WWCLOCK: 플래시 메모리의 비대칭적 입출력 비용을 고려한 페이지 교체 알고리즘

(WWCLOCK: Page Replacement Algorithm Considering Asymmetric I/O Cost of Flash Memory)

박 준 석 † 이 은 지 †
(Junseok Park) (Eunji Lee)
서 현 민 † 고 건 ††
(Hyunmin Seo) (Kern Koh)

요 약 낸드 플래시 메모리는 하드디스크와 달리 읽기 입출력과 쓰기 입출력이 소모하는 시간 및 전력량이 다르며 그 비율은 SLC, MLC, SSD 등 다양한 형태에 따라 상이하다. 특히 최근에는 내장 메모리 장치와 함께 외장 메모리 카드 또는 USB 메모리를 동시에 사용하는 경우도 증가하고 있어서, 버퍼 캐시 교체 알고리즘을 설계하는 데 있어서 페이지의 재참조 확률뿐 만 아니라 접근 장치와 참조 종류에 따른 입출력 비용을 함께 고려해야 한다. 본 논문은 페이지의 참조 빈도(frequency), 최근성(recency) 정보와 함께 읽기와 쓰기의 입출력 비용을 직접적으로 고려하는 WWCLOCK (Write-Weighted CLOCK) 알고리즘을 제안한다. WWCLOCK은 입출력 비용이 다른 다양한 2차 저장장치에 대해 적용 가능하며, CLOCK에 가까운 낮은 시간 및 공간 복잡도를 갖고 있다. 트레이스 기반 시뮬레이션을 통해 제안된 알고리즘이 LRU 알고리즘에 비해 전체 입출력 실행 시간을 평균 36.2% 감소시킴을 보인다.

키워드 : 버퍼 캐시 교체 알고리즘, 비용 인식, 이기종 저장장치

Abstract Flash memories have asymmetric I/O costs for read and write in terms of latency and energy consumption. However, the ratio of these costs is dependent on the type of storage. Moreover, it is becoming more common to use two flash memories on a system as an internal memory and an external memory card. For this reason, buffer cache replacement algorithms should consider I/O costs of device as well as possibility of reference. This paper presents WWCLOCK(Write-Weighted CLOCK) algorithm which directly uses I/O costs of devices along with recency and frequency of cache blocks to selecting a victim to evict from the buffer cache. WWCLOCK can be used for wide range of storage devices with different I/O cost and for systems that are using two or more memory devices at the same time. In addition to this, it has low time and space complexity comparable to CLOCK algorithm. Trace-driven simulations show that the proposed algorithm reduces the total I/O time compared with LRU by 36.2% on average.

Key words : Buffer Cache Replacement Algorithm, Cost-aware, Heterogeneous storage

1. 서 론

플래시 메모리는 비휘발성 메모리이며 그 크기가 작고 무게가 가벼울 뿐만 아니라 물리적인 충격에 강하여 휴대폰, PMP, PDA, MP3 플레이어 등의 휴대용 기기 대부분에 내장되고 있으며, 이와 더불어 외부 메모리 카드 형태로도 널리 사용되고 있다.

그러나 플래시 메모리는 표 1에 요약한 바와 같이 장치의 종류에 따라 쓰기(write) 입출력이 읽기(read) 입출력에 비해 약 8배에서 13배 더 많은 시간과 전력을 소모한다는 특징을 갖고 있다[1]. 특히 소형 휴대용 내장형 시스템에서 내장 메모리와 함께 외장 메모리 카드를 사용하는 경우 또는 SSD가 탑재된 랩톱(laptop) 컴퓨터에서 USB 메모리 장치를 부착하여 사용하는 경우 등 두 개 이상의 이기종 장치가 동시에 사용되는 경우에는 저장장치마다 서로 다른 입출력 비용을 갖게 된다. 이러한 시스템에서 버퍼 캐시 교체 기법은 각 저장장치와 입출력 종류에 따른 비용을 인식하고 입출력으로 인한 전체 전력 소모와 시간 지연을 최소화하도록 교체 대상을 선택해야 한다[2,3].

