The Journal of The Institute of Internet, Broadcasting and Communication (IIBC)
Vol. 15, No. 6, pp.129-134, Dec. 31, 2015. pISSN 2289-0238, eISSN 2289-0246

http://dx.doi.org/10.7236/JIIBC.2015.15.6.129
JIIBC 2015-6-18

1회성 쓰기 참조 특성을 고려하는 스마트폰 버퍼캐쉬 관리 기법

Buffer Cache Management of Smartphones Exploiting Write-Only-Once Characteristics

김도희*, 반효경**

Dohee Kim*, Hyokyung Bahn**

요 약 본 논문에서는 스마트폰 앱의 파일 접근 중 상당 부분이 1회성 쓰기 참조라는 것을 분석하고 이러한 특성을 고려한 버퍼캐쉬 관리 기법을 제안한다. 버퍼캐쉬는 여러 번 참조되는 파일 데이터를 스토리지 접근 없이 메모리에서 처리하여 성능 개선을 도모하지만, 쓰기 참조가 발생한 경우에는 짧은 시간 내에 스토리지에 직접 반영하여 시스템 크래쉬에 대비한다. 제안하는 기법은 1회의 쓰기만 발생한 캐쉬 데이터의 경우 스토리지 반영 직후 버퍼캐쉬에서 쫓아내어 캐쉬의 공간 효율을 높인다. 다양한 스마트폰 앱에 적용한 시뮬레이션 실험 결과 제안하는 기법은 기존 버퍼캐쉬 관리 기법에 비해 캐쉬적중률을 5-33% 향상시키고, 전력소모를 27-92% 줄일 수 있음을 보인다.

Abstract This paper analyzes file access characteristics of smartphone apps and finds that a large portion of file writes are performed only once. Based on this observation, we present a new buffer cache management scheme that considers this characteristics. Buffer cache improves storage performance by maintaining hot file data in memory thereby servicing subsequent requests without storage accesses. However, it should flush modified data to storage in order to resist system crashes. The proposed scheme evicts cache data that has been written only once upon flushes, thus improving cache space utilization. Simulation experiments show that the proposed scheme improves cache hit ratio by 5-33% and power consumption by 27-92%.

Key Words : Buffer cache, write reference, power consumption, smartphone.

Ⅰ. 서 론

버퍼캐쉬는 대용량 서버에서 모바일 디바이스에 이르는 다양한 컴퓨터 시스템에서 느린 스토리지의 접근 속도를 완충하기 위해 널리 사용된다[1]. 버퍼캐쉬는 최근에 사용된 파일 데이터를 메모리에 보관하여 동일 데이터에 대한 재요청시 스토리지 접근 없이 해당 데이터를 직접 서비스하여 성능 향상을 도모한다.

한편, 스마트폰에서는 갑작스런 물리적 충격 혹은 전원 오류 시 파일시스템의 일관성이 깨어지는 문제가 발생할 수 있다. 이를 방지하기 위해 저널링(혹은 로깅)을 사용하여 파일 데이터의 수정본을 관리한다[2]. 저널링은 파일 데이터를 수정하기 전에 복사본을 만들어 둠으로써 수정 도중에 크래쉬가 발생하더라도 직전 버전으로의 복원을 가능하게 한다. 이 과정에서 재사용되지 않을 데이터가 버퍼캐쉬에 추가로 생성되는 비효율성이 발생한다. 예를 들어 스마트폰의 버퍼캐쉬에서 파일 데이터 A_{data}가 수정될 경우 복사본 파일 $A_{journal}$을 먼저 생성한 후 A_{data}

*준회원, 이화여자대학교 컴퓨터공학과
**정회원, 이화여자대학교 컴퓨터공학과(교신저자)
접수일자: 2015년 10월 11일, 수정완료: 2015년 11월 11일
게재확정일자: 2015년 12월 11일

Received: 11 October, 2015 / Revised: 11 November, 2015 / Accepted: 11 December, 2015
**Corresponding Author: bahn@ewha.ac.kr
Dept. of Computer Engineering, Ewha University, Korea

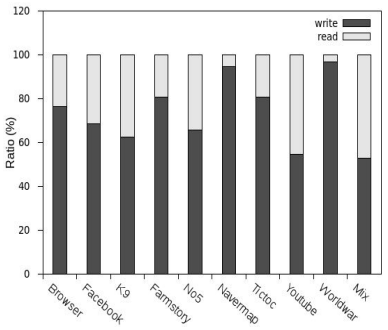

그림 1. 스마트폰 앱별 파일 읽기/쓰기 연산 비율
Fig. 1. File read/write ratio for smartphone apps.

