논문 2020-15-33

플래시 변환 계층을 위한 DPW-LRU 캐시 교체 알고리즘 분석 및 개선

(Analysis and Improvement of the DPW-LRU Cache Replacement Algorithm for Flash Translation Layer)

이 형 봉, 정 태 윤*
(Hyung-Bong Lee, Tae-Yun Chung)

Abstract : Although flash disks are being used widely instead of hard disks, it is difficult to optimize for effective utilization of flash disks because overwrite in place is impossible and the power consumption and time required for read, write, and erase operations are all different. One of these optimization issues is a cache management strategy to minimize write operations. The cache operates at two levels: an operating system equipped with flash disks and a translation layer within the flash disk. Most studies deal with the operating system-level cache strategy. In this study, we implement and analyse the DPW-LRU algorithm which is one of the recently proposed operating system cache replacement algorithms to apply to FTL, and grope with some improvements. As a result of the experiment, the DPW-LRU algorithm maintained superiority even in the FTL environment, and showed better performance with a slight improvement.

Keywords : Cache Replacement Algorithm, CF-LRU, DPW-LRU, Flash Memory, FTL, LRU

Ⅰ. 서 론

진공관과 상반되는 고체 상태의 반도체 집적 회로를 이용하여 전원과 관계없이 데이터의 기록을 지속할 수 있는 저장 장치를 SSD (Solid-state Drive)라 부른다 [1]. 이러한 SSD의 어원에 충실하자면 2000년대 이전에 많이 사용되었던 EPROM이나 EEPROM도 모두 SSD에 해당되지만 이들은 저장 내용을 수정하기 위한 지우기 작업을 별도의 장치로 떼어내 진행해야 했다. 따라서, 이들을 하드디스크와 같이 컴퓨터의 보조기억 장치로 사용하는 일은 불가능하다. 이런 단점이 개선되어 신속한 온라인 지우기가 가능한 소자로 등장한 것이 플래시 메모리이고 [2], 최근에는 SSD를 NAND 플래시 메모리 기반 SSD로 지칭하는 경향이 있다 [3].

플래시 메모리는 하드 디스크의 헤드와 같은 기계적 장치를 필요로 하지 않기 때문에 접근 속도가 빠르다는 점이 가장 큰 장점이다. 반면에 표 1에서 보는 바와 같이 읽기, 쓰기, 지우기 연산을 위한 소요 시간과 소모 에너지가 모두 다르고 [4, 5], 제자리 덮어쓰기가 불가능하다는 아주 중대한 단점을 가지고 있다. 이와 같이 극복해야 할 단점의 다양성은 플래시 메모리의 효과적 활용을 위한 최적화 작업을 어렵게 한다. 이러한 최적화 작업을 지원하기 위해 SSD 내부에 약간의 RAM ([5]의 경우 256 MB)과 함께 펌웨어 탑재 공간을 제공하는데, 이를 플래시 변환 계층 즉, FTL (Flash Translation Layer)이라 한다. FTL의 주요 기능은 논리 블록과 플래시 물리 페이지의 매핑, 페이지 캐싱, 가비지 (garbage) 수집 등으로 이들은 플래시 메모리의 성능을 좌우한다 [6].

이 연구에서는 FTL 기능 중 캐시 교체 알고리즘에 집중하여, 가장 최근 연구 결과 중의 하나인

*Corresponding Author (tychung@gwnu.ac.kr)
Received: Aug. 31, 2020, Revised: Sep. 28, 2020, Accepted: Oct. 27, 2020.
H.B. Lee, T.Y. Chung: Gangneung-Wonju National University (Prof.)
※ 이 논문은 2020년도 정부 (산업통상자원부)의재원으로 한국산업기술진흥원 지원을 받아 수행된 연구임 (R0006229, 차세대 생명・건강산업생태계 조성사업).

ⓒ IEMEK J. Embed. Sys. Appl. 2020 Dec. 15(6) 289-297
ISSN : 1975-5066
http://dx.doi.org/10.14372/IEMEK.2020.15.6.289

표 1. 플래시 메모리의 연산별 접근 시간과 에너지 소모

Table 1. Access Time and Energy Consumption of Flash Memory by Operation

cost item \ operation	read	write	erase
time(㎲)	25	250	1500
energy(㎌)	0.5	7.5	40

운영체제 캐시 알고리즘 DPW (Dynamic Page Weight)-LRU [7]를 FTL 환경에 적용할 수 있는지의 타당성을 조명해보고 개선점을 모색한다. 이를 위하여, 2장에서 관련 연구를 살펴보고, 3장에서 DPW-LRU 알고리즘을 분석 및 구현하며, 4장에서 평가 실시 및 개선점을 제안한다. 그리고 마지막 5장에서 결론으로 이 논문을 맺는다.

