히 SSD수명을 향상시켰다. 하지만, LARC는 SSD 블록 교체 정책으로 LRU정책을 사용하고 있어, 다양한 읽기/쓰기 컴퓨팅 환경에 효과적으로 대응하지 못해 SSD의 성능을 향상하는데 한계가 있었다. 따라서 본 논문에서는 SSD의 LRU정책 대신에 블록의 가장 최근 정보를 고려하는 에이지(Age)와 과거 액세스 정보를 고려하는 재사용 간격(Reuse interval)을 기반으로 최근성을 정의하고, 히트 횟수를 통해 블록이 얼마나 빈번히 사용되는지 고려하였다. 교체할 블록을 결정할 때 각각의 히트 횟수에 계산된 최근성 값을 나누어 블록 중요도를 결정한다. 가장 낮은 블록 중요도를 갖는 블록이 새로 HDD에서 할당되는 블록을 위해 교체된다. 블록의 재사용 간격과 최근 사용되었던 시간, 그리고 히트 횟수 모두를 고려함으로써 SSD를 HDD의 캐시로 사용하는 Hybrid SSD 저장시스템의 성능과 SSD 수명 모두 향상시키는 기법을 제안한다.

3. 재사용 간격 기반 블록 교체 기법

본 장에서는 제안한 기법에 대해 상세히 기술 한다. SSD를 HDD의 캐시로 사용하는 Hybrid 저장시스템에서 제안하는 기법은 SSD 히트율을 향상시킴으로써 시스템 성능을 제고하고, 쓰기 횟수를 최소화함으로써 SSD 수명을 연장시킨다. 제안한 기법은 효과적으로 SSD 쓰기 횟수를 최소화하기 위해서, 기존에 연구된 LARC의 Ghost Queue 정책을 사용하였다. HDD에 있는 블록이 최초로 한 번 요청될 경우, 요청된 블록의 메타데이터(Metadata)는 Ghost Queue에 저장하고, LRU알고리즘으로 관리한다. 한 번 요청된 블록을 저장하는 Ghost Queue에서 요청이 한 번 더 발생한 경우, 앞으로 자주 쓰일 블록으로 간주하고, SSD로 요청된 블록을 옮긴다. SSD Queue는 블록의 메타데이터를 저장하고 있으며, 이 메타데이터는 본 논문에서 제안한 BRI 알고리즘에 의해 관리된다. 이를 통해, 자주 쓰이는 블록을 SSD에 비교적으로 오랫동안 저장할 수 있다.

3.1 시스템 모델

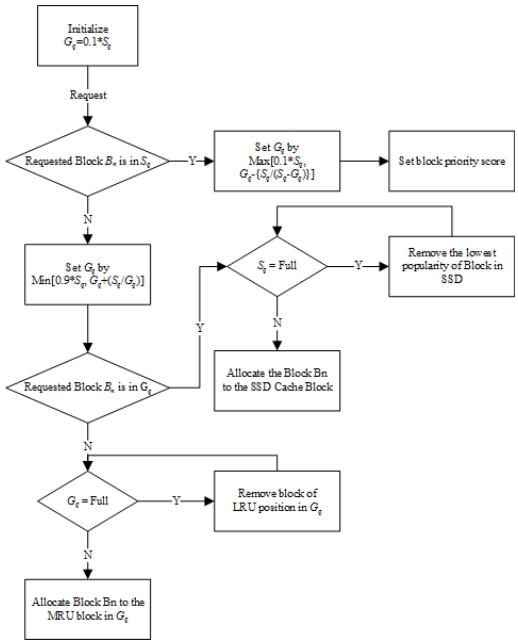
그림 2는 BRI알고리즘의 시스템 모델로, 크게 HDD, Ghost Queue, SSD Queue로 구성된다. 블록이 요청될 경우, 가장 먼저 SSD Queue에 요청된 블록이 있는지 찾는다. SSD Queue에 요청된 블록이 없을 경우, 그 다음 Ghost Queue에 요청된 블록이 있는지 찾는다. 요청된 블록이 Ghost Queue에 있다면, 요청된 블록이 저장된 HDD에서 SSD로 옮긴다. 반면에 요청된 블록이 SSD Queue와 Ghost Queue에 없을 경우, 요청된 블록을 Ghost Queue의 MRU위치에 저장한다.



