모바일 저장장치 성능 향상을 위한 두 캐시 계층 통합 주소 변환 테이블 관리기법*

김윤아○, 최인혁, 이성진†, 김지홍
서울대학교 컴퓨터공학부
† 대구경북과학기술원 전기전자컴퓨터공학과
{yoonakim, ihchoi, jihong}@davinci.snu.ac.kr, †sungjin.lee@dgist.ac.kr

Two L2P Table Cache Layers' Integrated Management Techniques for Improving Mobile Storage Performance

Yoona Kim○, Inhyuk Choi, Sungjin Lee†, and Jihong Kim
Department of Computer Science and Engineering, Seoul National University
† Department of Electrical Engineering and Computer, DGIST

요 약

저장장치의 크기가 점차 커지며 플래시 기반 저장장치의 주소 변환 테이블 관리에 필요한 메모리 공간의 요구량 또한 커지는 추세이다. 모바일 저장장치로 사용되는 UFS의 경우 시스템 및 하드웨어 특성상 UFS 내장 메모리 공간을 늘리기에 많은 제약이 존재하여 해당 테이블을 관리하는데 어려움이 있다. 이를 보완하기 위해, 호스트 메모리를 사용해 주소 변환 테이블을 적재할 수 있는 HPB 기법이 제안되었다. 하지만, 본 논문에서는 호스트 메모리와 UFS의 SRAM이 상호 교환적으로 관리되지 않아 주어진 자원을 낭비하는 문제를 발견하였다. 따라서, 낭비되는 자원을 최소화하며 두 계층의 특성을 고려한 통합 주소 변환 테이블 관리기법을 제안한다. 모바일 기기 트레이스를 기반으로 실험한 결과, 기존 관리기법 대비 캐시 적중률은 5% 향상하였고, 낭비되었던 공간 자원을 95% 감소하였으며 주소 변환 테이블 업데이트로 발생되는 가비지 컬렉션 횟수가 43% 감소하였다.

1. 서 론

모바일 기기를 사용하는 사용자의 데이터 및 응용들의 사이즈가 점차 커지며, 탑재되는 플래시 기반 저장장치인 UFS의 용량 또한 TB 이상으로 커지는 추세이다. UFS의 사이즈가 커짐에 따라 플래시 기반 저장장치 관리에 필수적으로 요구되는 메타데이터의 사이즈 또한 그에 비례하여 커지게 된다. 플래시 기반 저장장치는 물리적 특성상 덮어쓰기가 허용되지 않는다. 이미 쓰인 데이터의 수정이 필요한 경우 기존에 작성한 물리 페이지를 무효화하고 쓰기 가능한 상태의 새로운 물리 페이지에 쓰기 요청을 처리해야 한다. 호스트가 전달한 *논리 주솟값*의 실제 저장된 물리 페이지의 위치를 나타내는 *물리 주솟값*을 항상 기록해야 하고 이를 주소 변환 테이블이라 칭한다. 테이블의 사이즈는 저장장치 용량의 0.1% 이며 플래시에 저장되어 있다. 예를 들어, UFS 용량이 1TB일 경우 전체 테이블의 사이즈는 1GB이다.

주소 변환 테이블의 사이즈가 커질수록, 더 많은 저장장치 내부 메모리 자원이 요구된다. 모든 요청들을 처리하기 위해 플래시 변환 계층은 주소 변환 테이블 참조는 필수이다. 빠른 응답시간 안에 호스트 요청을 처리하기 위해, 테이블을 저장장치에 내장된 메모리에 적재하여 테이블을 빠르게 참조한다.

하지만, 모바일 기기는 하드웨어 및 가격적인 제약이 많아 요구량에 맞추어 UFS 내장 메모리 용량을 확장하기에는 많은 어려움이 있다. UFS에는 대략 1—2MB 정도의 SRAM만 내장되어 있다. 플래시 변환 계층은 DFTL[1] 기법을 통해 필요한 테이블의 일부 엔트리만 SRAM에 적재한다. 전달된 요청의 주솟값이 SRAM에 적재되어 있지 않을 경우 플래시 메모리에서 해당 엔트리를 읽어와야 하며 이 경우 처리 시간이 길어진다.

