The Journal of The Institute of Internet, Broadcasting and Communication (IIBC)
Vol. 23, No. 3, pp.13-18, Jun. 30, 2023. pISSN 2289-0238, eISSN 2289-0246

https://doi.org/10.7236/JIIBC.2023.23.3.13
JIIBC 2023-3-2

앱의 특성과 메모리 상황을 고려하는 스마트폰 스와핑 정책

A Smartphone Swapping Policy that Considers the Characteristics of Applications and Memory Situations

반효경*, 김지선

Hyokyung Bahn*, Jisun Kim

요 약 모바일 앱의 수가 지속적으로 증가함에 따라 스마트 폰 메모리 시스템에 스와핑 기능 탑재시 발생하는 오버헤드가 점점 증가하고 있다. 데스크탑이나 서버 시스템과 달리 스마트폰의 기본 세팅에서는 스와핑을 지원하지 않으며, 따라서 가용 메모리가 부족할 경우 앱들은 강제로 종료된다. 이는 스마트폰에 스와핑 기능 탑재시 지나친 스토리지 접근으로 인한 오버헤드가 발생하기 때문이다. 본 논문에서는 스토리지 접근량을 크게 줄이는 스마트폰용 스와핑 정책을 제안한다. 제안하는 정책에서는 모바일 앱을 그 기능적 특성에 따라 카테고리화하고 앱의 우선순위와 메모리 상황을 고려해서 스와핑 대상이 되는 앱의 수를 조절한다. 안드로이드 레퍼런스 디바이스를 활용한 실측 실험을 통해 제안하는 스와핑 정책이 다양한 모바일 앱에 대해 스와핑의 오버헤드를 크게 줄일 수 있음을 확인하였다.

Abstract As the number of mobile applications increases rapidly, the overhead of swapping in smartphone memory systems keeps increasing. Unlike desktop or server systems, the basic setting of smartphones does not support swapping, so applications are killed when available memory is exhausted. This is because swapping in smartphones incurs excessive storage I/O overhead. In this article, we propose a swapping policy for smartphones, which significantly reduces storage I/Os. The proposed policy categorizes mobile applications based on their functional characteristics, and controls the number of applications to be swapped based on application priorities and memory situations. Measurement experiments with Android reference devices show that the proposed swapping policy dramatically reduces the overhead of swapping in various mobile applications.

Key Words : Swapping, mobile app, Android, smartphone, virtual memory.

Ⅰ. 서 론

모바일 플랫폼 기술의 발전과 스마트폰 앱의 증가로 데스크 탑 프로그램의 상당 부분이 모바일 환경에서도 호환성을 가지게 되었다[1, 2, 3]. 이와 더불어 카메라와 센서를 사용하는 모바일 앱이 지속적으로 증가하여 스마트폰의 멀티태스킹 기능이 점점 중요해지고 있다. 특히, 스마트폰의 부팅 후 실행 앱의 수가 증가함에 따라 메모리는 필연적으로 소진되며 이를 관리하기 위한 정책은 스마트폰 성능에서 매우 중요한 요소이다. 현재 안드로이

*정회원, 이화여자대학교 컴퓨터공학과
접수일자 2023년 5월 10일, 수정완료 2023년 5월 30일
게재확정일자 2023년 6월 9일

Received: 10 May, 2023 / Revised: 30 May, 2023 /
Accepted: 9 June, 2023
*Corresponding Author: bahn@ewha.ac.kr
Dept. of Computer Engineering, Ewha University, Korea

드를 비롯한 대부분의 스마트폰 시스템은 메모리의 여유 공간이 일정 수준 이하로 떨어질 경우 최근에 사용되지 않은 앱을 강제로 종료시켜 메모리 공간을 마련한다[4]. 그러나, 이러한 앱의 강제 종료는 사용자가 아닌 시스템 차원의 결정이기 때문에 중요한 앱이 종료되는 경우 앱의 실행 문맥을 잃어버리는 문제가 발생할 수 있다.

예를 들어 주식, 가상화폐 등 금융 앱들의 동시 실행 중 메모리 부족으로 앱들이 강제 종료되는 경우 치명적인 문제가 유발될 수 있다. 이러한 문제를 해결하기 위해 일부 앱들은 메모리 상의 문맥을 파일에 저장하여 강제 종료 후 재실행시 이를 다시 파일 시스템으로부터 읽어 복원하는 방법을 지원하고 있다. 그러나, 이는 시스템 차원이 아닌 사용자 앱 차원에서의 지원이므로 그 기능이 제한적이다.

