The Journal of The Institute of Internet, Broadcasting and Communication (IIBC)
Vol. 22, No. 1, pp.71-76, Feb. 28, 2022. pISSN 2289-0238, eISSN 2289-0246

https://doi.org/10.7236/JIIBC.2022.22.1.71
JIIBC 2022-1-11

영속 메모리를 이용한 스마트폰 버퍼 캐시의 선별적 플러시 정책

Policy for Selective Flushing of Smartphone Buffer Cache using Persistent Memory

임수정*, 반효경**

Soojung Lim*, Hyokyung Bahn**

요 약 버퍼 캐시는 스토리지의 느린 속도를 완충하는 중요한 역할을 하지만, 데이터의 유실을 막기 위한 주기적인 플러시 연산으로 인해 스마트폰에서 그 효과가 크게 떨어진다. 본 논문에서는 소량의 영속 메모리에 선택적인 플러시 정책을 적용하여 스마트폰 버퍼 캐시의 플러시 오버헤드를 크게 줄일 수 있음을 보인다. 이는 스마트폰 앱의 I/O 분석 결과 대부분의 파일 쓰기가 소량의 핫 데이터에 집중돼 있는 반면 상당 부분의 파일 데이터는 1회성 쓰기에 국한한다는 점에 근거한다. 제안하는 기법은 플러시 상황 발생 시 자주 수정되는 데이터를 영속 메모리로 우회 플러시하고 그렇지 않은 데이터만을 스토리지로 플러시한다. 이를 통해 스토리지 쓰기량을 크게 줄이는 동시에 영속 메모리의 공간 효율성을 높인다. 인기 있는 스마트폰 앱의 I/O 트레이스를 이용한 재현 실험을 통해 제안하는 기법이 스토리지 쓰기량을 평균 25.8%, 최대 37.8%까지 줄임을 보인다.

Abstract Buffer cache bridges the performance gap between memory and storage, but its effectiveness is limited due to periodic flush, performed to prevent data loss in smartphones. This paper shows that selective flushing technique with small persistent memory can reduce the flushing overhead of smartphone buffer cache significantly. This is due to our I/O analysis of smartphone applications in that a certain hot data account for most of file writes, while a large proportion of file data incurs single-writes. The proposed selective flushing policy performs flushing to persistent memory for frequently updated data, and storage flushing is performed only for single-write data. This eliminates storage write traffic and also improves the space efficiency of persistent memory. Simulations with popular smartphone application I/O traces show that the proposed policy reduces write traffic to storage by 24.8% on average and up to 37.8%.

Key Words : periodic flushing, file access, mobile application, smartphone, buffer cache, persistent memory

*준회원, 이화여자대학교 컴퓨터공학과
**정회원, 이화여자대학교 컴퓨터공학과(교신저자)
접수일자 2021년 11월 15일, 수정완료 2022년 1월 18일
게재확정일자 2022년 2월 4일

Received: 15 November, 2021 / Revised: 18 January, 2022 / Accepted: 4 February, 2022
*Corresponding Author: bahn@ewha.ac.kr
Dept. of Computer Engineering, Ewha University, Korea

Ⅰ. 서 론

최근 모바일 앱 기술의 발전으로 웹 브라우저, 이메일 등 전통적인 서비스뿐 아니라 소셜 미디어, 개인 방송, 위치 기반 서비스 등을 스마트폰을 통해 이용하는 빈도가 늘고 있다[1-5]. 스마트폰을 통해 실행되는 서비스의 수가 급격히 증가함에 따라 스마트폰의 성능 저하가 중요한 문제로 부상하고 있다. 스마트폰의 성능에 관한 연구에 따르면 스마트폰 시스템의 성능에 병목현상을 일으키는 가장 주요한 원인은 CPU 코어나 무선 네트워크가 아닌 스토리지인 것으로 분석되고 있다[6]. 느린 스토리지 접근 시간과 데이터의 용량 증가로 인해 I/O 작업이 모바일 시스템의 성능에 가장 큰 악영향을 미치고 있는 것이다[7]. 캐싱 기법은 이러한 성능 격차를 해소하기 위한 방법으로 버퍼 캐시, 웹 캐시 등 다양한 영역에서 널리 활용되고 있다[8, 9, 10].