(그림 2) BRI알고리즘의 시스템 모델
(Figure 2) The system model of BRI algorithm

BRI알고리즘의 주된 목적은 사용 기록을 바탕으로 자주 요청되는 블록을 구별하는 것이다. 동적으로 크기가 조정되는 Ghost Queue를 SSD에 할당하기 위한 필터로 사용함으로써, BRI 알고리즘은 효과적으로 불필요한 쓰기 횟수를 줄이고 히트율을 높일 수 있다.

그림 3은 제안한 BRI알고리즘의 흐름도이다. SSD의 불필요한 쓰기 연산을 줄이기 위해서, Ghost Queue의 사이즈(G_q)를 동적으로 조절한다. Ghost Queue는 LRU 정책에 의해 관리되기 때문에, Ghost Queue의 사이즈가 작을수록 방출되기까지 짧은 대기시간을 의미한다. Ghost Queue의 크기는 블록 사용패턴에 따라 SSD의 쓰기 횟수를 효과적으로 줄일 수 있는 크기로 재조정한다. SSD에 블록이 저장된 후에는 BRI 교체 정책에 의해 관리된다.



(그림 3) BRI 알고리즘의 흐름도
(Figure 3) The flowchart of BRI algorithm

3.2 블록 중요도 정의

(표 1) 블록 중요도 모델 변수 정의
(Table 1) The variables used to model block priority

변수	정의
T	현재 시간(cycle)
$T_r(n)$	블록n의 최초 액세스 시간
$A(n)$	블록n의 에이지(Age)
$I_i(n)$	블록n의 i번째와 i+1번째의 재사용간격
$AI(n)$	재사용간격 평균
$R(n)$	블록n의 최근성
$C(n)$	블록n의 액세스 횟수
$P(n)$	블록n의 중요도

SSD에 할당된 블록을 효과적으로 관리하기 위해 BRI 알고리즘은 방출할 블록을 정의하는 중요도(Priority)를 정의하고 이를 기반으로 블록 우선순위를 정의한다. 블록 중요도는 블록의 최근성(Recency)과 참조빈도(Frequency)로 정의한다. 블록의 최근성은 재사용 간격(Reuse Interval)과 에이지(Age)로 정의하고, 블록의 참조빈도는 블록의 요청횟수로 정의한다.

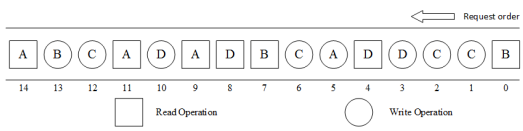
3.2.1 최근성(Recency)

블록의 최근성(Recency)은 재사용 간격(Reuse Interval)과 에이지(Age)로 정의한다. 블록 n의 에이지는 최근 사용 이후로 SSD에 얼마나 오랫동안 체류하고 있는지를 의미한다. 따라서 블록의 에이지는 다음과 같이 계산된다.

$$A(n) = T - T_r(n) \tag{3}$$

블록의 에이지(Age)는 마지막으로 사용된 이후에서의 체류시간을 고려한다. 따라서 과거의 사용 정보를 나타내지 못한다.