Ⅱ. FTL 캐시 관련 연구

1. 캐시 검색 시간 성능

캐시가 제공해야할 기본 연산은 검색 메소드이다. 검색 시간을 단축하기 위해 도입하는 일반적인 방법은 그림 1과 같이 캐시를 그룹으로 나누고, 특정 논리 블럭에 대한 캐시를 해당 블럭 번호의 해싱 결과에 대응되는 그룹에 할당한다. 이때 검색은 필요에 따라 양방향 중 한 방향의 연결 리스트를 따라 순회한다. 그런데, 8K 크기의 캐시 16,384개(총 128MB)인 FTL 환경에서는 연결 리스트를 따라 검색하는 것 보다 해당 캐시 그룹 영역을 선형 탐색하는 방법이 더 효율적이라는 결과를 보인 연구가 있다 [8]. 이 경우에는 LRU 순서에 따른 검색이 아닌 랜덤 순서로 검색하는 결과인데, 그룹 크기에 관계없이 우수하다는 점을 보인다.

연결 리스트를 사용하지 않는 다른 방법으로 캐시 상단에 B+tree를 배치하는 연구가 있는데 [9], 추가 메모리가 요구되고 검색 성능은 캐시 영역 순차 탐색과 큰 차이를 보이지 않는다.

2. 쓰기 팽창을 억제하는 캐시 교체 알고리즘

플래시 메모리의 쓰기는 곧 가비지를 생산하고, 가비지를 수집하기 위한 지우기 연산이 이어진다. 플래시 메모리의 지우기는 여러 개의 페이지로 구성된 블록 단위로 이루어지고, 각각의 블록에는 가비지와 유효 페이지가 섞여 있기 때문에 유효 페이지를 다른 곳으로 복사하여 이동시키기 위한 또 다른 쓰기 작업이 필요하다. 즉, 플래시 메모리의 쓰기 작업은 또 다른 쓰기 작업을 유발하는 쓰기 팽창 과정으로 규정할 수 있다.

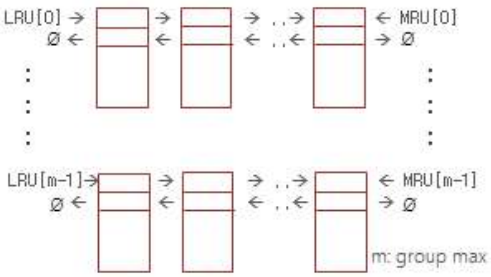

그림 1. 그룹핑에 의한 캐시 구성

Fig. 1 Cache Construction by Grouping

따라서, 플래시 메모리 위에서 작동하는 캐시에서는 쓰기 빈도를 낮추는 목표에 높은 비중을 두어야 한다 [10]. 일반적인 운영체제 캐시에서는 캐시 적중률에 우선순위를 두는 LRU 알고리즘을 적용하지만, LRU를 기반으로 하면서도 플래시 메모리의 쓰기 빈도를 줄이는 연구의 출발로 CF-LRU (Clean First LRU) [10]를 들 수 있다. CF-LRU는 그림 1의 LRU 캐시 리스트에서 각 그룹별로 단순히 $LRU\ [g]_0$을 희생 캐시로 선택하지 않고, LRU 리스트 상위 몇 개 (윈도우: $0 \sim w-1$) 중에서 클린(clean) 페이지가 존재하면 LRU 순위가 조금 낮더라도 교체 대상으로 선정한다. CF-LRU의 단점은 당분간 참조되지 않을 더티 (dirty) 페이지가 교체되지 않고 오래 머물러 캐시 전체의 효율성에 악영향을 미칠 수 있다는 점이다.

LRU-WSR [11]은 CF-LRU의 더티 페이지 존속 문제를 해결하기 위해 콜드 (cold)·핫 (hot) 개념과 2차 기회 (2nd chance) 알고리즘을 도입한다. 즉, 첫 참조이면 콜드, 두 번 이상 참조이면 핫으로 정의하고, 클린이거나 더티-콜드인 페이지를 동일 순위로 먼저 선택하여 교체한다. 다만 더티 페이지인 경우 MRU에서 LRU로 이동하는 동한 참조가 일어나 핫이 되었다면 콜드 상태로 다시 MRU→LRU 과정을 거치도록 함으로써 가급적 더티-핫 페이지들이 캐시에 남도록 유도한다. LRU-WSR은 클린-콜드와 클린-핫 페이지 사이에 차별이 없다는 아쉬움이 있는데, 이를 해결하기 위해 CCF-LRU (Cold Clean First LRU) [12]는 클린-콜드 페이지를 별도의 리스트에 최우선 교체 순서로 관리하고, 나머지는 다른 리스트에 관리하여 후순위로 LRU-WSR을 적용한다. 그러나 CCF-LRU는 최근 막 들어온 클린-콜드 페이지에 대한 고려가 없다.