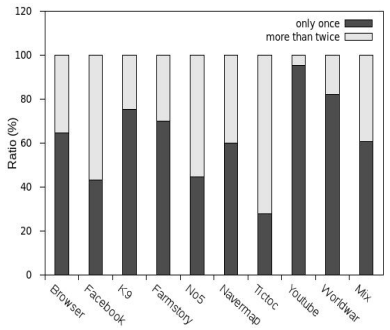

그림 2. 1번의 쓰기만 발생한 블록 비율
Fig. 2. Ratio of blocks written only once.

를 수정한다. 이렇게 되면 $A_{journal}$은 낡은 데이터이고 최신 데이터는 A_{data}이므로 향후 모든 읽기 쓰기 요청은 A_{data}에 대해 발생한다. 한편, 버퍼캐쉬에 존재하던 $A_{journal}$과 A_{data}는 스토리지 반영(flush) 연산에 의해 스토리지에 기록되어 갑작스런 전원 오류에 대비한다. 스토리지 기록 후 최신본 A_{data}는 계속 사용될 수 있으나 낡은 데이터인 $A_{journal}$은 사용 가능성이 없으므로 버퍼캐쉬에 남아 있는 동안 캐쉬 공간이 낭비된다.

본 논문은 스마트폰 앱의 파일참조 분석을 통해 상당 부분의 스마트폰 파일 입출력 요청이 읽기보다 쓰기 위주라는 것과 이들 쓰기 중 상당 부분은 1회성 쓰기 참조라는 것을 보인다. 이러한 쓰기 참조는 SQLite와 같은 스마트폰용 데이터베이스가 저널링을 하는 과정에서 발생하는 것으로 이들이 한정된 버퍼캐쉬 공간을 낭비하는 주요 요인이 된다. 이에 본 논문에서는 1번만 쓰기 참조가 이루어진 파일 데이터의 경우 스토리지 반영 시점에 버퍼캐쉬에서 쫓아내어 공간 효율성을 높이는 방안을 제안한다. 다양한 스마트폰 앱의 파일 참조 트레이스를 이용한 시뮬레이션을 통해 제안한 기법이 기존 버퍼캐쉬 대비 성능을 5-33% 향상시키고 전력 소모를 27-92% 줄임을 보인다.

II. 본 문

1. 스마트폰 앱의 파일 참조 분석

본 논문에서는 대표적인 스마트폰 앱들의 파일 참조 트레이스를 추출하여 읽기 쓰기 참조의 비율 및 특성을 분석하였다. 그림 1은 전체 파일 접근 중 읽기와 쓰기의 비율을 보여주고 있다. 그림에서 보는 것처럼 파일 데이터 접근의 상당 부분이 쓰기 참조인 것을 확인할 수 있다. 쓰기의 비율이 가장 낮은 유튜브의 경우에도 54%를 차지했으며 일부 앱은 쓰기 참조가 97%에 이르렀다.

그림 2는 쓰기 참조가 발생한 데이터 중 1번의 쓰기만 발생한 비율을 보여주고 있다. 그림에서 보는 것처럼 1회성 쓰기 참조가 발생한 데이터의 비율은 최대 95%에 이를 정도로 큰 비중을 차지하는 것을 확인할 수 있다. 이러한 데이터는 캐쉬에 두더라도 성능 향상에 도움이 되지 않으므로 이러한 불필요한 데이터에 의해 캐쉬 공간이 낭비되고 있음을 알 수 있다.