주소 변환 테이블 참조로 인해 느려지는 응답시간을 개선하기 위해, 호스트 메모리의 일부 공간을 주소 변환 테이블 캐시로 사용하는 Host Performance Booster (HPB)가 소개되었다[5]. 그림 1은 일반적으로 DFTL을 사용한 경우와 HPB를 적용한 경우의 테이블 참조 과정을 나타낸다. DFTL의 경우, 플래시 변환 계층이 SRAM에 물리 주솟값이 적재되어 있는지 확인하고, 캐시 실패(Cache miss)일 경우 전체 테이블이 저장되어 있는 플래시에서 물리 주솟값을 읽기 요청을 통해 SRAM에 적재한다. 반면, HPB의 경우 호스트 메모리에 요청한 물리 주솟값이 적재되어 있으면, 해당 물리 주솟값을 읽기 요청과 함께 전달하여 물리 주솟값을 위한 추가적인 플래시 읽기 수행 없이 빠르게 읽기 요청을 처리할 수 있다.

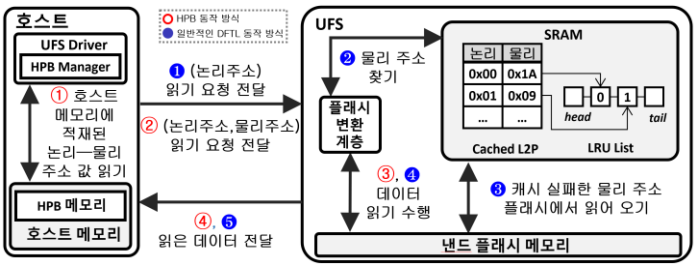


그림 1. 주소 변환 테이블 참조 과정

* 이 논문은 삼성전자의 지원(IO201207-07809-01)을 받아 수행된 결과임. (교신저자: 김지홍)

본 논문에서는 두 캐시 계층이 서로의 특성을 고려하지 않고 통합적으로 관리되지 못해 주어진 자원이 낭비되는 문제 현상을 처음으로 확인하였다. 이를 해결하고자, 두 독립적 캐시 계층인 호스트 메모리와 저장장치 SRAM에 적재된 주소 변환 테이블 엔트리들을 통합적으로 관리하여 주어진 자원을 최대한 효율적으로 활용할 수 있는 기법을 제안한다. 실제 모바일 기기에서 추출한 트레이스를 기반으로 실험한 결과 기존 관리기법 대비 캐시 적중률이 5% 향상되었고 낭비되었던 공간 자원을 95% 감소하며 주소 변환 테이블 업데이트로 발생되는 가비지 컬렉션 오버헤드가 43% 감소됨을 확인하였다.

본 논문의 구성은 다음과 같다. 2장에서는 기존의 HPB 관리기법과 UFS의 DFTL이 통합적으로 관리되지 못해 나타나는 문제를 분석한다. 이러한 문제점을 해결하기 위해 3장에서는 두 캐시 계층의 특성에 따라 동작하는 통합 관리기법을 설명한다. 4장에서는 제안한 기법을 시뮬레이션 환경에서 성능 및 효과를 평가하고 5장에서 결론을 맺는다.

2. 기존 HPB 관리기법의 공간 활용 분석

기존 HPB 관리기법을 적용한 시스템이 얼마나 잘 동작하는지 이해하기 위해, UFS의 SRAM만 활용하는 단일 캐시 시스템과 HPB가 결합된 통합 캐시 시스템 환경을 시뮬레이터에 구현하고 실제 모바일 기기에서 추출한 트레이스를 기반으로 캐시 적중률을 평가하였다. 사용한 UFS의 용량은 128GB이고, SRAM 사이즈는 기본값으로 512KB로 설정하였다. 기존 관리기법이 통합적으로 잘 동작하는지 평가하기 위해, 두 시스템의 전체 캐시 사이즈를 동일하게 설정하였다. 따라서, 물리적으로는 불가능하지만 단일 캐시 시스템의 SRAM 사이즈는 비교하는 통합 캐시 시스템이 사용하는 캐시 사이즈와 동일하게 설정하였다. 그림 2는 실험 결과로, x축은 전체 주소 변환 테이블의 해당 비율만큼 시스템 캐시 사이즈 설정값을 나타내고, y축은 각 케이스의 캐시 적중률이다.