한편, 기존의 버퍼 캐시는 휘발성 메모리인 DRAM 메모리를 캐시 매체로 사용하므로 캐시 내에서 수정된 데이터가 스토리지에 반영되기 전에 전원이 나갈 경우 데이터의 불일치 문제가 발생할 수 있다. 이러한 상황에 대처하기 위해 파일 시스템은 짧은 시간 간격으로 수정된 데이터를 스토리지에 반영하는 플러시 또는 저널링을 수행한다[10]. 이러한 주기적인 스토리지 쓰기 연산은 저장 데이터의 신뢰성을 향상시키지만 버퍼 캐시에 여유 공간이 있는 경우에도 빈번한 스토리지 접근을 유발해 버퍼 캐시의 효율을 저하시킨다[10]. 즉, 버퍼 캐시를 통해 스토리지의 읽기 연산은 크게 개선할 수 있는 반면, 쓰기 연산의 경우 캐시 용량을 아무리 늘이더라도 주기적인 플러시에 의해 그 효용이 제한적일 수밖에 없다.

본 논문에서는 이러한 쓰기 연산의 오버헤드를 줄이기 위해 버퍼 캐시에 소량의 영속 메모리를 추가하고 수정된 데이터를 휘발성 버퍼 캐시가 아닌 영속 메모리에 보관하여 빈번한 스토리지 접근을 막고자 한다. 영속 메모리는 전원이 차단되더라도 내용을 유지할 수 있으므로 주기적인 스토리지 플러시가 필요하지 않다. 비록 영속 메모리 기술이 빠르게 발전하고 있으나, 아직까지 단위 용량당 가격이 높기 때문에 DRAM을 전면 대체하기는 어려운 상황이다. 따라서, 본 논문에서는 I/O 성능을 높이기 위한 추가적인 구성 요소로 영속 메모리를 채택하고자 한다. 그림 1은 영속 메모리를 추가한 새로운 버퍼 캐시 아키텍처를 보여주고 있다. 이러한 아키텍처에서는 영속 메모리의 용량이 한정적이므로, 공간 효율적인 관리 기술이 중요하다.

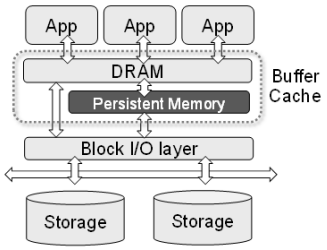


그림 1. 영속 메모리를 추가한 버퍼캐시 구조
Fig. 1. Buffer cache architecture with persistent memory

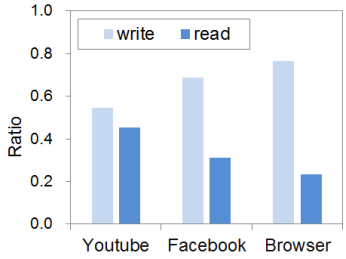
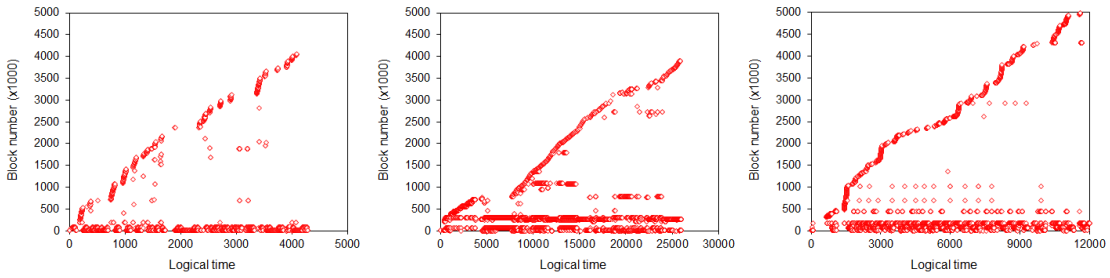