AD-LRU (Adaptive Double LRU) [13]은 콜드 페이지와 핫 페이지를 위한 두 개의 리스트를 두고, 첫 참조 시 콜드 페이지에 삽입한 다음, 재 참조 이후에는 핫 리스트로 이동시킨다. 이 때, 희생 페이지는 콜드 리스트가 일정 크기 이상이면 콜드 리스트에서, 그렇지 않으면 핫 리스트에서 선택함으로써 CCF-LRU의 미흡한 점을 보완한다.

PT-LRU (Probabilistic Triplicate LRU) [14]는 클린-콜드, 핫 (더티-핫, 클린-핫), 더티-콜드 등 세 개의 리스트를 두고 클린-콜드에서 출발하여 재 참조 시 핫 리스트로 이동시킨다. 교체 페이지는 우선 클린-콜드 리스트 크기가 충분하면 여기서 선택한다. 클린-콜드 리스트에서 실패한 경우 확률에 따라 핫과 더티-콜드 리스트 중 한 곳을 선택하되 더티-콜드에 더 확률을 둔다. 교체 대상이 핫 리스트로 결정된 경우 LP_WSR 알고리즘에 의해 클린-핫 페이지를 교체 페이지로 선정하되 검색 도중에 발견된 다른 페이지들은 핫 리스트의 MRU (2차 기회)나 더티-콜드 (우선 순위 격상)로 이동시킨다.

위에서 살펴본 캐시 교체 알고리즘들은 참조경과시간, 청결여부, 재참조여부 등 세 가지 요인만을 고려하지만, 이 연구에서 분석·구현·개선할 DPW-LRU 알고리즘은 참조 패턴까지 고려하여 쓰기 팽창 관점에서 더욱 우수한 결과를 얻는다.

Ⅲ. DPW-LRU 알고리즘 분석 및 구현

1. DPW-LRU 알고리즘

■ 리스트 관리

AD-LRU에서 제안된 콜드, 핫 등 두 개의 리스트를 그대로 도입하되 이들을 각각 ER (Exchange Region)과 WR (Working Region)로 명명한다. ER은 페이지 교체 후보 영역을, WR은 페이지 유지 영역을 각각 의미한다. 페이지 첫 참조 시 읽기 참조는 ER에 삽입하고, 쓰기 참조는 곧바로 WR에 삽입한다. 이 후, ER 페이지에 재참조가 일어나면 WR의 MRU 위치로 이동시킨다.

■ 페이지 교체 및 희생 페이지 선택

WR에 페이지 삽입이 필요할 때 해당 리스트의 크기가 임계치를 초과하면 WR의 LRU 상위 몇 개를 대상으로 페이지 가중치 DPW를 구하여 가장 가벼운 페이지를 ER로 이동시킨다. 이 때, WR에서와 유사하게 ER의 크기가 임계치를 초과하면 LRU 알고리즘에 따라 희생 페이지를 선택하여 최종적으로

```
Input: Pno,   // Flash page number ↔ Pno_page
       Prw,   // Flash page read/write
       WR,    // the working region
       ER     // the exchange region
Output: Pno_page, Updated WR, ER, free_pages
 1: if Pno_page is in cache then
 2:     if Pno_page ∈ WR then
 3:       Move Pno_page to the MRU of WR;
 4:     else
 5:       if WR_size < WR_size_limit then
 6:         Insert Pno_page into the MRU of WR;
 7:       else
 8:         victim = DPW_LRU(WR);
 9          Move victim into MRU of ER;
10:         Insert Pno_page into MRU of WR;
11:       end if
12:     end if
13: else   // Pno_page is not in cache
13-1:  Pno_page = get_page_from_free_pages(Pno);
14:    if Prw == Read then
15:      if ER_size < ER_size_limit then
16:         Insert Pno_page into MRU of ER;
17:      else
18:         victim = LRU(ER);
19:         Evict victim to flash memory;
19-1:       Move victim to free_pages;
20:         Insert Pno_page into MRU of ER;
21:      end if
22:    else  // Prw == Write
23:      if WR_size < WR_size_limit then
24:         Insert Pno_page into MRU of WR;
25:      else
26:         victim1 = DPW_LRU(WR);
26-1:       if ER_size == ER_size_limit then
26-2:          victim2 = LRU(ER);
26-3:          Evict victim2 to flash memory;
26-4:          Move victim2 to free_pages;
26-5:       end if
27:         Move victim1 into MRU of ER;
28:         Insert Pno_page into MRU of WR;
29:         ~~Evict the LRU of ER to ash memory;~~
30:      end if
31:    end if
32: end if
33: return Pno_page;
```

그림 2. 보완된 DPW-LRU 알고리즘

Fig. 2 Compensated DPW-LRU Algorithm

빈 페이지를 확보한다. 즉, WR의 희생 페이지는 DPW 방법으로 선정하여 ER로 보내고, ER의 희생 페이지는 LRU 알고리즘으로 선정하여 빈 페이지를 확보하는데, 희생 페이지 선택 작업은 각 리스트의 크기가 한계를 초과할 때 이루어진다.