그림 2를 통해, 통합 캐시 시스템의 캐시 적중률은 단일 캐시 시스템보다 최대 12% 낮은 것을 확인할 수 있다. 즉, 통합 캐시 시스템은 늘어나는 캐시 공간을 효율적으로 활용하고 있지 못함을 파악할 수 있다. 이러한 현상은 두 캐시 계층의 관리기법 분석을 통해 각 계층을 통합적으로 관리하는 기법의 부재 때문임을 파악하였다. 다시 말해, HPB 관리기법과[4] DFTL은 서로 다른 계층을 전혀 고려하지 않으며 동작한다.

HPB는 많은 읽기 요청이 있는 논리 주솟값에 대한 엔트리들을 선택적으로 호스트 메모리에 적재한다. 호스트 메모리에 적재되어 있는 논리 주솟값에 대한 쓰기 요청이 오게 되면, 해당 엔트리는 무효화된다. 이는, 덮어쓰기가 불가능한 저장장치의 특성상 기존 물리 주솟값은 무효화되고 새로운 물리 주솟값으로 주소 변환 엔트리가 변경되어, HPB에 적재된 기존 엔트리는 더 이상 유효하지 않기 때문이다. 반면, UFS의 SRAM에는 전달된 모든 요청의 논리 주소의 물리 주솟값들을 적재한다. 쓰기 요청이 오면 새로운 물리 페이지의 주솟값으로 업데이트되어 SRAM에 항상 적재되고 더티 (플래시에 저장된 주솟값과 SRAM에 적재된 값이 다른 경우) 플래그가 표시되며 최대한 나중에 쫓겨나도록 관리된다. 그 이유는, 더티 엔트리가 쫓겨나면 업데이트된 물리 주솟값으로 플래시에 저장된 주소 변환 테이블에 플래시 쓰기 요청을 통해 수정해야 하기 때문에 SRAM에 오래 유지하는 것이 비용적으로 효율적이다. 따라서, 서로 다른 특성을 가지는 두 캐시 계층은 각 특성에 맞게 관리되어 주어진 공간을 최대한 활용하여 높은 캐시 적중률을 제공할 수 있도록 동작해야 한다.

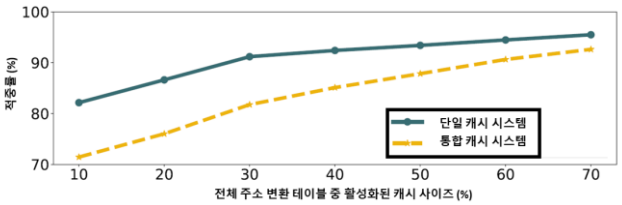


그림 2. 통합 캐시 시스템의 공간 효율 분석

3. 두 계층 통합적 주소 변환 테이블 관리기법

본 장에서는, 두 캐시 계층의 특성을 고려한 통합 관리기법을 제시한다. 본 관리기법을 통하여 두 독립적인 캐시 계층에 적재되어 있는 엔트리들의 공존 현상을 최소화하여 주어진 공간 자원을 최대한으로 활용 가능하도록 한다.

3.1 HPB의 호스트 메모리 캐시 관리기법

HPB를 활용하였을 때 가장 효과적인 경우는 호스트 메모리에 적재되어 있는 엔트리들에 대해 읽기 요청이 오는 경우이다. 반면에, 적재되어 있는 엔트리에 쓰기 요청이 수행되어 해당 값이 무효화된 경우 새로 업데이트된 물리 주솟값으로 다시 읽어오기 전까지는 시스템 성능에 도움이 되지 않으며 공간적인 낭비이다. 따라서, HPB 공간이 쓰기에 의해 업데이트되어 유효하지 않은 엔트리들을 적재하는데 낭비되는 현상을 방지해야 한다.

만약 무효화된 엔트리의 업데이트된 값을 UFS에 요청하여 읽어오면, 추후 읽기 요청에 의해 필요한 경우 효과적일 수 있지만 이는 동기화 오버헤드를 유발할 수 있다. 그 이유는 UFS에서 최신 항목을 가져오더라도 많은 항목들은 다시 쓰기 요청이 수행되어 또 무효화될 가능성이 높기 때문이다. 업데이트된 물리 주솟값을 UFS에서 HPB로 가져오더라도 두 계층 모두 같은 엔트리를 보유하게 되어 공간적으로 비효율적이다.