그림 4는 블록 A, B, C, D의 읽기/쓰기 요청의 예시이다. 여기에서 사각형은 읽기 요청이고, 원형은 쓰기 요청을 뜻한다. 블록 A의 에이지는 0으로, $A(A)$=0이다. 위에 서술한 식에 의하면 현재 시간 T는 14이고, 최근에 사용된 시간은 14로 그 차가 0이 되기 때문이다. 마찬가지로 $A(B)$=1, $A(C)$=2, $A(D)$=4이다. 낮은 에이지 값을 가질수록 가장 최근에 사용된 블록을 뜻한다. 하지만, 에이지를 활용한 기법은 가장 최근에 사용된 블록의 시간만을 고려하기 때문에, 블록의 최근성을 완전히 반영하지 못하는 문제점이 있다. 이러한 문제점을 해결하기 위해, BRI알고리즘은 블록의 과거사용 기록에 기반을 두어 재사용 간격을 같이 고려한다.



(그림 4) 블록 읽기/쓰기 요청의 예
(Figure 4) An example of read/write requests

3.2.2 재사용 간격(Reuse interval)

제안하는 기법에서 블록 재사용 간격을 정의하기 위해, 블록 사용 순서(σ_a^b)는 다음과 같다.

$$\sigma_a^b = \begin{cases} \sigma[a], \sigma[a+1], \ldots, \sigma[b] & a \le b \\ 0, & a > b \end{cases} \tag{4}$$

σ_a^b는 시간 a부터 b까지 사용된 모든 블록 액세스 순서를 뜻한다. 따라서 그림 4의 경우 σ_1^5는 '*CCDDA*'가 된다. 이를 기반으로 고유 블록의 수, $|\sigma|$는 다음과 같이 정의된다.

$$|\sigma| = |\{i : i \in \sigma\}| \quad (5)$$

앞서 기술한 정의를 바탕으로, $|\sigma_1^5|$ 은 3이 된다. 따라서 시간 a부터 b까지의 블록 n의 사용에 따른 시간 ($S_n(\sigma_a^b)$)은 다음과 같다.

$$S_n(\sigma_a^b) = \{i : (a \le i \le b) \wedge (\sigma[i] = n)\} \quad (6)$$

정의된 사용 시간에 의해, $S_d(\sigma_1^5)$ 는 {3, 4}가 된다. 이를 바탕으로, 사용 블록 순서의 최댓값을 구하는 과정은 다음과 같다.

$$Max_n(\sigma_a^b) = \begin{cases} \max S_n(\sigma_a^b), & S_n(\sigma_a^b) \neq 0 \\ 0, & otherwise \end{cases} \quad (7)$$

수식에 따르면, $Max_d(\sigma_1^5)$ 는 max{3, 4}=4가 된다. 서술한 식을 기반으로 블록 n의 재사용 간격($D(n)$)은 다음과 같이 계산한다.

$$I(n) = \begin{cases} \left|\sigma^{t-1}_{Max_n(\sigma_0^{t-1})+1}\right|, & Max_n(\sigma_0^{t-1}) \neq 0 \\ \infty, & otherwise \end{cases} \quad (8)$$

t는 가장 최근에 블록n이 사용된 시간을 의미한다. 블록n이 사용될 때마다 재사용 간격을 계산한다. 과거의 재사용 간격을 반영하기 위해 블록n이 사용될 때마다 평균($I(n)$)을 계산한다.

$$AI(n) = \frac{\sum_{i=1}^{C(n)-1} I_i(n)}{C(n)-1} \quad (9)$$

그림 4의 블록 읽기/쓰기 요청의 예시를 기반으로 계산하면, $I_1(A)$=3, $I_2(A)$=1, $I_3(A)$=2가 되고 이의 평균을 계산하면, $AI(A)$=2가 된다. 이와 같은 방법으로 나머지 블록 B, C, D에 대해 구하면, $AI(B)$=3, $AI(C)$=2, $AI(D)$=1.66이 된다. I의 값이 낮은 블록일수록 앞으로 더 자주 사용될 블록임을 의미한다. 마지막으로 블록의 최근성($R(n)$)을 블록의 에이지(Age)와 재사용 간격(Reuse interval)에 각각의 가중치를 이용한 뒤 합하여 정의하였다.