그림 2에 DPW-LRU의 리스트 관리 및 페이지 교체 알고리즘을 보였는데. 이 그림에서 라인 13-1, 19-1, 26-1~26-4, 29는 구현 과정에서 발

견된 미흡한 부분을 보완한 내용이다. 즉, 라인 13-1에서 플래시 페이지를 읽기 위한 프리 캐시 페이지를 확보하고, 라인 19_{-1}에서는 플러시 된 희생 페이지를 프리 리스트로 이동시킨다. 라인 26_{-1}~26_{-5}에서는 WR에서 ER로의 이동 페이지가 있을 때, ER에 여유 공간이 없다면 ER의 희생 페이지를 미리 방출시킨 후 이동 페이지를 ER에 삽입한다.

2. DPW (Dynamic Page Weight) 측정

DPW-LRU 알고리즘은 쓰기 빈도를 최소화하기 위한 캐시 페이지별 가중치에 참조경과시간, 클린여부, 재참조여부 외에 참조 패턴을 고려하기 위해 아래의 세 가지 인자를 정의하고 이들을 최종 DPW 산출에 적용하여 낮은 값일수록 높은 방출 우선순위를 부여한다.

■ TLI (Temporal Locality Interval)

TLI는 캐시 페이지가 평균 몇 번의 입·출력 요청마다 참조되었는지를 측정하는 인자로서, 시간적 참조 지역성을 의미한다. TLI가 낮은 상태 즉, 시간적 참조 지역성이 높은 페이지가 LRU 상위에 위치한다면 이 페이지는 오래전에 집중적으로 참조되었고 앞으로는 참조될 가능성이 낮다고 판단할 수 있고, 반대로, TLI가 높은 경우에는 앞으로도 드물고 규칙적으로 참조가 일어날 가능성이 높다고 판단할 수 있기 때문에, TLI가 낮은 페이지에 더 높은 페이지 교체 우선순위를 둔다. TLI를 얻기 위해서는 입·출력 요청에 대하여 매번 1씩 증가는 순서번호를 부여하고, 해당 순서에서 참조된 페이지 번호를 일렬로 기록해야 한다.

구현 과정에서 보완한 TLI 측정 수식을 식 1에 보였다. 식 1에서 S는 DPW를 측정할 윈도우 내 일어난 <입·출력 순서번호, 참조 페이지 번호> 쌍들의 시계열 전체 집합을, Rpid는 S 내 각 원소들의 참조 페이지 번호를, Ri는 S 내에서 참조 페이지 번호가 i인 참조를, ASki 는 S 내에서 참조 페이지 번호가 i인 것 중 k번째 참조의 입·출력 순서번호를, o는 참조 페이지 번호가 i인 것 중 마지막 참조 순서를 각각 의미한다.

$$P_i^{TLI} = \frac{\sum_{k=2}^{o}(AS_k^i - AS_{k-1}^i)}{|\{R_i \mid R_{pid} = i,\ R_i \in S\}|}. \quad (1)$$

■ EC (Eviction Cost)

EC는 DPW를 측정할 윈도우 기간 동안 측정된 읽기와 쓰기 지연의 상대적 비율을 의미한다. 후보 희생 페이지가 클린이라면 나중에 읽기를 동반할 것을 예상하여 읽기 지연 비율을, 더티 페이지라면 쓰기 지연 비율을 가중치로 적용한다. 그런데, 운영체제 환경과는 달리 FTL 환경에서는 읽기 지연과 쓰기 지연이 확정적이므로 식 2와 같이 고정된 수식을 사용할 수 있다 (읽기 지연 25us, 쓰기 지연 200us 전제).

$$P_i^{EC} = \begin{cases} \dfrac{25}{(25+200)}, & P_i \in \{clean\ pages\} \\ \dfrac{200}{(25+200)}, & P_i \in \{dirty\ pages\} \end{cases}. \quad (2)$$

■ RE (Recency)