2. 1회성 쓰기 참조를 고려한 버퍼캐쉬 관리 기법

캐쉬에서는 파일 데이터를 블록 단위로 관리하며, 캐쉬에 진입한 후 수정되지 않은 블록은 클린 블록(clean block), 수정된 블록은 더티 블록(dirty block)으로 구분한다. 더티 블록은 프로그램이 명시적으로 동기화(sync) 요청을 할 때, 혹은 동기화 요청이 없더라도 시스템에서 5초 주기로 스토리지에 반영(flush)하며, 반영 후 클린 상태로 바뀌게 된다. 예를 들어, 그림 3에서 블록 A는 0초 지점에 쓰기 참조가 발생하여 더티 블록이 되었고, 3초 지점에는 읽기 참조가 발생하였다. 5초 시점에 주기적인 스토리지 반영 연산이 수행되어 수정본이 스토리지에 기록되며 이후 블록 A는 클린 블록이 된다. 블록 B의 경우 7초와 9초 지점에 각각 쓰기 참조가 발생했는데, 10초 지점에 스토리지 반영시 마지막으로 쓰기가 발생한 9초 지점의 내용이 스토리지에 기록된다.

앞 절에서 스마트폰 버퍼캐쉬에서의 쓰기 참조 분석을 통해 1회성 쓰기 참조의 비율이 높음을 확인하였다. 따라서 제안하는 기법은 스토리지 반영 연산이 발생할

때, 1회성 쓰기 참조만 발생한 블록을 버퍼캐쉬에서 조기 방출함으로써 캐쉬 공간의 효율을 높인다. 이때, 쓰기 참조가 1번만 발생했더라도 읽기 참조가 발생한 블록은 캐쉬에서 쫓아내지 않는다. 이는 읽기 참조가 발생한 블록의 경우 짧은 시간 안에 재사용 가능성이 높은 특징을 반영하여 캐쉬의 효과를 높이기 위함이다[1]. 그림 4는 그림 3과 동일한 읽기 쓰기 요청에 대해 제안하는 버퍼캐쉬가 발생시키는 입출력의 예를 보여주고 있다. 5초 지점에서 스토리지 반영(flush) 연산 발생 시 블록 A는 쓰기 횟수가 1번이지만 3초 지점에서 읽기 요청이 발생하였으므로 스토리지에 기록할 뿐 캐쉬에서 조기 방출하지 않는다. 블록 B의 경우에는 5초 지점까지 쓰기가 1번만 발생하고 읽기 요청이 없었으므로, 5초 지점에 블록 B의 수정 내용을 스토리지로 기록한 후 캐쉬에서 방출한다. 이와 같이 블록이 조기 방출 대상인지 아닌지를 구별하기 위해 제안하는 기법은 블록마다 방출플래그(eviction flag)를 두어 블록의 쓰기 횟수가 1번이고 읽기 참조가 발생하지 않았는지 여부를 표시한다. 그림에서 방출플래그가 'o'인 경우 해당 블록은 스토리지 반영 시점에 캐쉬에서 삭제할 대상이고 'x'인 경우 그렇지 않음을 표시한다. 한편, 쓰기 횟수가 1회여서 삭제된 블록이 다시 캐쉬에 진입한 경우 그 쓰기 횟수를 0부터 재계산할 경우 해당 블록이 캐쉬에서 삭제 후 재진입되는 일이 반복될 수 있다. 따라서, 캐쉬에서 데이터가 삭제되더라도 해당 블록의 쓰기 발생 여부는 히스토리 버퍼(history buffer)를 두어 한동안 저장하도록 한다. 예를 들어 블록 B는 5초 지점에서 읽기 참조가 없고 쓰기 횟수만 1회이므로 방출플래그가

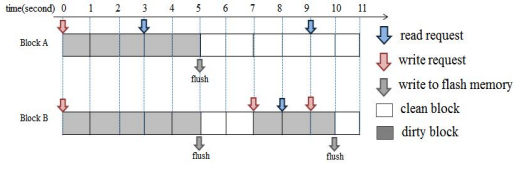

그림 3. 기존 버퍼캐쉬에서 입출력 발생의 예
Fig. 3. I/O situations on existing buffer cache.

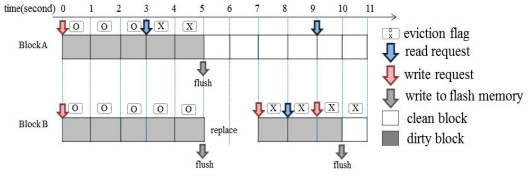

그림 4. 제안하는 기법에서 입출력 발생의 예
Fig. 4. I/O situations on proposed buffer cache.