그림 2. 스마트폰 앱의 파일 읽기/쓰기 비율
Fig. 2. File read/write ratio of smartphone applications

본 논문에서는 스마트폰에서 발생하는 파일의 쓰기 연산 특성을 분석하였으며, 그 결과 소량의 영속 메모리로도 주기적인 쓰기 연산을 대폭 줄일 수 있음을 보인다. 본 논문에서 분석한 스마트폰 파일의 쓰기 연산특징은 크게 세 가지로 요약된다. 첫째, 스마트폰 앱은 데스크탑이나 서버 프로그램과 달리 읽기 연산 대비 파일의 쓰기 연산 비율이 매우 높다. 둘째, 일부 핫 데이터가 스마트폰 파일 쓰기의 대부분을 차지하며 이러한 핫 데이터는 앱이 실행되는 동안 지속적으로 참조된다. 셋째, 스마트폰 앱에서 접근하는 파일의 상당 부분은 1회성 참조이다. 이러한 분석을 바탕으로 본 논문에서는 선별적 플러싱이라는 새로운 버퍼 캐시 관리 정책을 제안한다. 제안하는 정책은 파일시스템이 버퍼 캐시의 수정 데이터를 플러시하는 주기가 도래할 때 이를 스토리지에 모두 반영하는 대신 영속 메모리와 스토리지에 선별적으로 반영한다. 수정된 모든 데이터를 영속 메모리에 반영할 경우 한정된 영속 메모리 공간이 빠르게 포화 상태에 이르러 결국 스토리지로 2차 플러시가 유발될 수밖에 없으므로 제안하는 기법은 이를 방지하기 위해 선별적 플러시 정책을 사용한다. 이러한 정책은 스마트폰 파일 쓰기의 특성 분석에 근거하며, 빈번히 수정되는 핫 데이터의 경우 영속 메모리에, 그렇지 않은 1회성 쓰기 데이터의 경우 스토리지에 직접 플러시하여 영속 메모리의 공간 효율성과



(a) Youtube (b) Facebook (c) Browser

그림 3. 시간에 따른 쓰기 발생 블록 번호
Fig. 3. Block numbers that have been written as time progresses

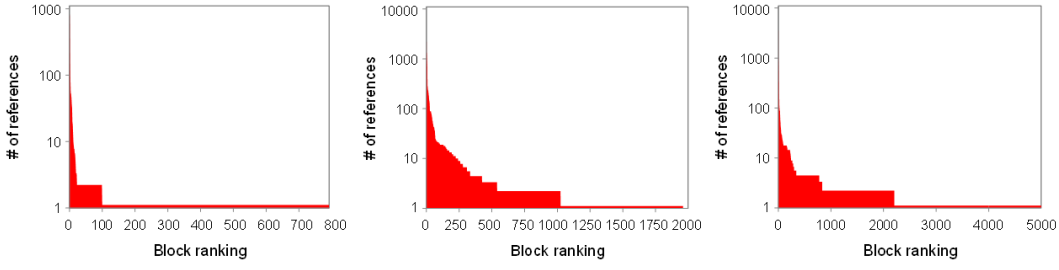
스토리지 접근을 줄이는 두 가지 목적을 모두 달성한다. 스마트폰 앱의 스토리지 접근 트레이스를 사용한 재현 실험을 통해 제안하는 선별적 플러시 정책이 스토리지 쓰기 I/O를 평균 24.8%, 최대 37.8%까지 줄이는 것을 확인하였다.

II. 스마트폰 파일의 쓰기 특성 분석

본 장에서는 스마트폰 파일 접근의 쓰기 특성을 분석한다. 분석을 위해 Facebook, Youtube, Browser 등 3종의 앱에서 발생하는 파일 접근 트레이스를 추출하였다. 트레이스 수집을 위한 앱의 실행 시간은 앱당 15~20분으로 하였으며, 그림 2는 수집한 트레이스의 파일 읽기/쓰기 비율을 보여준다. 그림에서 보는 것처럼 3가지 앱 모두 쓰기 연산 비중이 높게 나타났다. 이러한 높은 파일 쓰기 비율은 PC나 서버 응용에서 읽기 비율이 높은 경우가 많은 것과는 상이한 특성으로, 이는 스마트폰 앱이 파일 조작에 SQLite 데이터베이스를 사용하기 때문인 것으로 분석된다.