페이지가 참조된 지 얼마나 오래 되었는가 즉, 참신성을 측정하는 인자로서 현재의 입·출력 순서번호와 후보 희생 페이지의 마지막 참조 시의 입·출력 순서번호간의 거리를 주된 측정값으로 사용하되, 식 3과 같이 현재의 입·출력 순서번호 대비로 변환하여 페이지 가중치에 적용한다. 식 3에서 AScurrent는 가장 최근의 마지막 입·출력 순서번호를, ASoi는 참조 페이지 번호가 i인 참조인 것 중에서 마지막 참조 시의 입·출력 순서번호를 각각 의미한다. RE 값이 크면 상대적으로 오래 전에 참조되었다는 의미 즉, 참신성이 낮다는 의미이므로 후보 희생 페이지 선택 가능성에 가점으로 작용한다.

$$P_i^{RE} = \frac{AS_{current} - AS_o^i}{AS_{current}}. \quad (3)$$

■ 최종 DPW 도출

식 1~3으로 측정한 TLI, EC, RE 인자를 식 4와 같이 종합하여 최종 DPW 값으로 도출하여 그 값이 가장 작은 페이지를 희생 페이지로 선택한다. 이 식에서 α는 페이지 가중 상수로서, 상대 비교하는 과정에서는 의미가 없기 때문에 1로 설정해도 무방하다. 식 4가 함축적으로 의미하는 바는 클린 페이지이고, 시간적 지역성이 낮으며, 참신성이 낮을수록 희생 페이지 우선순위를 높이자는 것이다.

$$P_i^{DPW} = \alpha \frac{P_i^{EC} \bullet P_i^{TLI}}{P_i^{RE}}. \quad (4)$$

3. DPW-LRU 알고리즘 구현

■ 참조 스트링 관리를 위한 시계열 리스트 관리

정확한 TLI 값을 얻기 위해서는 최근 일정 기간 내 모든 캐시 참조에 대하여 <입출력 순서번호, 참조 페이지 번호> 쌍을 기록해야한다. 이를 위한 일반적인 방법은 일정 크기의 배열을 기반으로 하는 환형 큐로 구현하는 것인데, 그 크기는 "캐시 페이지 개수 x 캐시 페이지 평균 재참조 회수" 만큼 할당되어야 한다. 디스크 시뮬레이션을 위한 트레이스 데이터 Financial1 [15]의 경우 LRU 알고리즘을 적용했을 때 평균 재참조 회수가 약 10회이므로 캐시 페이지 수가 16K 개인 FTL 환경에서는 1.2MB (16K x 10 x sizeof (int) x 2)가 추가로 필요하다. DPW-LRU 연구 [12]에서는 이에 대한 공간 복잡도가 O (1)이기 때문에 큰 문제가 되지 않는다고 하지만, 리소스가 극히 제한적인 FTL 환경에서는 이를 절약할 필요성이 있다

이를 위하여 이 연구에서는 시계열 리스트를 별도로 관리하지 않는 이동 합산 방법을 도입한다. 이동 합산은 그림 3과 같이 각각의 캐시에 바로 전 참조 시의 입·출력 순서번호를 기록하고 다음 번 참조 시 순서번호 차를 누적하는 방식이다. 그림 3에서 캐시의 pseq는 DPW 측정 시 마지막 입·출력 순서번호에 해당되고, 이를 이용하여 주어진 캐시의 DPW를 계산하는 프로시저는 그림 4와 같다.

■ 캐시 구성

이 연구에서는 일반적인 FTL RAM 크기인 256MB [5]의 절반인 128MB에 8K 크기의 캐시 16K개를 그림 1 형태의 128개로 그룹핑하고, 링크 순회 대신 구간 검색 방법 [8, 16]을 적용한다. 특정 LBA (Logical Block Address)의 캐시 그룹은 "LBA % 128"의 식으로 결정하고, 캐시 리스트는 각 그룹별로 FR (free), WR, ER 등 세 개로 운영하며, 특정 캐시는 셋 중 어느 한 리스트에 존재한다.

각 그룹의 ER 리스트의 최대 크기는 그룹별 캐시 개수 128의 30% (=38)로 설정한다. 이에 따라 그룹별 WR 리스트의 최대 크기는 89 (=128-38-1)로 결정된다. WR 리스트의 최대 크기가 90이 아닌 89인 이유는 그림 2의 DPW-LRU 알고리즘에서 캐싱을 먼저 하고 난 다음 페이지를 교체 (방출)하므로 항상 하나 이상의 빈 캐시가 유지되어야 하기 때문이다.