가상 메모리 스와핑은 운영체제 차원에서 메모리 데이터를 스토리지로 저장할 수 있는 기술로 널리 연구되어 왔다[5, 6]. 그러나, 스마트폰 환경에서의 스와핑은 스토리지 접근 트래픽을 증가시켜 시스템 전체의 성능을 크게 저하시키는 것으로 알려져 있다[5]. 일부 연구에서는 고속 스토리지 매체를 사용해 스와핑으로 인한 성능 저하를 해소하는 정책이 연구된 바 있다[5]. 그러나, 새로운 하드웨어의 탑재는 모바일 기기의 단가를 높이고 아키텍쳐의 변화를 초래하여 기존 모델과의 호환성 문제를 발생시킬 수 있다.

본 논문에서는 새로운 하드웨어를 추가하는 대신 스마트폰 앱의 특성과 시스템의 메모리 상황을 고려하여 스와핑 대상 앱의 수를 조절하는 새로운 스와핑 정책을 제안한다. 제안하는 방식은 모든 앱의 메모리 데이터에 대해 스와핑 기능을 지원하는 대신 앱의 문맥 저장 특성과 중요도를 고려하여 스와핑을 선별적으로 지원한다.

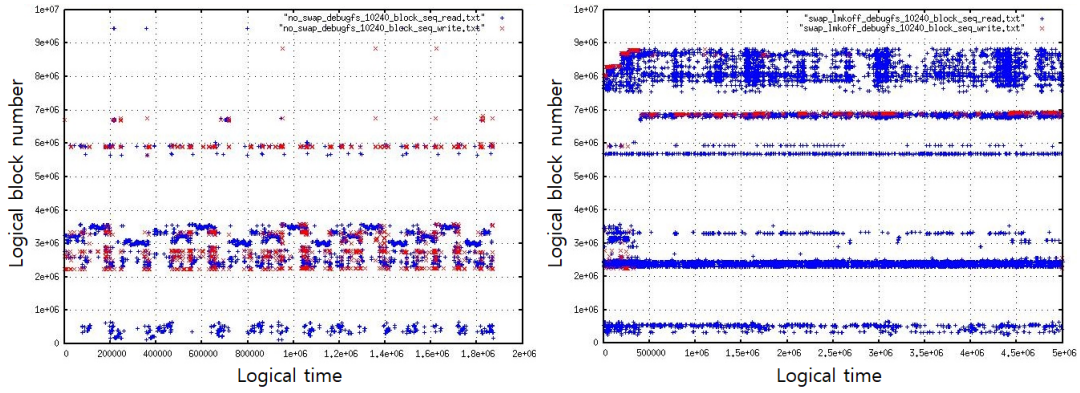
제안하는 정책의 효과를 검증하기 위해 안드로이드 레퍼런스 디바이스 상에서 앱 실행에 따른 지연 시간을 기존의 스와핑 기술 및 안드로이드의 메모리 관리 기술과 비교하였다. 실험 결과 제안하는 기술은 스와핑을 지원하지 않는 방식과 유사한 성능을 나타내는 것을 확인할 수 있었다. 기존의 스와핑과 비교했을 때, 제안하는 정책은 스마트폰 앱들의 시작 지연 시간을 평균 43%까지 개선하는 것을 확인할 수 있었다.

II. 스마트폰의 스와핑 오버헤드 분석

스마트폰의 메모리 시스템에서 스와핑을 지원할 경우 잦은 스토리지 입출력으로 인해 막대한 성능 저하를 초래하는 것으로 알려져 있다[5]. 본 장에서는 다양한 스마트폰 앱들을 실행시킴에 따라 스와핑으로 인한 스토리지 접근 오버헤드가 어떻게 나타나는지를 분석한다. 이러한 실험을 위해 먼저 ftrace 유틸리티를 이용해서 스마트폰 앱들의 스토리지 접근 트레이스를 수집하였다. 스와핑 상황을 살펴보기 위해 일련의 스마트폰 앱들을 차례로 실행시켜 가용 메모리 공간을 모두 채우는 상황을 실현했으며, 충분한 수의 앱이 실행된 이후 일련의 앱들을 반복 실행했다.