이러한 현상을 방지하기 위해 쓰기 요청에 의해 새로운 물리 주소로 업데이트되는 엔트리는 호스트 메모리에서 즉시 제거하는 기법을 제안한다. 모바일 시스템의 호스트 메모리는 응용들이 사용하기에도 여전히 충분하지 않은 자원으로 제안하는 기법을 통해 중복되는 엔트리로 낭비되는 메모리 자원을 최소화할 수 있다. 또한, 최신 항목을 가져오기 위한 동기화 오버헤드를 방지할 수 있다. 그림 3은 제안하는 기법의 동작 방식을 보여주며, 쓰기 요청이 오는 논리 주솟값은 SRAM에 새로 업데이트된 물리 주솟값으로 더티 플래그와 함께 적재되기 때문에 HPB에서 바로 쫓아내어 공간을 확보한다.

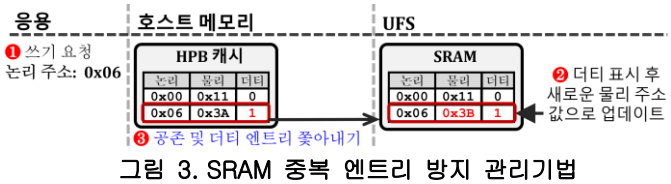


그림 3. SRAM 중복 엔트리 방지 관리기법

3.2 UFS 내장 SRAM 캐시 관리기법

HPB 관리기법에 의해 호스트 메모리로 적재되는 엔트리들은 무조건 최근에 UFS에 전달된 읽기 요청들에 대한 엔트리이다. 따라서, 해당 엔트리들은 항상 UFS SRAM에 적재되어 있다. 이러한 엔트리들은 HPB 관리기법에 의해 호스트 메모리에서 쫓겨나지 않는 이상, HPB의 호스트 메모리 캐시에서 항상 적중되며 더 이상 SRAM에서 참조되지 않는다. 게다가, SRAM의 사이즈는 매우 한정적이기 때문에 두 계층이 중복되는 엔트리를 보유하는 것은 공간적으로 매우 낭비다. 이 문제를 해결하기 위해, HPB로 엔트리들을 적재할 때 SRAM 캐시에서 해당 엔트리들을 즉시 쫓아내는 기법을 제안한다. 그림 4와 같이, HPB로 적재되는 엔트리들은 SRAM에서 바로 쫓아내어 공간 낭비를 최소화한다.

제안하는 두 관리기법을 결합하면 자연스럽게 많은 더티 엔트리들은 SRAM에 남게 되고 클린 (플래시에 저장된 물리 주솟값과 캐시에 적재된 엔트리의 값이 같은 경우) 엔트리들은 HPB에 대부분 남게 된다. 또한, 각 계층에 적재되는 하나의 엔트리는 1024개의 연속적인 논리 주솟값들로 이루어져 있어, 본 기법을 통해 더티 엔트리들이 쫓겨날 때 동일한 페이지로 한 번에 제거될 가능성이 높아져 테이블 업데이트로 인한 플래시 쓰기 동작 횟수가 크게 감소하게 된다.

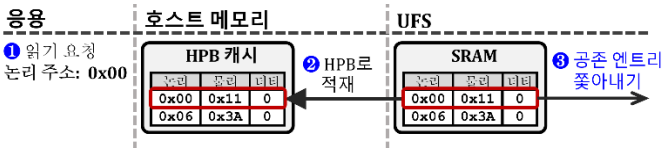


그림 4. HPB 중복 엔트리 방지 관리기법

4. 실험

4.1 실험 환경

제안한 기법들의 효과를 증명하기 위해 UFS 내부 SRAM 관리기법을 수정하여 평가해야 한다. 하지만, 실제 UFS를 수정하는 것은 불가능하기 때문에 FlashDriver[3] 시뮬레이터를 수정하고 HPB 계층까지 동작할 수 있도록 확장하여 평가하였다. 평가에 사용한 워크로드로는 pixel5a 모바일 기기에서 응용 9개를 사용자 패턴을 고려해 실행하며 blktrace[2]를 통해 수집한 I/O 트레이스를 사용했다. HPB 사이즈는 전체 주소 변환 테이블의 50%를 적재할 수 있는 사이즈로 설정하였다.