WR 리스트에서 페이지 교체 시 희생 페이지의 대상이 되는 즉, DPW 측정 대상이 되는 LRU 상위

```
void ftl_accounting_dpw(FCACHE *fcp)
{
    fcp->tl += IOSEQ() - fcp->pseq;
    fcp->tc++;
    fcp->pseq = IOSEQ();
}
```

그림 3. 시간적 지역성 카운팅 코드

Fig. 3 Code for Accounting of Temporal Locality

```
double ftl_get_dpw1(FCACHE *fcp)
{
    double  tli, re, ec;
    tli = (double)fcp->tl / fcp->tc;
    ec = (fcp->dirty) ? (double)200/(200+25)
                      : (double) 25/(200+25);
    re = (double)(IOSEQ() - fcp->pseq) /
                                    IOSEQ();
    return(tli*ec/re);
}
```

그림 4. DPW 계산 코드

Fig. 4 Code for Calculation of DPW

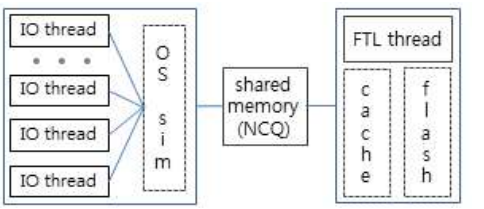

그림 5. 시뮬레이터 구조

Fig. 5 Architecture of the Simulator

윈도우 크기는 그룹별로 캐시 개수의 5% (128x0.05=6)로 설정한다.

IV. DPW-LRU 알고리즘 평가 및 개선

1. 실험 환경

DPW-LRU 알고리즘의 실험을 위해 [16]에서 소개된 그림 5의 구조를 갖는 시뮬레이터를 사용하고, 컴퓨팅 환경은 표 2와 같다.

2. 워크로드 및 실험 방법

현장에서 도출된 트레이스 데이터는 표 3과 같고 [15], 난수를 이용하는 합성 데이터는 표 4와 같다. 표 3의 I/O Ratio는 운영체제가 FTL에 요구

하는 페이지 입·출력 빈도의 비율을 의미한다.

이들 데이터를 표 5와 같이 네 가지 유형의 워크로드로 구성하여 LRU, CF-LRU, DPW-LRU* (보완된 DPW-LRU) 등 세 알고리즘의 플래시 쓰기 빈도 성능을 비교한다.

표 2. 실험 컴퓨팅 환경
Table 2. Experimental Computing Environment

Items	Specification
CPU	Intel E7500 (Duo Core), 2.93GHz
Memory	2.0GB
OS	LINUX (Ubuntu 16.04)

표 3. 현장 트레이스 데이터
Table 3. Real-world Trace Data

Data	Max LBA	N. of I/O	I/O Ratio(%)
Financial1	1,215,633	5,334,984	23/77
Financial2	1,215,222	3,699,194	82/18

표 4. 합성 데이터 (각 데이터별 스레드 4 개)
Table 4. Synthetic Data (4 threads per data)

Data	Specification
Syn_srd	sequential read 1K~9KB, 20,000 times, repeat 5 times
Syn_swr	sequential write 1K~9KB, 20,000 times, repeat 5 times
Syn_rrd	read from random location 1K~9KB, 20,000 times, repeat 5 times
Syn_rwr	write to random location 1K~9KB, 20,000 times, repeat 5 times
Syn_rrw	read and write at random location 1K~9KB, 20,000 times, repeat 5 times

표 5 워크로드 구성
Table 5. Configuration of Workload

Name	Combination of Data
Wk1	Financial1
Wk2	Financial2
Wk3	Financial1 + Financial2
Wk4	Financial1 + Financial2 + all of Syn*

표 6. 알고리즘 별 플래시 쓰기 빈도 (DPW-LRU)
Table 6. Real Write Count by Algorithm (DPW-LRU)

Data	WK1	WK2	WK3	WK4
LRU	756,543	142,403	1,043,722	2,172,552
CF-LRU	756,210	140,663	1,041,544	2,169,710
DPW-LRU	750,698	140,320	1,035,297	2,160,958

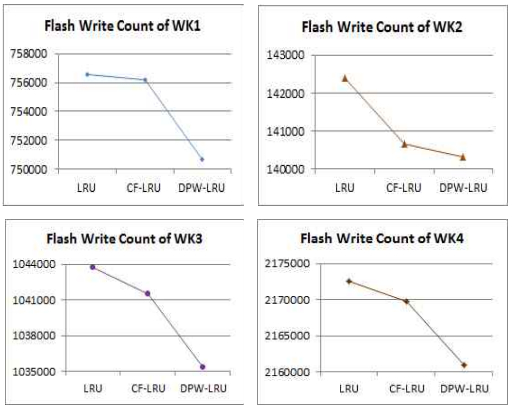

그림 6. DPW-LRU의 플래시 쓰기 빈도 성능
Fig. 6 Flash Write Count Performance of DPW-LRU

3. 실험 결과 및 개선

표 6, 그림 6에서 보는 바와 같이 플래시 쓰기 팽창 억제 관점에서 DPW-LRU의 우수성이 뚜렷하다. 그런데, DPW를 구하는 식 4가 쓰기 팽창을 억제하는 최선의 조합이라는 과학적 근거는 제시되어 있지 않기 때문에 휴리스틱에 의한 시행착오를 거쳐 얻어진 결과라고 판단된다. 이를테면 RE를 구하는 식 3에서 분모의 AScurrent는 모든 캐시 페이지에게 동일하므로 식 4의 DPW에 의한 캐시 페이지 간 차별화에 영향을 끼치지 않는다.