'o'으로 셋팅돼 있어 스토리지 반영 후 캐쉬에서 삭제되지만, 7초 지점에 다시 쓰기 참조가 발생하여 캐쉬에 진입할 때는 히스토리 버퍼에 블록 B가 존재하므로(즉, 최초의 쓰기가 아니므로) 방출플래그는 'x'로 바뀌게 된다.

III. 성능 평가

본 장에서는 제안한 기법의 성능을 평가하기 위해 수행한 시뮬레이션 실험 결과를 기술한다. 버퍼캐쉬에서 데이터를 관리하는 단위인 블록 크기는 대부분의 운영체제에서 사용하는 4KB로 설정하였다. 실험에 사용된 DRAM 및 플래시메모리의 성능과 소모전력은 표 1과 같다[4,5]. DRAM의 읽기/쓰기 접근 시간은 50ns이고 읽기/쓰기 시 소모되는 전력량은 비트 당 0.1nJ이다. 한편, DRAM의 경우 읽기/쓰기가 없는 휴지상태에도 지속적인 전력재공급(refresh) 연산을 통해 정적전력(static power)이 GB당 1W가 소모되며 이를 실험에 반영하였다. 플래시메모리의 읽기/쓰기 접근 시간은 4KB 페이지 당 각각 284.2μs와 1833μs이고, 전력소모는 9.5μJ과 76.1μ

표 1. 접근 시간과 전력소모
Table 1. Access time and power consumption.

매체	연산	접근 시간	전력(에너지)소모
DRAM	read	50(ns)	0.1(nJ/bit)
	write	50(ns)	0.1(nJ/bit)
	refresh	-	1(W/GB)
플래시 메모리	read	284.2(μs/4KB)	9.5(μJ/4KB)
	write	1833(μs/4KB)	76.1(μJ/4KB)

표 2. 실험에 사용된 트레이스의 특성
Table 2. Characteristics of traces used in experiments.

Workload	Total # of references	Ratio of read/write
Browser	26,221	1:3.38
Facebook	25,962	1:2.15
Farmstory	34,360	1:4.17
No5	155,299	1:1.93
Navermap	407,857	1:18.03
Tictoc	31,906	1:5.87
Worldwar	12,320	1:44.15
Mix1	628,739	1:6.05
Mix2	438,410	1:1.23