$$R(n) = \omega_A A(n) + \omega_L I(n), \qquad (\omega_A + \omega_L = 1) \quad (10)$$

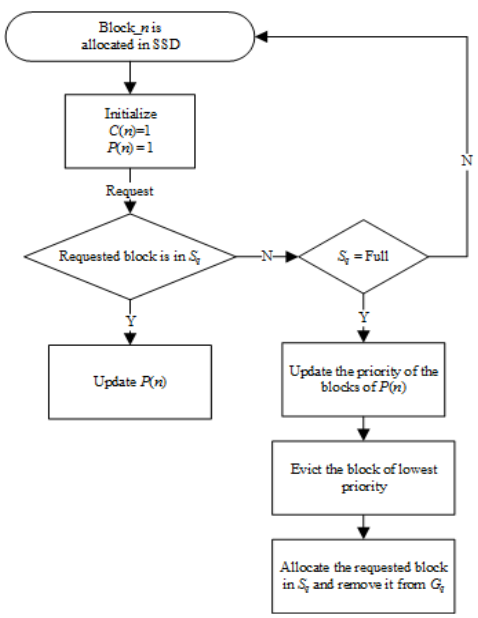
A값및 I값과 마찬가지로 낮은 R 값을 가지는 블록일수록 자주 사용되고 있으며 앞으로도 자주 사용될 것으로 예상되므로 SSD에 오랫동안 유지하게 된다.

3.2.3 블록 중요도(Block Priority)

블록 n의 중요도($P(n)$)는 다음과 같이 정의된다.

$$P(n) = \frac{C(n)}{R(n)} \quad (11)$$

$P(n)$값은 블록 n이 히트될 때마다 증가하며, 더 높은 최근성을 가져 상대적으로 $R(n)$의 값은 낮아진다. 어떤 블록이 처음 SSD에 할당되면, $P(n)$의 값은 1로 초기화된다. 이 블록이 다시 사용될 경우, 위의 수식에 따라 $P(n)$을 계산한다. 기존에 연구된 블록 교체 기법과 비교하여, 본 논문이 제안하는 블록 교체 기법은 SSD에 저장된 블록을 에이지(Age), 재사용 간격(Reuse interval)과 참조빈도(Frequency) 정보에 기반을 두어 더욱 효율적으로 관리할 수 있다.



(그림 5) BRI알고리즘의 블록교체 흐름도
(Figure 5) The flow chart for the replacement of BRI algorithm

그림 5는 SSD에 할당된 블록을 관리하는 알고리즘의 흐름도이다. 특정 블록에 대한 읽기 혹은 쓰기요청이 들어오게 되면, 가장 먼저 SSD Queue에 요청된 블록이 있는지 검색한다. 요청된 블록이 SSD Queue에 있을 경우, SSD에 저장된 블록의 $P(n)$은 제안한 알고리즘에 따라 갱신한다. 반면, 요청된 블록이 SSD Queue에 없을 경우, 그

다음 동작은 그림 3에 서술한 순서도에 의해 동작한다. 그림 5는 SSD의 내부 관리 알고리즘에 대해서만 국한한다. 요청된 블록이 SSD Queue에 없고 Ghost Queue에 있다면, 요청된 블록을 SSD Queue로 이동한다. 이때, Queue가 꽉 차 있다면, SSD에 저장된 블록 중 $P(n)$이 가장 낮은 블록을 HDD로 방출되고, 요청된 블록을 대신 SSD에 할당한다.