TLI, RE, EC 인자를 활용하되 더 낳은 결과를 얻을 수 있는 다른 인자 조합은 없는지를 시행착오를 통해 실험한 결과, 표 5의 모든 워크로드에 대하여 비율은 미미하지만 더 우수한 결과를 낳는 지표 산출 식을 다음과 같이 발견할 수 있었다.

• 식 3에서 분모를 제거하여 RE를 구하는 새로운 식 3'을 도입한다.

$$P_i^{RE'} = AS_{current} - AS_o^i. \qquad (3')$$

• DPW를 구하는 새로운 식 4'를 도입한다.

$$P_i^{DPW'} = \frac{P_i^{TLI}}{P_i^{RE'}} + P_i^{EC}. \qquad (4')$$

식 4'의 착안 동기는 가장 기본적인 페이지 교체 비용을 EC로 보고, 여기에 부수적으로 TLI와 RE를 고려하자는 데 있다.

표 7. 알고리즘 별 플래시 쓰기 빈도 (DPW-LRU')
Table 7. Real Write Count by Algorithm (DPW-LRU')

Data	WK1	WK2	WK3	WK4
DPW-LRU	750,698	140,320	1,035,297	2,160,958
DPW-LRU'	750,653	140,264	1,034,649	2,160,710

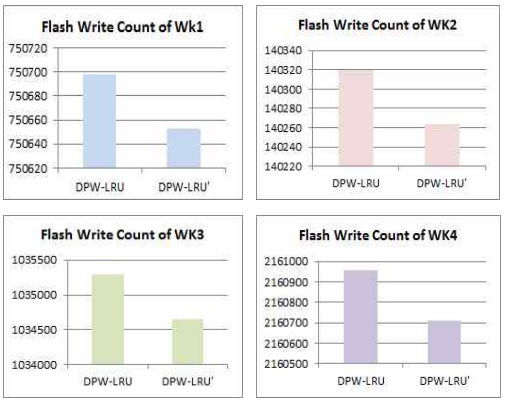

그림 7. DPW-LRU'의 플래시 쓰기 빈도 성능
Fig. 7 Flash Write Count Performance of DPW-LRU'

식 4'를 적용한 DPW-LRU' 알고리즘의 성능을 표 7, 그림 7에 보였는데, 모든 워크로드에 대하여 DPW-LRU보다 더 우수함을 확인할 수 있다.

V. 결 론

희생 페이지 선정을 위한 지표 획득에 과학적 근거 제시가 미흡하지만 DPW-LRU는 플래시 쓰기 팽창 억제 관점에서 지금까지 알려진 가장 우수한 캐시 교체 알고리즘 중의 하나이다. 이 연구에서는 DPW-LRU를 FTL 환경에서 구현하고, 여기서 제시된 지표 산출 식을 제시하고 검증하였다. 그 결과 실험 대상인 모든 워크로드에 대하여 안정적인 우수성이 확인되었다. 비록 그 차가 크지는 않지만 단 한 번의 쓰기라도 절약된다면 이는 곧 플래시 메모리의 용량과 수명에 영향을 미치므로 그 의의가 결코 적지 않은 것으로 본다. 이로써, DPW-LRU는 운영체제 환경뿐만 아니라 FTL 환경에도 적용 가능한 매우 유용한 캐시 교체 알고리즘인 것으로 판단된다.