실험에 사용한 스마트폰은 안드로이드 레퍼런스 디바이스인 픽셀 폰으로 그림 1은 스와핑을 사용한 경우와 그렇지 않은 경우 스토리지 접근의 분포를 비교해서 보여주고 있다. 그림에서 x축은 스토리지 블록의 참조가 일어날 때마다 1씩 증가하는 논리적인 시간을 나타내며 y축은 스토리지의 블록 번호를 나타낸다. 그림에서 빨간색과 파란색 점들은 각각 쓰기와 읽기 접근을 나타내고 있다. 그림에서 보는 것처럼 스와핑을 사용하는 경우 그렇지 않은 경우에 비해 상당히 많은 수의 스토리지 접근이 발생하는 것을 확인할 수 있다. 이는 스와핑을 사용하지 않는 경우 메모리 공간이 부족할 때 앱을 강제 종료시켜 여유 공간을 확보하기 때문에 메모리 내의 데이터를 스토리지에 저장하는 절차를 거치지 않아 스토리지 접근을 매우 줄일 수 있기 때문이다.

한편, 스와핑을 사용할 경우 그림 1(b)에서 보는 것처럼 스토리지 접근이 특정 블록들에 집중되며 시간이 흘러도 이러한 블록들은 지속적으로 접근되는 특징이 발견되었다. 이에 비해 스와핑을 사용하지 않는 그림 1(a)의 경우에는 지속적인 참조를 나타내는 블록이 잘 드러나지 않는 것을 확인할 수 있다. 이를 좀 더 명확히 확인하기 위해 본 논문에서는 각 스토리지 블록에 대한 접근 횟수를 누적하여 이를 그림 2에 표시하였다. 그림에서 보는 것처럼 스와핑을 사용할 경우 접근 횟수가 200회 이상인 스토리지 블록들이 존재하는 것을 확인할 수 있다. 그러나, 스와핑을 사용하지 않는 경우 이러한 블록들은 발견되지 않았다. 한편, 스마트폰의 메모리 용량이 충분한 경우에는 스와핑을 지원하더라도 이로 인해 성능 저하가 발생한다고 볼 수 없다. 그러나, 스와핑의 대상이 되는 앱의 수가 급격히 증가하게 되면 비록 메모리 용량이 크더라도 필수 서비스 컴포넌트나 공유 라이브러리 등이 메모리로부터 방출되는 상황이 발생하게 되는 것이다. 따라서, 스마트폰에서는 이러한 상황을 방지하기 위해 스와핑의 대상이 되는 앱의 수를 제한하는 정책이 필요하다.



(a) 스와핑을 미지원 스마트폰 (b) 스와핑 지원 스마트폰

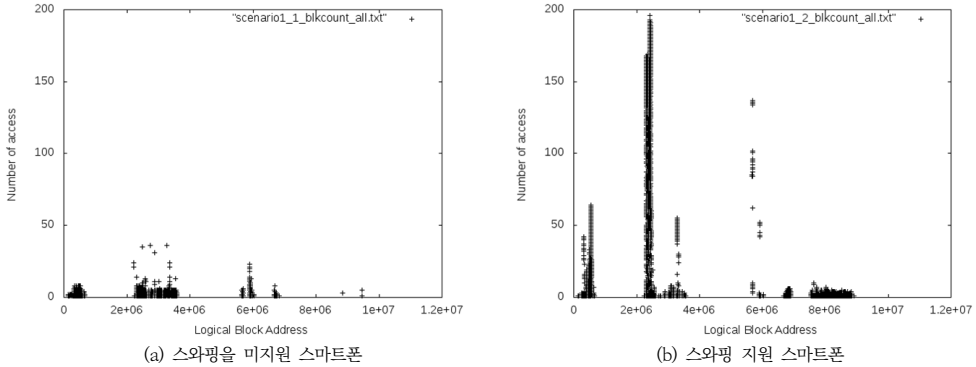
그림 1. 스와핑 지원 여부에 따른 스마트폰의 스토리지 접근 분포
Fig. 1. Distributions of smartphone storage accesses depending on swapping supports.

III. 제안하는 스와핑 정책

스와핑의 오버헤드를 줄이기 위해 본 논문에서는 모바일 앱의 특성과 메모리 상황을 고려해서 스와핑의 대상이 되는 앱의 수를 조절하는 방식을 채택한다. 일부 모바일 앱들은 메모리 상의 데이터를 모두 스왑 영역에 저장하는 대신 소량의 문맥만을 스토리지에 유지하는 것으로 충분하다. 예를 들어 안드로이드의 멀티미디어 플레이어는 재생 중 정지된 영상의 중간 지점을 저장하고 이를 재활성화 시점에 복원한다. 멀티미디어 플레이어에서 재생 중인 시점 이외의 메모리 데이터는 저장할 필요가 없으므로 이러한 앱은 스와핑의 대상에서 제외한다. 한편, 사용자가 하루에 사용하는 앱의 수가 제한적이므로 스와핑의 대상이 되는 앱을 선별하는 작업은 빈번히 발생하지 않는다.