그림 3은 시간이 흐름에 따라 쓰기가 발생한 데이터의 블록 번호를 보여준다. 이 그림에서 x축의 시간은 데이터에 대한 쓰기 연산이 1번 발생할 때마다 1씩 증가하는 논리적 시간의 개념이다. 그림에서 보는 것처럼 시간이 흐름에 따라 새로운 데이터가 사용되는 경향이 꾸준히 나타나지만, 한편으로는 일부 낮은 번호의 데이터가 시간에 무관하게 지속적으로 사용되는 특성을 보였다. 이와 같은 데이터를 본 논문에서는 쓰기 핫 데이터로 규명하고 이를 영속 메모리에 저장할 주요 타겟으로 설정하였다.

그림 4는 스마트폰 파일의 쓰기 횟수를 기반으로 데이터 블록의 순위를 부여하고 순위에 따른 분포를 표시하였다. 그림의 x축은 쓰기 횟수에 따른 데이터 블록의 순위를 나타내고 y축은 해당 순위의 데이터에 대한 총 쓰기 횟수를 나타낸다. 그림에서 볼 수 있듯이 상위 순위의 특정 데이터가 전체 쓰기 접근의 상당 부분을 차지하고 있다. 이와 반대로 쓰기 횟수가 1회에 불과한 데이터도 다수가 존재함을 확인할 수 있다. 특히 Youtube의 경우 100위에서 800위까지의 데이터가 1회성 쓰기를 유발한 블록에 해당됨을 볼 수 있다. 나머지 두 앱에서도 쓰기가



(a) Youtube (b) Facebook (c) Browser

그림 4. 블록 순위에 따른 쓰기 횟수
Fig. 4. Number of write accesses for the given block ranks

발생한 파일 데이터의 절반 이상이 1번만 쓰여진 블록임을 확인할 수 있다. 이렇게 높은 비율을 차지하는 1회성 쓰기 데이터의 경우 영속 메모리에 보관하더라도 다시 사용되지 않기 때문에 결국 스토리지에 쓰기를 유발하게 된다. 즉, 이러한 1회성 쓰기 데이터는 스토리지로의 플러시 횟수를 줄이는 데 기여하지 못하므로 공간에 제약이 있는 영속 메모리의 보관 대상으로 적절하지 않다는 의미이다.

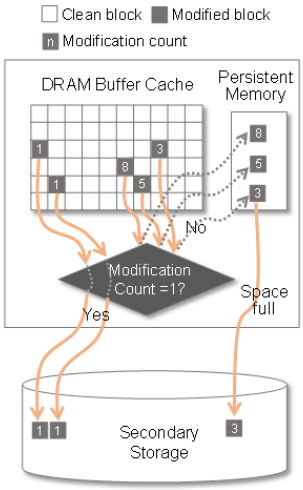


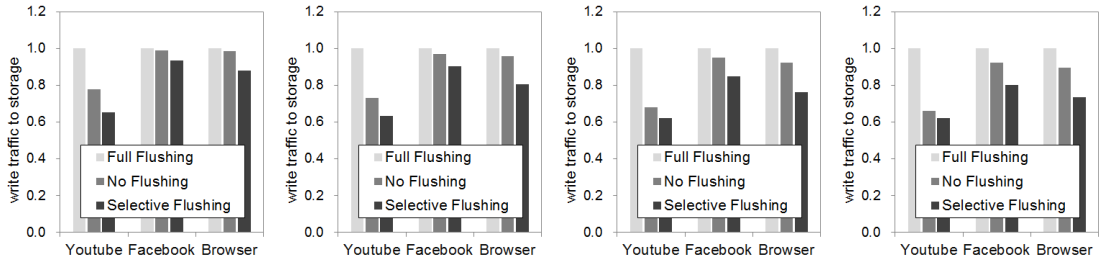
그림 5. 선별적 플러시 정책의 기본 동작
Fig. 5. Basic workings of the selective flushing policy