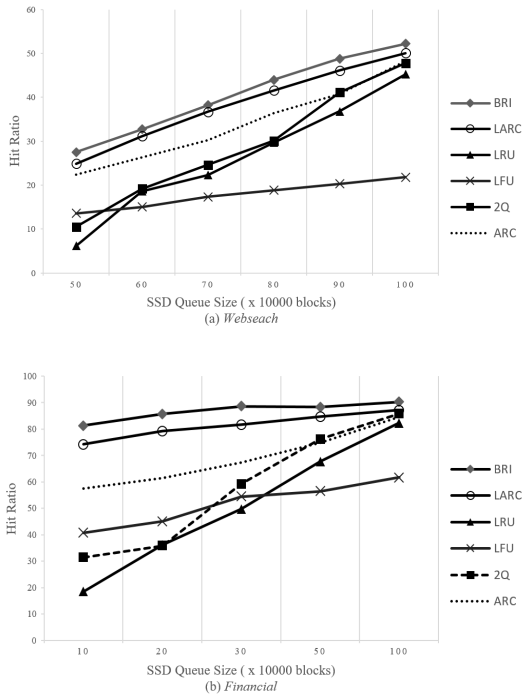
4. 성능평가

본 장에서는 제안한 블록 교체 기법과 기존 교체 알고리즘의 성능을 비교 분석한 결과를 기술한다. 본 성능비교를 위해 C++언어로 구현한 시뮬레이터를 이용하였으며, 제안한 기법은 기존 연구에서 널리 사용된 입출력(I/O)기반의 trace를 이용하여 실험하였다. 또한, 본 실험에서 사용된 trace는 UMass Trace Repository[13]에서 제공하며, 표 2에 사용된 trace에 관한 정보를 요약하였다. 사용된 trace는 실제 입력된 데이터에 기반을 두었고 읽기/쓰기 연산 비율이 비대칭으로 구성되어있다. 본 실험을 위해 'Websearch' trace 블록의 크기를 4096 byte로 정의하였으며, 'Financial' trace 블록의 크기를 512 byte로 정의하였다. 또한, SSD Queue의 크기를 증가시키면서 실험하였다. 하지만, 최근성을 계산하기 위해 에이지(Age)와 재사용 간격(Reuse interval)을 1:1로 합산할 경우, 재사용 간격에 의해 계산되는 과거 블록 사용 정보가 최근 블록 사용 정보보다 더욱 우선되어 히트율이 오히려 감소하는 문제점이 있었다. 따라서 상기의 문제점을 개선하기 위해 가중치를 바꾸면서 실험을 하였고, 평균적으로 에이지(Age)와 재사용간격(Reuse interval)의 가중치의 비를 각각 3:1로 설정하는 것이 히트율을 최대한 향상시킬 수 있음을 확인하였다.

(표 2) 각 I/O workload trace에 따른 특징[13]
(Table 2) The specification of I/O workload

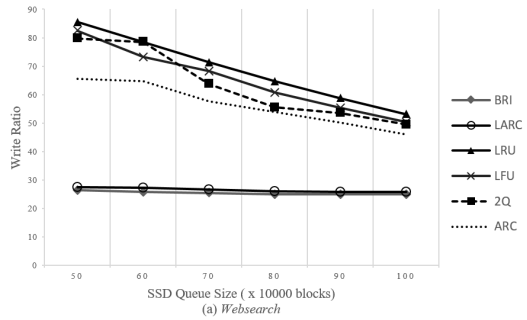
	Requests(x1000)		
	Read	Write	Total
Websearch	4578	0.99	4579
Financial	3046	653	3699

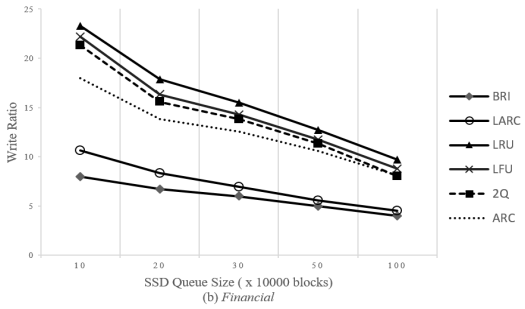
그림 6은 두 가지의 Trace파일을 이용하여 I/O 연산 후의 SSD 히트율을 보여준다. 그림에서 보는 것처럼 제안하는 BRI알고리즘이 Workload종류와 SSD Queue 크기에서 기존 교체 알고리즘보다 향상된 히트율을 보여준다.