본 논문은 각 버퍼 캐시 블록의 중요도를 계산함에

· 연구는 2008년 정부(교육과학기술부)의 재원으로 한국학술진흥재단의 지원을 받아 수행되었음(KRF-2008-314-D00344). 또한 서울대학교 컴퓨터연구소의 연구장비 및 공간을 지원받아 이루어졌음
· 이 논문은 2009 한국컴퓨터종합학술대회에서 'WWCLOCK: 플래시 메모리의 비대칭적 입출력 비용을 고려한 페이지 교체 알고리즘'의 제목으로 발표된 논문을 확장한 것임

† 학생회원 : 서울대학교 컴퓨터공학부
redo@oslab.snu.ac.kr
ejlee@oslab.snu.ac.kr
yzzmin@oslab.snu.ac.kr
†† 종신회원 : 서울대학교 컴퓨터공학부 교수
kernkoh@oslab.snu.ac.kr

논문접수 : 2009년 8월 13일
심사완료 : 2009년 10월 5일

정보과학회논문지: 컴퓨팅의 실제 및 레터 제15권 제12호(2009.12)

표 1 SLC 및 MLC 낸드 플래시 메모리의 주요 특징

Type	SLC	MLC
Page Size	2KB+64B	4KB+128B
Block Size	128KB+4KB	512KB+16KB
Page Read Time	25us	60us
Page Write Time	200us	800us
Block Erase Time	1.5ms	1.5ms
Operating Characteristics	3.3V/15mA	3.3V/15mA
Standby Power Dissipation	3.3V/10uA	3.3V/10uA
Cost per GB (mid 2007)	$10.37	$6.81

있어서 재참조 가능성과 함께 입출력 비용을 직접적으로 고려하는 WWCLOCK (Write-Weighted CLOCK) 알고리즘을 제안한다. WWCLOCK은 각 캐시 블록에 대해 그 블록의 재참조 확률을 나타내는 최근성(recency), 빈도(frequency), 그리고 입출력 비용을 반영하는 가중치 요소를 하나의 값으로 통합한 ECS(Expected Cost Saving, 기대 비용 절감) 값을 계산하고, ECS 값이 임계치(threshold)보다 낮은 페이지를 교체한다. 트레이스 기반 시뮬레이션을 통해 제안된 알고리즘이 LRU 알고리즘에 비해 평균 36.2% 감소된 전체 입출력 실행 시간 성능을 가짐을 보인다.

2. 관련 연구

LRFU[4]와 LIRS[5]는 하드디스크 기반의 시스템에서 캐시 블록의 최근성과 빈도를 함께 고려하여 캐시 적중률을 최대화시키고자 제안된 알고리즘이다. LRFU는 각 캐시 블록에 대해 그것의 중요도를 나타내는 CRF(Combined Recency and Frequency)값을 유지하는데, CRF값은 참조가 이루어질 때마다 1씩 증가시킴으로써 빈도를 반영하고, 1 블록 참조마다 모든 캐시 블록의 CRF 값을 일정 비율로 감쇠(decay)함으로써 최근성을 반영한다. 또한 감쇠 비율을 결정하는 λ값에 따라 최근성을 얼마나 중요하게 반영할지를 조절할 수 있다.