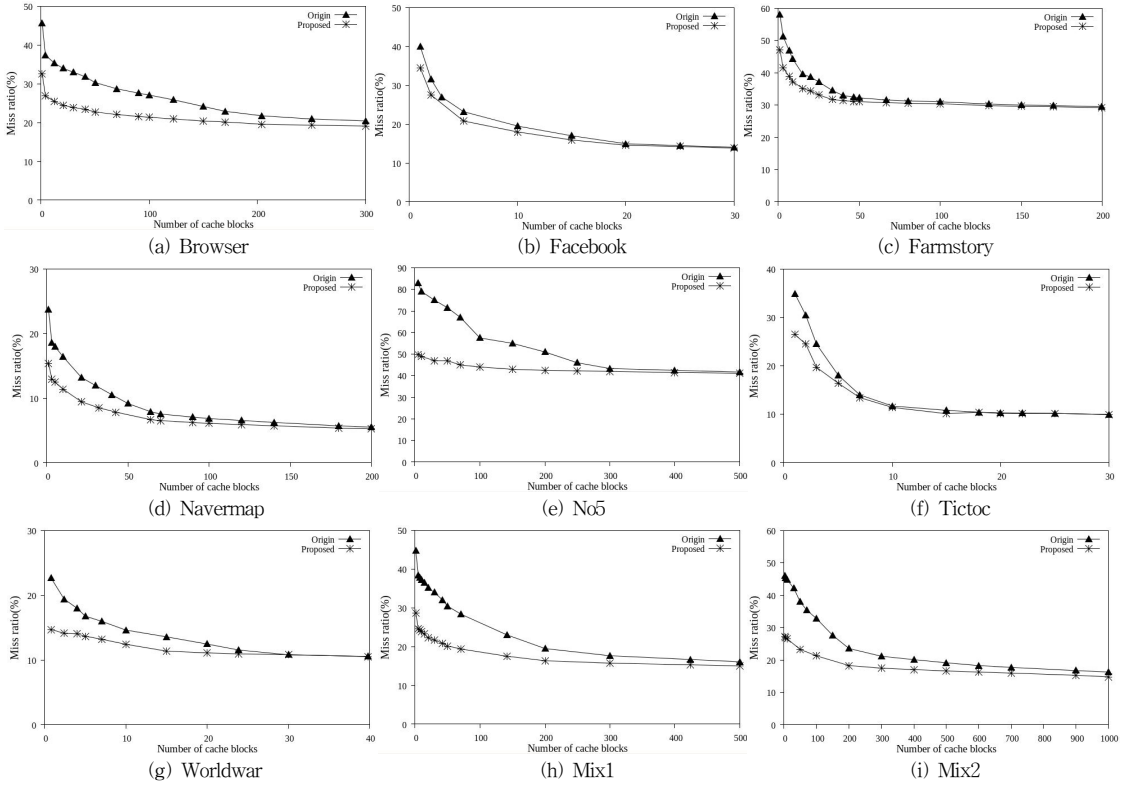

그림 5. 캐쉬 크기에 따른 버퍼 캐시 접근 실패율
Fig. 5. Cache miss ratio as a function of the cache size.

J이다.

실험에 사용된 트레이스는 스마트폰에서 사용되는 Browser, Facebook, Farmstory, No5, Navermap, Tictoc, Worldwar 등 7가지 앱 실행시의 파일 참조 트레이스와 여러 앱을 섞어서 수행한 경우의 파일 참조 트레이스인 Mix1, Mix2로 구성된다. 트레이스는 각 앱을 15-20분간 실행하면서 발생한 파일 참조 기록을 추출하였다. 실제 환경에서 스마트폰 사용자가 여러 앱을 사용하는 것과 유사한 조건에서 제안한 기법의 성능을 평가하기 위해 Mix1은 Facebook과 Navermap, No5를 함께 수행하였고, Mix2는 Browser와 Tictoc, No5, Facebook을 함께 수행하였다. 표 2는 추출된 트레이스의 특성을 정리한 것이다.

그림 5는 캐쉬 크기의 변화에 따라 기존 캐쉬 관리기법과 제안한 기법의 캐쉬 부재율(miss ratio)을 보여주고 있다. 기존 기법은 LRU(least recently used) 캐쉬 교체 알고리즘과 5초 단위의 스토리지 반영 연산(flush)을 사용하며, 제안한 기법은 반영 연산 시점에 1회의 쓰기만 발생한 블록을 캐쉬에서 조기 방출한다. 그림에서 보는 것처럼 모든 경우에서 제안한 기법이 기존 기법보다 성능이 향상되었으며, 성능 향상폭은 6-33%에 이르렀다. 특히, 제안한 기법의 성능 개선은 캐쉬 크기가 작을수록 더 큰 것을 확인할 수 있었다.

제안한 기법의 캐쉬 공간 확보 효과가 어느 정도인지를 알아보기 위해 동일한 성능을 나타내는 데에 필요한 캐쉬 공간이 기존 방식과 제안한 방식에서 어떻게 다른지 비교해 보았다. 그림 6은 Mix1 워크로드에서 동일한 성능을 나타내는 두 지점인 A와 B에서 기존 기법과 제안한 기법에 필요한 캐쉬 크기를 보여준다. 그림에서 보는 것처럼 제안하는 기법은 기존 기법에 비해 동일한 성능을 나타내기 위해 1/4 이하의 캐쉬 공간이 필요한 것을 확인할 수 있다. 그림 7-8은 동일한 캐쉬 성능을 나타내는 지점에 대해 기존 기법과 제안한 기법의 총 수행시간과 전력소모를 비교한 것이다. 제안한 기법은 기존 기법

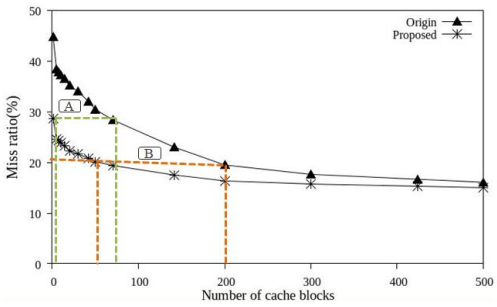

그림 6. 기존 기법과 제안하는 기법에서 동일한 성능을 나타내는 캐쉬 크기 비교
Fig. 6. Cache sizes for the same performance on existing and proposed caches.