일부 연구에 따르면 스마트폰 사용자의 하루 평균 사용 앱의 수가 10개 남짓이라는 분석이 있다[7]. 그러나, 사용하지 않을 때 전원을 끄고 사용 시점에 재부팅하는 PC와 달리 스마트폰은 오랜 기간 켜두기 때문에 실행 중인 앱의 수는 지속적으로 증가할 수밖에 없다. 따라서, 스마트폰 환경에서 스와핑이 지원될 경우 메모리 용량이 크더라도 시간이 흐름에 따라 사용 앱의 수가 증가하고 결국 메모리 공간은 모두 소진된다. 이에 따라 스와핑이 발생하고 과도한 스토리지 접근으로 인한 시스템의 성능 저하가 발생하게 되는 것이다. 이러한 상황에서 본 연구는 시스템을 정상 상태로 유지하기 위해 스와핑의 대상이 되는 앱의 수를 제한한다.

스와핑의 대상 앱을 선정하기 위해 본 논문에서는 사용 중인 앱들을 우선순위에 따라 클래스로 분류한다. 그런 다음 우선순위가 높은 클래스에 대해서만 스와핑을



(a) 스와핑을 미지원 스마트폰 (b) 스와핑 지원 스마트폰

그림 2. 스와핑 지원 여부에 따른 스마트폰의 스토리지 위치별 접근 횟수
Fig. 2. Number of accesses for each storage location depending on swapping supports.

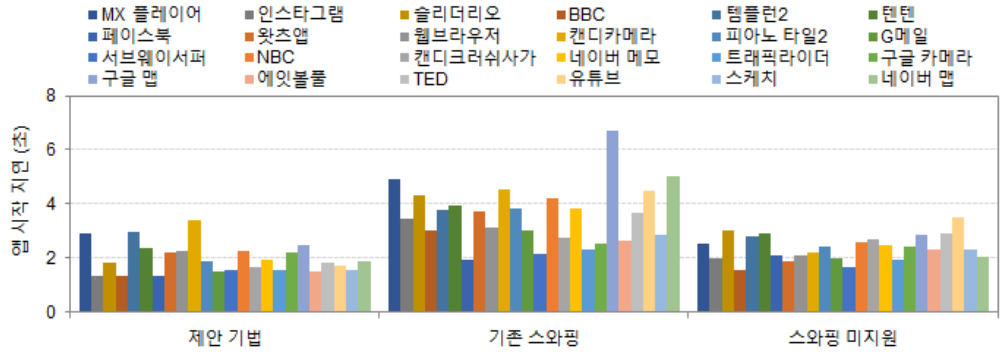


그림 3. 3가지 정책의 적용에 따른 앱들의 초기 지연 시간 비교
Fig. 3. Comparison of each app's launch time when three policies are adopted.

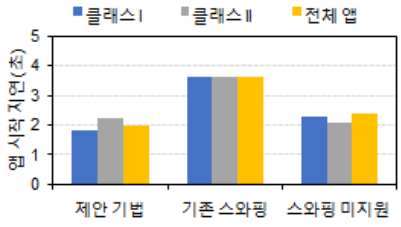


그림 4. 3가지 정책의 성능 비교
Fig. 4. Performance comparison of 3 policies.

지원한다. 이때 낮은 우선순위의 클래스는 안드로이드의 멀티미디어 플레이어처럼 앱이 스스로 필요한 정보를 저장하거나 또는 앱 자체의 중요도가 상대적으로 떨어져 스와핑이 필요하지 않은 앱들로 구성된다. 이러한 앱들은 메모리 공간이 부족할 때 스마트폰의 기본 메모리 관리 정책에 의해 강제 종료된다.

본 논문의 앱별 클래스 분류 방식은 기존 연구와 유사하게 앱들의 기능적 특성 및 중요도에 근거하였다[8]. 본 논문의 앱 클래스 분류에서 가장 높은 클래스에는 금융 앱, 오피스 앱, 위치추적 앱 등을 포함시켰다. 예를 들어, 비트 코인 트레이딩과 차량용 네비게이션은 이러한 클래스에 속한다. 소셜 앱과 스트리밍 앱은 차순위 클래스로 분류하였다. 이러한 앱들은 대부분 네트워크에 기반한 개인화된 서비스를 필요로 하는 앱들이다. 끝으로 가장 낮은 클래스의 앱에는 광고와 뉴스 앱 등을 포함시켰다.