CFLRU[6]와 LRU-WSR[7]은 플래시 메모리 환경에서 읽기 입출력과 쓰기 입출력이 전력 소모량 측면에서 비용이 다르다는 점에 착안하여 기존의 캐시 적중률 최적화 목표에서 벗어나 캐시 적중률은 다소 낮아지더라도 더티(dirty) 페이지보다 클린(clean) 페이지를 먼저 추방함으로써 쓰기 입출력 횟수를 감소시켜 전체 전력 소모량을 낮추는 버퍼 캐시 관리 정책이다. CFLRU는 추방할 페이지를 선택하는 과정에서 LRU(Least Recently Used) 위치로부터 window_size 값으로 지정된 개수의 페이지들을 검사하여 클린(clean) 페이지가 존재하면 클린 페이지를 추방하고 그렇지 않으면 LRU 위치에 있는 더티 페이지를 추방하는 방식으로 동작한다. 그러나 window_size 범위 내에 클린 페이지가 존재할 경우 아무리 오래 전에 사용되었던 더티 페이지라도 메모리에 계속 잔류할 수 있다는 문제점을 갖고 있다. 또한 CFLRU는 입출력 비용을 인식하지 못하고 읽기와 쓰기 입출력의 비용이 다르다는 점만 활용한 방식이기 때문에 다른 매체로 2차 저장장치를 변경할 경우 또는 두 개 이상의 이기종 저장장치가 동시에 사용되는 환경에 적응하기 어렵다.

3. 제안하는 알고리즘

3.1 개념

WWCLOCK(Write-Weighted CLOCK) 알고리즘은 캐시 블록의 재참조 가능성과 입출력 비용을 ECS (Expected Cost Saving, 기대 비용 절감) 값으로 통합하여 유지한다. 그림 1은 과거 n회 접근되었던 캐시 블록 p에 대한 ECS 값을 계산하는 수식을 나타낸다. 수식을 통해 알 수 있듯이 ECS 값은 해당 블록에 대해 과거에 발생했던 각 참조의 입출력 비용에 시간 진행에 의한 감쇠를 적용한 뒤 합산한 값이며, 이 값이 높은 블록일 수록 메모리에 보관하는 것이 전체 전력 소모량 및 실행 시간을 최소화시킬 수 있음을 나타낸다. 수식에서 n은 해당 페이지의 전체 과거 참조 횟수, W_i는 과거 i 번째 참조의 종류에 따른 가중치, d는 감쇠 비율(decay factor), δ_i는 과거 i 번째 참조가 이루어진 시점으로부터 현재까지의 시간을 감쇠 주기 횟수로 표현한 값이다. 가중치는 해당 참조와 연관된 장치의 입출력 비용을 반영하는 값이다.

$$ECS_p = \sum_{i=1}^{n} W_i \left(\frac{1}{d}\right)^{\delta_i}$$

그림 1 캐시 블록 p에 대한 ECS 값을 계산하는 수식

각 캐시 블록의 ECS 값은 참조의 최근성을 함께 고려하기 위해 시간이 지남에 따라 감쇠(decay)된다. 감쇠를 통해 각 블록의 중요도는 과거의 참조를 완전히 무시하지는 않지만 최근의 참조에 비해서는 낮게 평가하여 계산된다. 그림 2는 감쇠에 대한 예를 보여주고 있다.

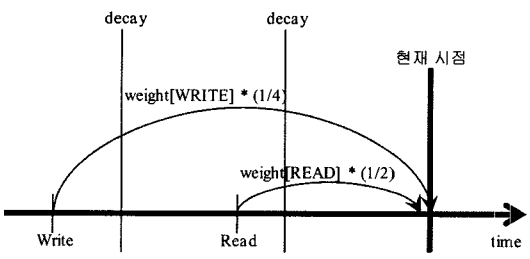

그림 2 과거 참조 기록에 의한 영향의 감쇠

이 방식의 ECS 값 계산은 ECS 값이 알려진 특정 시점으로부터 현재 시점까지의 감쇠 주기 횟수만 알면 그보다 과거의 참조 시점과 참조 종류의 정보를 유지하지 않아도 현재의 ECS 값을 재계산할 수 있다[4].