III. 선택적 플러싱 기법

버퍼 캐시에 파일 데이터를 저장하는 기본 단위는 블록이다. 한편, 버퍼 캐시에 저장된 블록은 캐시에 들어온 후 변경되지 않은 클린 블록과 쓰기 연산이 발생해 변경된 수정 블록으로 나뉜다. 클린 블록의 경우 캐시에 저장된 데이터와 스토리지에 있는 원본 데이터가 동일하다. 따라서, 클린 블록이 버퍼 캐시에서 방출될 때에는 단순히 삭제하는 것으로 충분하다. 반면 수정 블록이 캐시에서 방출될 때에는 삭제하기 전에 이를 스토리지의 원본 위치에 반영해야 한다. 또한, 돌발적인 전원 오류에 대비하기 위해 파일 시스템은 버퍼 캐시에서 삭제될 때까지 수정 블록의 스토리지 반영을 지연하는 대신 주기적으로 플러시하는 정책을 사용한다[10]. 예를 들어, Ext4 파일 시스템의 경우 매 5초마다 수정 블록을 플러시한다. 또한, 스마트폰 앱은 SQLite를 사용하여 동기식 쓰기를 자주 요청한다. 따라서 수정 블록을 스토리지로 반영하는 작업은 데스크탑 시스템에 비해 더욱 빈번히 발생한다. 한편, 수정 블록이 스토리지로 반영된 후에는 캐시에 저장된 데이터와 스토리지 데이터가 동일한 버전이 되므로 수정 블록의 상태는 클린 블록으로 바뀐다.

본 논문의 II장에서는 스마트폰 파일 접근 분석을 통해 제한된 수의 파일 블록이 스토리지에 지속적인 쓰기 연산을 발생시키는 반면 한 번만 쓰기가 발생하는 블록이 다수인 것을 확인한 바 있다. 이러한 특성을 고려하여 선별적 플러시 정책에서는 수정 블록을 스토리지 또는 영속 메모리에 선별적으로 반영한다. 그림 5는 제안하는 정책의 동작 방식을 보여주고 있다. 소량의 영속 메모리를 DRAM 버퍼 캐시와 스토리지 사이에 위치시키고 빈번한 쓰기 발생 블록들을 흡수하여 스토리지의 쓰기 트래픽을 흡수한다. 영속 메모리로 플러시할 대상 블록의 선정은 DRAM 버퍼 캐시에서 해당 블록이 수정된 횟수를 기준으로 한다. 수정 횟수가 1인 경우 해당 블록은 더 이상 쓰기연산이 발생하지 않을 블록으로 분류해서 이를 스토리지로 직접 플러시한다. 반면, 1회성 쓰기 블록이 아닌 경우 플러시 위치를 스토리지가 아닌 영속 메모리로 한다.

이제 그림 5를 통해 제안한 기법의 동작을 좀더 자세히 살펴보도록 하겠다. 스마트폰 앱이 파일을 사용하고자 하는 경우 파일 시스템은 해당 블록이 DRAM 버퍼 캐시에 존재하는지 먼저 검색한다. 버퍼 캐시에서 해당 블록이 발견되면 캐시된 블록을 직접 전달한다. 그렇지 않은 경우 요청된 블록을 스토리지에서 찾은 후 DRAM 버퍼 캐시에 읽어 들인다. 이때 제안하는 기법은 각 블록에 대한 수정 횟수를 유지한다. 수정 횟수는 앱이 해당 블록에 쓰기 요청을 할 때마다 1씩 증가시킨다. 앱이 동기식 쓰기를 요청하거나 파일 시스템의 플러시 주기가 도래하면 DRAM 버퍼 캐시의 수정 블록들은 스토리지와 영속 메모리 중 한 곳으로 플러시된다. 블록의 수정 횟수가 1이면 스토리지로, 1보다 크면 영속 메모리로 플러시 위치가 결정된다. 이는 아직까지 영속 메모리의 단위 용량당 가격이 높기 때문에 공간 효율성을 고려한 관리 정책이며, 1회성 쓰기 데이터는 영속 메모리에 저장하더라도 스토리지의 쓰기 횟수를 줄이는 효과가 없다는 점을 감안한 것이다. 스토리지나 영속 메모리로 수정 데이터가 플러시된 후에는 DRAM 버퍼 캐시의 블록 상태를 클린 블록으로 변경한다.