(그림 6) Trace에 따른 히트율 비교
(Figure 6) The comparison of hit ratio of the schemes with different traces

그림 7은 두 가지의 Trace파일을 이용하여 SSD 쓰기율를 비교한 결과이다. 여기서 기술한 쓰기 율은 전체 액세스 요청대비 SSD 쓰기 연산 횟수를 뜻한다. SSD를 HDD의 캐시로 사용하는 Hybrid 저장시스템에서 블록교체는 쓰기연산을 일으키는 가장 큰 원인이다. 즉 블록교체가 적게 일어날수록, SSD의 수명향상을 기대할 수 있다. SSD 큐 크기가 커질수록 BRI알고리즘은 다른 알고리즘들에 비해 낮은 쓰기율을 보여준다.





(그림 7) Trace에 따른 쓰기율 비교
(Figure 7) The comparison of Write ratio of different schemes

5. 결 론

본 논문에서는 최근성과 참조빈도를 고려한 새로운 SSD 블록 교체 기법을 제안하였다. 블록의 성능과 수명 모두 향상시키기 위해 본 논문에서 제안하는 기법은 크게 두 가지의 방법으로 나눌 수 있다. 첫 번째 방법은 방출할 블록을 채택하기 위해 블록 우선순위를 활용하는 방법이다. 블록의 우선순위는 블록의 히트 수를 고려한 참조빈도와 블록 재사용 간격과 블록 에이지(Age)를 고려한 최근성을 고려하여 결정한다. 두 번째 방법은 SSD의 쓰기 연산 횟수를 줄이기 위해 Ghost Queue의 크기를 재조정하는 방법이다. 실험을 통해 기존의 블록 교체 알고리즘에 비해 본 논문에서 제안한 기법이 향상된 히트율과 감소된 쓰기 연산 횟수를 제공함을 확인하였다. 따라서 SSD 기반 스토리지 시스템에 본 논문에서 제안하는 기법을 적용한다면 SSD의 향상된 수명과 성능을 제공할 것으로 기대한다.

참 고 문 헌 (Reference)

[1] S. Huang, Q. Wei, C. Chen, D. Feng, “Improving flash-based disk cache with lazy adaptive replacement”, in Mass Storage Systems and Technologies (MSST), IEEE 29th Symposium on, IEEE, 2013, pp. 1-10. http://dx.doi.org/10.1109/MSST.2013.6558447

[2] Y. Liu, X. Ge, X. Huang, D. H. Du, “MOLAR: A Cost-Efficient, High-Performance SSD-Based Hybrid Storage Cache”, The Computer Journal, 2015. http://dx.doi.org/10.1093/comjnl/bxu156

[3] Q. Wei, L. Zeng, J. Chen, C. Chen, “A Popularity-Aware Buffer Management to Improve Buffer Hit Ratio and Write Sequentiality for Solid-State Drive”, IEEE Transactions on Magnetics, 49(6), 2013, pp. 2786-2793. http://dx.doi.org/10.1109/TMAG.2013.2249579

[4] A. Dan, D. Towsley, “An approximate analysis of the LRU and FIFO buffer replacement schemes”, Proceedings of the 1990 ACM SIGMETRICS Conference on Measurement and Modeling of Computer Systems, 1990, pp. 143-152. http://dx.doi.org/10.1145/98460.98525

[5] J.T. Robinson and M. V. Devarakonda, “Data cache management using frequency-based replacement”, Proceedings, ACM SIGMETRICS Conference on Measurement and Modeling of Computer Systems, 1990, pp. 134-142. http://dx.doi.org/10.1145/98460.98523

[6] D. Lee, J. Choi, J. H. Kim, S. H. 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”, IEEE Trans. Computers, vol. 50, no. 12, pp. 1352–1360, 2001. http://doi.ieeecomputersociety.org/10.1109/TC.2001.970573