3.2 WWCLOCK 알고리즘의 구현

WWCLOCK은 LRFU와 유사하게 1개의 힙(heap) 자료구조를 이용하여 구현할 수 있다. 이때 감쇠는 논리 시간의 흐름과 동일하게 발생하는데, 즉 1 블록 참조마다 모든 캐시 블록에 대해 1 감쇠 주기를 진행시킨다. 이 구현 방식에서 실제로 ECS 값을 재계산해야 하는 블록은 전체 캐시 블록 중 현재 참조된 블록과 힙 구조상 그것의 하위 트리(subtree)에 해당하는 블록들로 한정되고 나머지 블록들의 ECS 값 재계산은 나중으로 미룰 수 있다. 그러나 모든 참조에 대해 CRF의 증가량이 1로 동일한 LRFU와 달리 WWCLOCK에서는 참조 종류에 따라 가중치 값이 다르기 때문에 새로운 블록이 캐시에 추가될 때마다 전체 캐시 블록에 대해 ECS 값을 재계산해야 한다.

버퍼 캐시 교체 알고리즘의 시간적 오버헤드를 줄이기 위해 WWCLOCK은 힙 구조 대신 CLOCK과 같은 환형 큐(circular queue) 구조로 구현한다. CLOCK 형태의 관리는 전체 블록 중 가장 ECS 값이 작은 블록을 추방하는 대신 클락 핸드(clock hand)가 지시하는 블록의 ECS 값을 임계치(threshold)와 비교하여 임계치 이상이면 해당 블록의 ECS 값에 1회의 감쇠를 적용한 뒤 다음 블록으로 클락 핸드를 이동시키고, 임계치 이하의 ECS 값을 갖는 블록을 발견할 경우 해당 블록을 교체한다. 즉, 클락 핸드가 1회 돌아오는 것이 해당 블록에 대한 1 감쇠 주기가 된다. 새로 캐시에 추가되는 블록은 클락 핸드의 바로 뒤에 연결한다. 매 참조마다 참조된 블록에 대해 해당 스토리지의 참조 종류에 따른 가중치만큼 ECS 값을 증가시킨다. 그림 3은 참조와 클락 핸드에 의한 감쇠 적용에 따라 ECS 값을 재계산해 나가는 예를 보여주고 있다.

이 구현 방식에서 블록 참조 시에는 감쇠 계산이 전혀 필요 없으며 참조된 블록의 ECS 값을 현재 참조 종류에 따른 W값만큼 증가시키는 덧셈만 1회 실행한다. 캐시 블록을 교체해야 하는 경우에는 임계치보다 낮은 ECS 값을 갖는 블록을 발견할 때까지 클락 핸드가 이동하게 되는데, 이때 클락 핸드가 지나간 블록들에 대해 감쇠를 1회씩 적용하게 된다. 알고리즘의 공간 오버헤드는 LRFU와 같이 과거의 참조 시각이나 참조 종류 등의 정보를 유지할 필요없이 각 블록에 대해 1개의 ECS 값만 유지하면 된다.

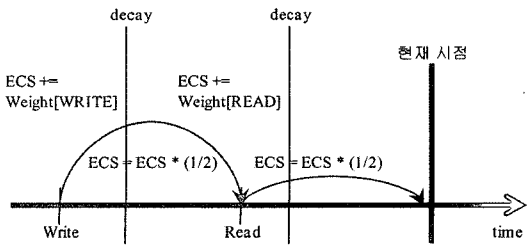

그림 3 참조와 감쇠에 대한 ECS 값의 재계산

LRFU와 마찬가지로 WWCLOCK도 감쇠 비율 d의 값에 따라 얼마나 참조의 최근성을 중시하여 평가할지를 조절할 수 있다. 감쇠 비율을 1에 가까운 값으로 하면 감쇠가 거의 이루어지지 않게 되므로 참조의 최근성을 무시하고 빈도 및 입출력 비용만을 고려하게 된다. 반대로 감쇠 비율을 무한대로 하고 임계치를 1로 설정하면 클락 핸드가 1회 지나갈 때 블록의 ECS 값이 0으로 초기화되고 클락 핸드가 다시 돌아오기 전까지 1회 이상 참조된 블록을 캐시에 유지하므로 CLOCK과 동일하게 동작한다.