(a) PM = 10 blocks (b) PM = 20 blocks (c) PM = 30 blocks (d) PM = 40 blocks

그림 6. 선별적 플러싱, 전체 플러싱, 노 플러싱 정책의 비교
Fig. 6. Comparison of selective flushing, full flushing, and no flushing

Ⅳ. 성능 평가

본 장에서는 트레이스 기반 시뮬레이션을 통해 제안하는 기법인 Selective Flushing의 효과를 Full Flushing, No Flushing과의 비교를 통해 검증한다. Full Flushing은 버퍼 캐시의 모든 수정 블록을 스토리지로 직접 플러시하는 방법을 뜻하며, No Flushing은 수정 블록을 모두 영속 메모리로 플러시하는 기법을 뜻한다.

그림 6은 영속 메모리 크기에 따른 세 가지 정책의 스토리지 쓰기량을 보여주고 있다. 그림에서 보는 것처럼 Selective Flushing은 모든 앱과 영속 메모리 크기에 대해 스토리지 쓰기량을 크게 줄이는 것을 확인할 수 있다. Selective Flushing의 스토리지 쓰기량 감소폭은 Full Flushing에 비해 평균 24.8%, 최대 37.8%, No Flushing에 비해 평균 10.1%, 최대 16.0%에 이르렀다. Selective Flushing은 자주 업데이트되는 데이터를 스토리지에 곧바로 반영하지 않고 영속 메모리에 흡수하므로 Full Flushing에 비해 스토리지 쓰기량이 크게 줄어든다. Youtube의 경우 스토리지 쓰기량을 30% 이상 줄였는데 이는 앱의 특성상 스토리지에 발생하는 쓰기 연산이 상대적으로 적어 대부분의 수정 블록을 영속 메모리에 저장하는 것이 가능했기 때문이다. 이에 비해 Facebook과 Browser는 쓰기량 감소폭이 상대적으로 적은데, 이는 그림 4에서 보는 것처럼 쓰기량 순위가 100위를 넘어서는 블록에서도 많은 쓰기 연산이 발생하여 소량의 영속 메모리로 쓰기 연산을 흡수하기에 충분하지 않기 때문이다.

No Flushing과 비교한 Selective Flushing의 쓰기량 절감 효과는 Facebook과 Browser의 경우 영속 메모리 크기가 커질수록 더 높은 것을 확인할 수 있는데 이는 이 두 앱의 쓰기 요청이 많아 상대적으로 더 큰 영속 메모리가 소요되기 때문이다. 이 경우 영속 메모리에 저장할 데이터를 선별하여 공간 효율성을 높이는 것이 중요하다는 점을 암시한다. 반면 Youtube의 경우 영속 메모리가 커질수록 성능 격차가 줄어드는데, 이는 앱의 특성상 스토리지 쓰기량이 상대적으로 많지 않아 영속 메모리의 용량이 충분할 경우 Selective Flushing의 효과가 감소하기 때문이다.