References

[1] https://en.wikipedia.org/wiki/Solid-state_drive.
[2] B.M. Fujio, I. Hisakazu, "Semiconductor Memory Device and Method for Manufacturing the Same," US Patent 4531203, 1985.
[3] H.B. Lee, K.H Kwon, "Implementation of a Light Weight SSD Perpormance Evaluation Simulator based on Linux FUSE," Journal of Digital Contents Society," Vol. 20, No. 1, pp. 1-6, 2019 (in Korean).
[4] V. Mohan, T. Bunker, L. Grupp, S. Gurumurthi, M. R. Stan, S. Swanson, "Modeling Power Consumption of NAND Flash Memories Using FlashPower," IEEE Transactions on Computer-Aided Design of Integrated Circuit and Stsrems, Vol. 32, No. 7, pp. 1031-1044, 2013.
[5] Samsung Electronics, "Samsung SSD 850 Pro Data Sheet Rev. 3," 2017 (http://downloadcenter.samsung.com/content/UM/201711/20171115103115156/Samsung_SSD_850_PRO_Data_Sheet_Rev_3.pdf)
[6] C.N. Park, W.M. Cheon, J.G. Kang, K.H. Roh, and W.H. Cho, "A Reconfigurable FTL (Flash Translation Layer) Architecture for NAND Flash-Based Applications," ACM Transactions on Embedded Computing Systems, Vol. 7, No. 4, pp. 1-23, 2008.
[7] Y. Yuan, J. Zhang, G. Han, G. Jia, L. Yan, and W. Li, "DPW-LRU: An Efficient Buffer Management Policy Based on Dynamic Page Weight for Flash Memory in Cyber-Physical System," IEEE Access (Distributed Computing Infrastructure for Cyber-Physical Systems), Vol. 7, pp. 58810-58821, 2019.
[8] H.B. Lee, T.Y. Chung, "An In-depth Analysis and Improvement on Cache Mechanisms of SSD FTL," IEMEK J. Embed. Sys. Appl., Vol. 15, No. 1, pp. 9-15, 2019 (in Korean).
[9] S.H. Hwang, J.W. Kwak, "CL-Tree: B+ tree for NAND Flash Memory Using Cache Index List," Journal of the Korea Society of Computer and Information, Vol. 20, No. 4, pp.

1-10, 2015 (in Korean).
[10] S.Y, Park, D.W. Jung, J.U. Kang, J.S. Kim, J.W. Lee, "CFLRU: A Replacement Algorithm for Flash Memory," Proceedings of the 2006 international conference on Compilers, architecture and synthesis for embedded systems (CASES'06), pp. 234-241, 2006.
[11] H.Y. Jung, H.K. Shim, S.M. Park, SY. Kang, JH. Cha, "'LRU-WSR: Integration of LRU and Writes Sequence Reordering for Flash Memory," IEEE Transactions on Consumer Electronics, Vol. 54, No. 3, pp. 1215-1223, 2008.
[12] Z. Li, P. Jin, X. Su, K. Cui, L. Yue, "CCF-LRU: A New Buffer Eeplacement Algorithm for Flash Memory," IEEE Transactions on Consumer Electronics, Vol. 55, No. 3, pp. 1351-1359, 2009.
[13] P. Jin, Y. Ou, T. Härder, Z. Li, "AD-LRU: An Efficient Buffer Eeplacement Algorithm for Flash-based Databases," Data Knowledge Engineering, Vol. 72, pp. 83-102. 2012.
[14] J. Cui, W. Wu, Y. Wang, Z. Duan, "PT-LRU: A Probabilistic Page Replacement Algorithm for NAND Flash-based Consumer Electronics," IEEE Transactions on Consumer Electronics, Vol. 60, No. 4, pp. 614-622, 2014.
[15] UMass I/O Trace Repository (http://traces.cs.umass.edu/index.php/Storage/Storage)
[16] H.B. Lee, T.Y. Chung, "A Comparative Analysis on Page Caching Strategies Affecting Energy Consumption in the NAND Flash Translation Layer," IEMEK J. Embed. Sys. Appl., Vol. 13, No. 3, pp. 109-116, 2018 (in Korean).

Hyung-Bong Lee (이 형 봉)

He received the B.S. and M.S. degrees in Computer Science from Seoul National University, Seoul, Korea, in 1984 and 1986 respectively. He received his Ph.D. degree in Computer Science from Kangwon National University, Chuncheon, Korea, in 2002. From 1986 to 1994, he was a senior engineer in Computer R&D Division of LG Electronics. From 1995 to 1998, he was with DEC (Digital Equipment Corporation) as a UNIX consultant. From 1999 to 2003, he was an Associate Professor at Honam University, Gwangju, Korea. Since 2004, he has been a Professor in the Department of Computer Science & Engineering at Gangneung-Wonju National University, Wonju, Korea. His current research interests include embedded systems, wireless sensor networks, and data mining algorithms.

Email: hblee@gwnu.ac.kr

Tae-Yun Chung (정 태 윤)

He received the B.S., M.S., and Ph.D. degrees in the School of Electrical & Computer Engineering at Yonsei University, Seoul, Korea in 1987, 1989, and 2000 respectively. From 1989 to 1996, he was a Research Engineer of Samsung Advanced Institute of Technology. From 1996 to 2001, he was with Samsung Electronics as a Senior Research Engineer. From 2000 to 2001, he was a vice chair of International DVD-Forum. Since 2001, he has been with the Department of Electronics Engineering at Gangneung -Wonju National University, Gangneung, Korea, and is currently a Professor. Since 2004, he has been with GEMS-CRC (Gangwon Embedded Software Cooperative Research Center, Gangneung, Korea) as the Chef. His major fields are image signal processing, digital video encoding, multimedia, copy protection , and his current research interests are embedded system, sensor network, video encoding.

Email: tychung@gwnu.ac.kr