4. 성능 평가

WWCLOCK 알고리즘의 성능을 측정하기 위해 트레이스 기반 시뮬레이션을 실시하였다. 트레이스는 리눅스 커널 2.6의 운영체제에서 gcc 컴파일러로 gnuplot 프로그램 코드를 컴파일하고 빌드하는 과정에서 발생하는 정보를 파일시스템 계층에서 수집하여 사용하였다. 트레이스의 상세한 정보는 표 2 및 그림 4에 수록하였다.

2차 저장장치는 MLC 낸드 플래시 메모리를 사용하는 환경을 시뮬레이트하였다. MLC 낸드에 대해 읽기 참조에 대한 가중치는 1을, 쓰기 참조에 대한 가중치는

표 2 트레이스의 특성

트레이스	블록 읽기	블록 쓰기	메모리 사용량 (클린 / 더티)
gcc	44.1MB (82.8%)	9.2MB (17.1%)	3.6MB (32.6%) / 7.4MB (67.4%)

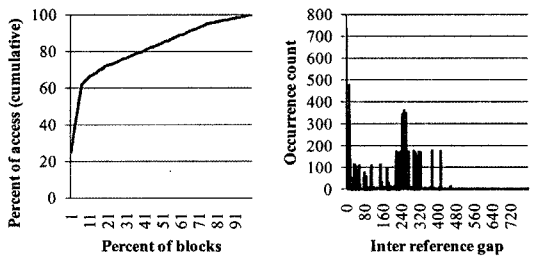

그림 4 gcc 트레이스의 지역성(locality)과 재참조 간격(inter reference gap) 특성

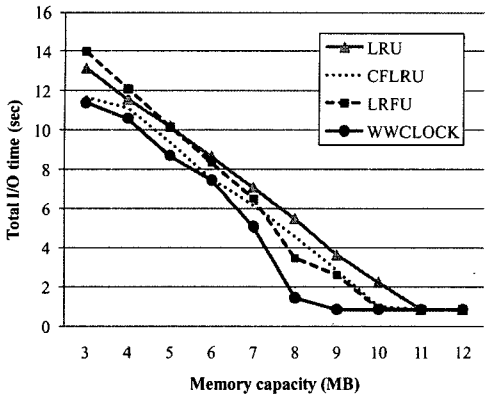

그림 5 gcc 워크로드에서의 전체 입출력 실행 시간

13을 사용하였다. 임계치는 1을 사용하였고 감쇠 요소 d의 값은 2를 사용하였다.

성능 비교를 위해 LRU, CFLRU, LRFU 알고리즘을 동일한 시뮬레이터 환경에 구현하였다. CFLRU의 동작에 필요한 *window_size* 값은 전체 캐시 블록 수의 25%를 사용하였다. LRFU의 실행에 필요한 λ값은 해당 트레이스에서 최소 입출력 실행 시간을 가지는 0.0001로 하였다.

그림 5는 gcc 워크로드에서 WWCLOCK을 전체 입출력 실행 시간 측면에서 다른 알고리즘과 비교한 것이다. 그래프의 모든 실행 시간은 강제 미스(compulsory miss)에 의한 시간을 포함한 값이다. LRU 알고리즘은 읽기와 쓰기 입출력의 비용 차이를 전혀 고려하지 않기 때문에 전체 입출력 실행 시간이 가장 크게 나타난다. LRFU는 입출력 비용을 고려하지는 않으나 재참조 확률을 고려함에 있어서 참조의 최근성과 빈도를 모두 고려하기 때문에 LRU보다 낮은 실행 시간을 가진다. CFLRU는 읽기와 쓰기 입출력 비용이 다르다는 가정하에 설계된 알고리즘이지만, 비용 차이의 양을 인식하지 못하며 블록의 재참조 확률을 고려함에 있어서도 참조의 빈도를 고려하지 않는다. 결과적으로 MLC 낸드 플래시 메모리와 같이 입출력 비용이 다른 장치나 다른 워크로드에 대해서는 window_size 파라메터를 재조정해야 적절한 성능을 얻을 수 있다.