Ⅴ. 결 론

스마트폰 데이터의 크기가 증가함에 따라 스토리지 I/O가 스마트폰 전체의 성능 병목을 일으키고 있다. 버퍼 캐시는 느린 스토리지 성능을 완충하는 중요한 역할을 하지만 쓰기 연산이 발생한 데이터의 경우 플러시를 통한 잦은 스토리지 반영이 필요하여 버퍼 캐시의 효율성을 크게 떨어뜨린다. 본 논문에서는 스마트폰 앱의 쓰기 I/O 특성을 분석하고 주기적 플러시 오버헤드를 효율적으로 제거할 수 있는 방법을 제시하였다. 제안하는 기법은 기존의 DRAM 버퍼 캐시에 소용량의 영속 메모리를 추가하고 이를 통해 스토리지에 발생하는 잦은 플러시의 오버헤드를 제거한다. 즉, 플러시를 스토리지 대신 영속 메모리로 우회시켜 스토리지 쓰기량을 크게 줄였으며, 영속 메모리의 공간효율성을 위해 1회성 쓰기 데이터의 경우 영속 메모리를 거치지 않고 스토리지로 곧바로 플러시되도록 하였다. 실제 스마트폰앱에서 추출한 파일 접근 트레이스를 이용한 재현 실험을 통해 제안하는 기법이 스토리지 쓰기량을 크게 감소시키는 것을 확인하였다.

References

[1] S. Han, "Analysis of content of YouTube channel about pet: focus on cream heroes," The Journal of the Institute of Internet, Broadcasting and Communication, vol. 19, no. 3, pp.33-39, 2019.
DOI: https://doi.org/10.7236/JIIBC.2019.19.3.33

[2] D. Kim, S. Lee, and H. Bahn, "An adaptive location detection scheme for energy-efficiency of smartphones," Pervasive and Mobile Computing, vol. 31, pp. 67-78, 2016.
DOI: https://doi.org/10.1016/j.pmcj.2016.04.012

[3] E. Lee and H. Bahn, "Electricity usage scheduling in smart building environments using smart devices," The Scientific World Journal, vol. 2013, Article 468097, pp. 1-11, 2013.
DOI: https://doi.org/10.1155/2013/468097

[4] B. Choi, S. Eom, C. Kim, and H. Lee, "Counterfeit bill identification based on deep learning using smartphone camera shooting images," Journal of KIIT, vol. 19, no. 3, pp. 1-8, 2021.
DOI: https://doi.org/10.14801/jkiit.2021.19.3.1

[5] J. Park, "Self-Awareness and Coping Behavior of Smartphone Dependence among Undergraduate Students," Journal of the Korea Academia-Industrial cooperation Society (JKAIS), vol. 22, no. 2, pp. 336-344, 2021.
DOI: https://doi.org/10.5762/KAIS.2021.22.2.336

[6] H. Kim, N. Agrawal, and C. Ungureanu, "Revisiting storage for smartphones," ACM Transactions on Storage, vol. 8, no. 4, Article 14, pp. 1-25, 2012.
DOI: https://doi.org/10.1145/2385603.2385607

[7] T. Kim and H. Bahn, "Implementation of the storage manager for an IPTV set-top box," IEEE Transactions on Consumer Electronics, vol. 54, no. 4, pp. 1770-1775, 2008.
DOI: https://doi.org/10.1109/TCE.2008.4711233

[8] O. Kwon, H. Bahn, and K. Koh, "Popularity and prefix aware interval caching for multimedia streaming servers," Proc. 8th IEEE International Conference on Computer and Information Technology, pp. 555-560, 2008.
DOI: https://doi.org/10.1109/CIT.2008.4594735

[9] H. Bahn, H. Lee, S. Noh, S. Min, and K. Koh, "Replica-aware caching for web proxies," Computer Communications, vol. 25, no. 3, pp. 183-188, 2002.
DOI: https://doi.org/10.1016/S0140-3664(01)00365-6

[10] E. Lee, H. Bahn, and S. Noh, "Unioning of the buffer cache and journaling layers with non-volatile memory," Proc. USENIX Conf. on File and Storage Technologies, pp.73-80, 2013.

저 자 소 개

임 수 정(준회원)

- 2007년 3월 ~ : 이화여자대학교 컴퓨터공학과 학사
- 2020년 3월 ~ : 이화여자대학교 컴퓨터공학과 석박통합과정
- 주관심분야 : 운영체제, 스토리지 시스템, 임베디드 시스템

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

※ This work was supported by the National Research Foundation of Korea (NRF) grant funded by the Korea government (MSIT) (No. 2019R1A2C1009275) and also by the ICT R&D program of MSIT/IITP (2020-0-00121, Development of data improvement and dataset correction technology based on data quality assessment).