에 비해 상대적으로 더 작은 캐쉬 크기를 가지기 때문에 성능은 더 나쁠 수 있으나 DRAM에 의한 전력 소모가 더 적은 것을 기대할 수 있다. 그러나, 그림 7에서 보는 것처럼 제안한 기법이 비록 1/4 이하의 캐쉬 크기를 가졌음에도 기존 기법에 비해 성능이 거의 저하되지 않는 것을 확인할 수 있다. 이는 성능 개선 효과가 없는 1회성 쓰기 참조 블록을 캐쉬에서 빨리 삭제해서 추가적인 공간 확보를 함으로써 얻게 된 결과이다. 또한 그림 8에서 보는 것처럼 제안한 기법은 기존 기법에 비해 에너지 소모가 크게 줄어든 것을 확인할 수 있다. 기존 기법 대비 에너지 절감률은 27-92%에 이르는 것으로 확인되었다. 이는 스마트폰 전력 소모의 상당 부분을 DRAM 메모리가 차지하고 있다는 기존 연구와도 일관성이 있다[3,6].

스마트폰은 대부분의 시간을 휴지 상태(idle state)로 있지만, 사용자가 스마트폰 사용을 원할 때는 언제든지 재개(resume)할 수 있도록 항상 대기 상태에 있어야 한다[3,7,8]. 이를 위해 DRAM 메모리는 읽기 쓰기 연산이 없더라도 지속적인 전력재공급(refresh) 연산을 해주어야 하며 이는 막대한 배터리 소모의 원인이 되고 있다[3]. 제안한 기법은 DRAM으로 구성되는 버퍼캐쉬의 크기를 줄여 전체 시스템의 전력소모를 줄일 수 있음을 보인 것이다.

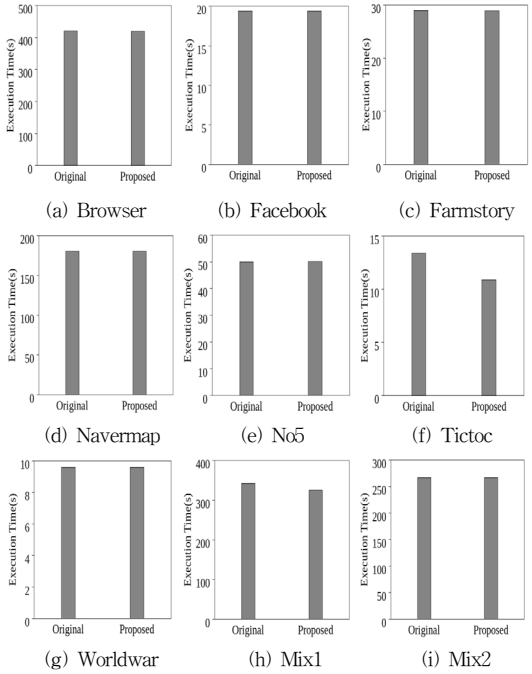

그림 7. 기존 기법과 캐쉬 크기가 1/4 이하인 환경에서의 제안한 기법 간 수행시간 비교
Fig. 7. Execution time of the proposed cache under the cache size of 1/4 that of existing cache.