실험한 워크로드에서 WWCLOCK 알고리즘은 전체 입출력 시간 측면에서 LRU에 비해 평균 36.2% 향상된 성능을 보였으며, CFLRU에 비해서는 평균 23.5%, LRFU에 비해서는 평균 26.3% 향상된 성능을 보였다.

WWCLOCK은 임계치를 조정함으로써 매 교체마다 얼마나 작은 ECS 값을 찾을 때까지 캐시 블록들을 검색할 지 결정할 수 있다. 그러므로 이 값은 시스템의 입출력 성능에 영향을 줄 뿐만 아니라 알고리즘의 실행

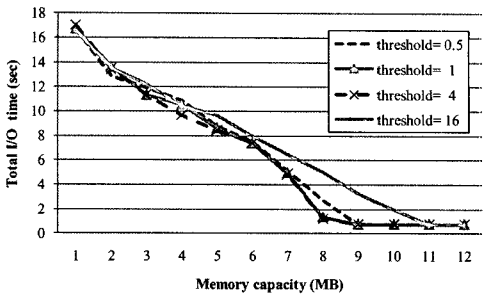

그림 6 임계치와 전체 입출력 성능

시간에도 영향을 미친다. 그림 6은 임계치를 다르게 하였을 때 입출력 성능의 변화를 나타낸 그래프이다. 1~4 정도의 임계치는 성능 상 큰 차이가 없고 16 부터는 성능이 감소하기 시작한다. 0.5와 같이 지나치게 작은 임계치는 오히려 입출력 성능에 악영향을 미칠 수 있는데, 이는 매우 작은 ECS 값을 가진 블록을 찾기 위해 클락 핸드가 많은 블록을 검색하는 과정에서 과도한 ECS 값 감쇠가 이루어지고, 그에 따라서 블록간 ECS 값의 차이가 감소하기 때문이다.

그림 7은 추방할 캐시 블록 1개를 찾기 위해 검색한 블록 수의 평균을 전체 캐시 블록 수와 임계치의 변화에 따라 나타낸 것이다. 너무 낮은 임계치는 WWCLOCK이 가장 ECS 값이 작은 블록을 찾을 때까지 검색을 계속하게 만들기 때문에 알고리즘의 시간적 오버헤드를 증가시킨다. 반대로 지나치게 높은 임계치는 가장 ECS 값이 높은 블록이라도 추방할 수 있기 때문에 알고리즘의 시간적 오버헤드는 낮지만 입출력 성능을 최소화하는데는 실패한다. 실험 결과 1에서 4 정도의 임계치가 시간적 오버헤드뿐만 아니라 입출력으로 인한 실행 시간 및 에너지 소모량 감소 측면에서도 적절한 것으로 나타났다. 임계치가 1이고 버퍼 캐시의 크기가 8MB 일 때 하나의 블록을 교체하기 위해 평균 10개의 블록을 조사하는데, 이는 전체 캐시 블록 중 0.49%에 해당하는 양이다. 임계치가 4일 때에는 평균 블록 검색

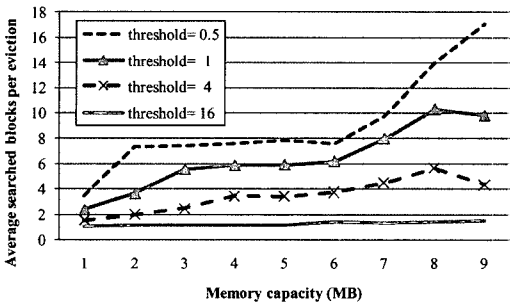

그림 7 임계치와 교체 당 평균 검색 블록 수

수가 더욱 낮아져 8MB의 버퍼 캐시 환경에서 전체 캐시 블록 중 0.28%에 해당하는 블록만을 검색한다.