[7] T. Johnson and D. Shasha, “2Q: A Low Overhead High Performance Buffer Management Replacement Algorithm,” in VLDB’94, Proceedings of 20th International Conference on Very Large Data Bases, September 1994, pp. 439–450. http://www.vldb.org/conf/1994/P439.PDF

[8] S. Jiang, X. Zhang, “LIRS: An Efficient low inter-reference recency set replacement policy to improve buffer cache performance”, 2002 ACM SIGMETRICS Conference on Measurement and Modeling of Computer Systems, 2002, pp. 31-42. http://dx.doi.org/10.1145/511334.511340

[9] N. Megiddo, D. Modha, “ARC: A Self-Tuning, Low Overhead Replacement Cache”, in Proceedings of the 2nd USENIX Conference on File and Storage Technologies, 2003, pp. 115-130. http://dl.acm.org/citation.cfm?id=1090708

[10] S. Park, D. Jung, J. Kang, J. Kim, J. Lee, “CFLRU: A Replacement Algorithm for Flash Memory”, In Proceedings of the international conference on Compilers,

architecture and synthesis for embedded systems, ACM, 2006, pp. 234-241.
http://dx.doi.org/10.1145/1176760.1176789

[11] H. Kim, S. Ahn, “BPLRU: A buffer management scheme for improving random writes in flash storage”, In Proceedings of the FAST, Vol8, 2008, pp. 1-14.
http://static.usenix.org/legacy/events/fast08/tech/full_papers/kim/kim.pdf

[12] G. Graefe, “The five-minute rule twenty years later, and how flash memory changes the rules”, in Proceedings of the 3rd international workshop on Data management on new hardware. ACM, 2007, pp.6.
http://dx.doi.org/10.1145/1363189.1363198

[13] UMass Trace Repository. [Online]. Available: http://traces.cs.umass.edu/index.php/Storage/Storage

[14] F. Chen, S. Jiang, X. Zhang, “SmartSaver: Turning Flash Drive into a Disk Energy Saver for Mobile Computers,”, Proceedings of the International Symposium on Low Power Electronics and Design, pp.412-417, 2006.
http://dx.doi.org/10.1109/LPE.2006.4271878

[15] Y. Kim, J. Kim, “ARC-H: Adaptive replacement cache management for heterogeneous storage devices”, Journal of Systems Architecture Vol 58(2), 2012, pp. 86-97.
http://dx.doi.org/doi:10.1016/j.sysarc.2011.12.002

◐ 저 자 소 개 ◑

유 상 현 (Sanghyun Yoo)
2014년 한신대학교 정보통신학과(공학사)
2014～현재 성균관대학교 전자전기컴퓨터공학과 석사과정
관심분야 : 유비쿼터스 컴퓨팅, 분산처리
E-mail : shyunyoo@skku.edu

김 경 태 (Kyung Tae Kim)
2003년 수원대학교 컴퓨터과학과(공학사)
2005년 성균관대학교 정보통신공학부 컴퓨터공학과(공학석사)
2013년 성균관대학교 정보통신공학부 컴퓨터공학과(공학박사)
2013～현재 성균관대학교 정보통신대학 연구교수
관심분야 : USN, UPTV, 이동 통신 시스템, 유비쿼터스 컴퓨팅
E-mail : kyungtaekim76@google.com

윤 희 용 (Hee Yong Youn)
1977년 서울대학교 전기공학과(공학사)
1979년 서울대학교 전기공학과(공학석사)
1988년 Univ. of Messachusetts at Amherst 컴퓨터공학과(공학박사)
1988년～1990년 Univ. of North Texas 조교수
1991년～1999년 Univ. of Texas at Arlington부교수
2000년～현재 성균관대학교 컴퓨터공학과 교수 및 유비쿼터스 컴퓨팅기술연구소 소장
관심분야 : 모바일 컴퓨팅, 분산처리, IoT, 스토리지시스템
E-mail : youn7147@skku.edu