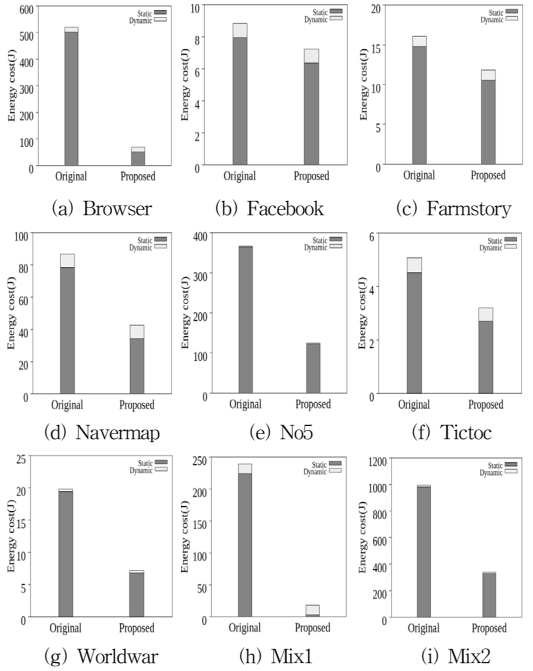

그림 8. 기존 기법과 캐쉬 크기가 1/4 이하인 환경에서의 제안한 기법 간 에너지 소모 비교
Fig. 8. Energy consumption of the proposed cache under the cache size of 1/4 that of existing cache.

Ⅳ. 결 론

본 논문에서는 스마트폰의 파일 참조 중 상당 부분이 1회성 쓰기 참조라는 것을 분석하고 이러한 특성을 반영하는 새로운 버퍼캐쉬 관리 기법을 제안하였다. 제안한 기법은 쓰기 참조만 1번 발생한 블록을 스토리지 반영 시점에 캐쉬에서 조기 방출함으로서 캐쉬 공간의 불필요한 낭비를 막는다. 다양한 스마트폰 앱을 이용한 실험 결과 제안하는 기법은 스마트폰 성능을 5-33% 향상시킴을 확인하였고, 전력소모를 평균 64% 줄일 수 있음을 보였다.

References

[1] E. Lee, H. Bahn, "Caching Strategies for High Performance Storage Media," ACM Trans. Storage, Vol. 10, No. 3, 2014.

[2] SQLite, https://www.sqlite.org

[3] S. Liu, K. Pattabiraman, T. Moscibroda, B. Zorn, "Flikker: Saving DRAM Refresh power through Critical Data Partitioning," Proc. ACM ASPLOS, pp. 213-224, 2011.

[4] S. Lee, H. Bahn, Sam H. Noh, "CLOCK-DWF: A Write-History-Aware Page Replacement Algorithm for Hybrid PCM and DRAM Memory Architectures," IEEE Trans. Computers, Vol 63, No. 9, pp. 2187-2200, 2014.

[5] G. Sun, Y. Joo, Y. Chen, D. Niu, Y. Xie, Y. Chen, H. Li, "A Hybrid Solid-State Storage Architecture for the Performance, Energy Consumption, and Lifetime Improvement," Proc. IEEE HPCA, pp. 1-12, 2010.

[6] A. Carroll, G. Heiser, "An Analysis of Power Consumption in a Smartphone," Proc. USENIX ATC, pp.271-284, 2010.

[7] D. Kim, H. Bahn, "An Adaptive Location Detection Scheme for Energy-Efficiency of Smartphones," the Journal of the Institute of Internet, Broadcasting and Communication (JIIBC), Vol. 15, No. 3, pp. 119-124, 2015.

[8] B. Kim et al., "Design of a Smart Phone Panoramic Photograph Support System Using Sensor and Camera Technology," Journal of the Korea Academia-Industrial cooperation Society, Vol. 15, No. 12, pp. 7187-7192, 2014.

저자 소개

김 도 희(준회원)

- 2012년 2월 : 이화여자대학교 대학교 컴퓨터공학과 학사
- 2012년 3월 ~ : 이화여자대학교 컴퓨터공학과 통합과정

<주관심분야 : 운영체제, 스토리지시스템, 임베디드시스템>

반 효 경(정회원)

- 1997년 2월 : 서울대학교 계산통계학과 학사
- 1999년 2월 : 서울대학교 전산과학과 석사
- 2002년 2월 : 서울대학교 컴퓨터공학부 박사.
- 2002년 9월 ~ : 이화여자대학교 컴퓨터공학과 교수.

<주관심분야 : 운영체제, 스토리지시스템, 임베디드시스템>

※ 본 연구는 미래창조과학부의 재원으로 한국연구재단(No.2011-0028825)의 지원과 2014학년도 이화여자대학교 대학원 장학금 지원으로 수행된 연구결과임.