5. 결 론

플래시 메모리 기술의 발전에 따라 플래시 메모리 기반 시스템도 SLC 낸드, MLC 낸드, SSD 등 다양한 입출력 성능을 가진 장치를 탑재할 수 있게 되었다. 특히 내장 메모리와 함께 외장 메모리 카드를 장착하여 사용하는 경우 한 시스템에서 둘 이상의 이기종 장치를 동시에 사용하는 환경이 된다. 이때 시스템의 일부 메모리 장치를 다른 장치로 변경할 경우 입출력 비용이 달라질 수 있으며 또한 어느 장치에 입출력을 하는가에 따라 다른 입출력 비용이 발생하게 된다. 이러한 환경에서 버퍼 캐시 교체 알고리즘은 블록의 재참조 가능성과 함께 각 장치와 참조 종류에 따른 입출력 비용을 인식하고 직접적으로 고려하여 전체 입출력 비용이 최소화되도록 해야 한다.

본 논문에서는 블록 참조의 최근성, 빈도와 함께 입출력 비용을 통합적으로 고려하는 페이지 교체 알고리즘을 제안하였다. 제안된 알고리즘은 MLC 낸드 플래시 메모리 등 입출력 비용이 다른 다양한 종류의 2차 저장 장치에 대해서도 적용 가능하며, CLOCK에 가까운 낮은 시간 및 공간 오버헤드를 가진다. 트레이스 기반 시뮬레이션을 통해 제안된 알고리즘이 LRU 알고리즘에 비해 평균 36.2% 감소된 전체 입출력 실행 시간 성능을 가짐을 보였다.

참 고 문 헌

[1] L.-P. Chang, "Hybrid solid-state disks: Combining heterogeneous NAND flash in large SSDs," *Proceedings of the 2008 conference on Asia and South Pacific design automation*, pp.428-433, 2008.

[2] Y.-J. Kim and J. Kim, "Device-aware cache replacement algorithm for heterogeneous mobile storage devices," *Proceedings of the 3rd International Conference on Embedded Software and Systems (ICESS)*, pp.13-24, 2007.

[3] B. C. Forney, A. C. Arpaci-Dusseau, R. H. Arpaci-Dusseau, "Storage-Aware Caching: Revisiting Caching for Heterogeneous Storage Systems," *Proceedings of the 1st USENIX Conference on File and Storage (FAST)*, 2002.

[4] D. Lee, J. Choi, J. H. Kim, S. H. Noh, S. L. Min, Y. Cho, C. S. Kim, "LRFU: A Spectrum of Policies that Subsumes the Least Recently Used and Least Frequently Used Policies," *IEEE Transactions on Computers*, vol.50, issue 12, 2001.

[5] S. Jiang, X. Zhang, "LIRS: An efficient low inter-reference recency set replacement policy to improve buffer cache performance," *Proceedings of the ACM SIGMETRICS conference*, 2002.

[6] S.-Y. Park, D. Jung, J.-U. Kang, J.-S. Kim, J. Lee, "CFLRU: a replacement algorithm for flash memory," *Proceedings of the international conference on Compilers, Architecture and Synthesis for Embedded Systems (CASES)*, pp.234-241, 2006.

[7] H. Jung, H. Shim, S. Park, S. Kang, and Jaehyuk Cha, "LRU-WSR: Integration of LRU and Writes Sequence Reordering for Flash Memory," *IEEE Transactions on Consumer Electronics*, vol.54, issue 3, pp.1215-1223, 2008.