4.2 기법 평가

그림 5에서는 제안하는 두 캐시 계층의 통합 관리기법을 적용했을 때와 기존 관리기법을 적용했을 때의 적중률과 두 계층 사이에 공존하는 엔트리 비율 그리고 가비지 컬렉션 효과를 보여주고 있다. 그림 5(a)에서는 2장에서 비교한 단일 캐시 시스템과 통합 캐시 시스템의 적중률을 비교하며, 본 논문에서 제안하는 통합 관리기법을 기존 관리기법과 비교한다. 제안하는 통합 관리기법을 적용하였을 때, 기존 통합 캐시 시스템 대비 적중률을 향상시키며, 단일 캐시 시스템의 적중률에 더 가까워졌다. 성능 향상의 이유는 그림 5(b)를 통해 확인할 수 있다. 그림 5(b)는 전체 호스트 읽기 및 쓰기 요청 중 HPB의 호스트 메모리와 UFS의 SRAM에 엔트리가 공존했던 경우를 나타낸다. 기존 20.1%의 엔트리가 공존하였던 경우가 제안하는 통합 관리기법을 통하여 0.9%만 공존한다.

해당 결과를 통해 각 계층별 특성에 맞는 엔트리들이 서로 공존하지 않게 됨을 알 수 있다. 특히, SRAM에는 대부분 더티 엔트리들만 위치하고, 공간이 부족하여 클린 엔트리들을 적재하기 위해 더티 엔트리들이 쫓아내는 경우가 줄어들게 된다. 이를 통해, 그림 5(c)에서 확인할 수 있듯 제안하는 통합 관리기법을 적용하여 가비지 컬렉션 횟수가 기존 대비 43% 감소했다. 다시 말해, 제안하는 기법을 통해 업데이트된 주소 변환 테이블을 플래시에 저장하기 위한 쓰기 및 지우기 트래픽이 줄어들어 전체적인 저장장치 수명이 또한 늘어난다.

5. 결론

본 논문에서는 기 제안된 HPB를 시스템에 적용했을 때, UFS SRAM 캐시와 통합적으로 동작하지 못해 캐시 자원이 공간적으로 낭비되며 주어진 자원이 최대한 활용되지 못하고 있는 문제점을 처음으로 지적하였다. 이러한 현상을 개선하기 위해, 각 계층의 특성을 분석하였고 이를 고려한 두 캐시 계층의 통합 관리기법을 제시하였다. 이를 활용하여 불필요한 호스트 메모리 공간 사용을 최소화하고 주어진 캐시 공간 자원을 최대한으로 활용함을 확인하였다.

참 고 문 헌

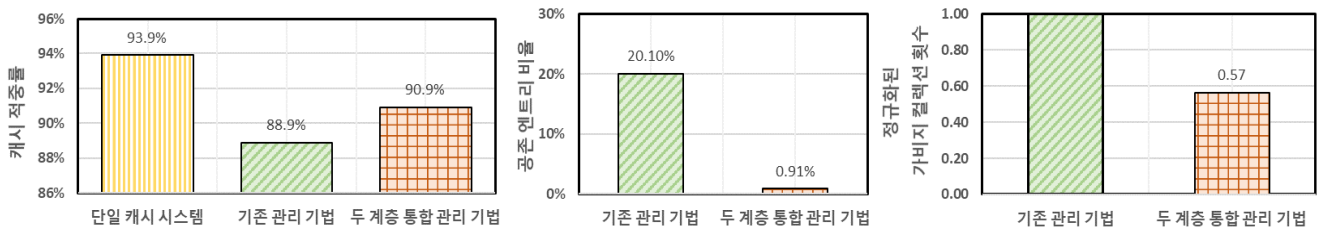
[1] Aayush Gupta et al, “DFTL: A flash translation layer employing demand-based selective caching of page-level address mappings”, ASPLOS (2009).

[2] blktrace, http://linux.die.net/man/8/blktrace.

[3] FlashDriver, https://github.com/dgist-datalab/FlashDriver.

[4] HPB Android kernel code. https://android.googlesource.com/kernel/common/+/refs/heads/android12-5.10/drivers/scsi/ufs/ufshpb.c.

[5] Woo-Ram Jeong et al, “Improving flash storage performance by caching address mapping table in host memory”, HotStorage (2017).



(a) 시스템 및 관리기법 별 캐시 적중률 (b) 두 계층간 공존하는 엔트리 비율 (c) 가비지 컬렉션 횟수

그림 5. 두 계층 통합 관리기법 적용을 통한 성능 개선 분석