지금부터는 본 논문이 제안하는 스와핑 정책의 동작에 대해 설명하도록 하겠다. 제안하는 정책의 핵심 아이디어는 과도한 스토리지 접근 상황이 발생하지 않도록 스와핑의 대상이 되는 앱의 수를 상황에 따라 조절하는 것이다. 이를 위해 본 논문에서는 앱들을 클래스로 분류하고 시스템의 메모리 상황에 근거해 클래스의 우선순위 중 스와핑 대상이 되는 클래스를 결정한다.

여유 메모리 공간이 임계치 이하에 도달하면 제안하는 스와핑 정책은 메모리 공간 확보를 위해 스와핑과 킬링을 동시에 진행한다. 참고로 킬링 데몬과 스와핑 데몬은 안드로이드와 리눅스에서 각각 메모리 여유공간이 부족할 때 활성화되어 메모리 공간을 확보하는 역할을 한다.

시스템 내의 여유 메모리 공간이 킬링 데몬의 하한선 이하가 되면 제안하는 정책은 낮은 우선순위 클래스의 앱들 중에서 가장 오래전에 사용된 앱을 종료시킨다. 또한, 여유 메모리 공간이 스와핑 데몬의 하한선 이하가 되면 제안하는 정책은 페이지 교체 알고리즘에 근거하여 높은 우선순위에 속하는 앱들의 페이지들 중 방출 대상 페이지를 선정하고 이를 메모리에서 쫓아낸다. 본 논문에서는 리눅스의 디폴트 교체 알고리즘인 2원화된 리스트, 즉 활성 리스트와 비활성 리스트를 이용한 클럭 알고리즘의 확장된 형태를 사용한다. 활성 리스트 상의 페이지들은 최근 사용 이력이 없는 경우 비활성 리스트로 강등되며, 비활성 리스트 상의 페이지는 재참조가 발생하는 경우 활성 리스트로 승격된다. 이때, 메모리로부터의 방출 대상 페이지는 비활성 리스트에서 선정된다[9].

IV. 성능 평가

본 논문에서는 스와핑 정책의 성능 평가를 위해 실측 실험을 하였다. 안드로이드 레퍼런스 디바이스인 픽셀 계열의 폰에서 메모리와 스왑 영역의 크기를 2GB씩으로 설정하고 제안한 스와핑 정책을 구현하였다. 제안한 정책에서는 스와핑 대상으로 선정된 높은 우선순위 클래스

의 앱들을 안드로이드의 킬링 데몬이 종료시키지 못하도록 설정하였다. 또한, 스와핑 데몬을 수정하여 낮은 우선순위 클래스에 속한 앱들을 스와핑 대상에서 제외하도록 구현하였다.

본 논문의 실험에서는 MX 플레이어, 인스타그램, 슬리더리오, BBC, 템플런2, 텐텐, 페이스북, 왓츠앱, 웹 브라우저, 캔디카메라, 피아노 타일2, G메일, 서브웨이 서퍼, NBC, 캔디크러쉬사가, 네이버 메모, 트래픽 라이더, 구글 카메라, 구글 맵, 에잇 볼풀, TED, 유튜브, 스케치, 네이버맵 등 24개의 앱들을 반복적으로 실행시켜 스와핑 방식의 효과에 대해 조사하였다. 비교를 위해 스와핑을 사용하지 않는 방식과 모든 앱에 대해 스와핑을 지원하는 방식에 대해 성능 측정을 하였다. 스와핑을 지원하지 않는 시스템에서는 모든 앱들이 우선순위 클래스와 무관하게 킬링의 대상이 되며, 안드로이드의 기본 세팅은 이와 같은 방식을 채택한다. 전체 스와핑 정책에서는 모든 앱들이 스와핑의 대상이 되며, 킬링은 사용되지 않는다. 이는 리눅스의 기본적인 세팅에 해당한다.

그림 3은 3가지 정책이 구현된 시스템에서 앱들의 실행 시 초기 지연시간의 평균을 보여주고 있다. 그림에서 보는 것처럼 스와핑 미지원 방식에 비해 기존 스와핑 방식은 성능이 크게 저하되는 것을 확인할 수 있다. 구체적으로 기존 스와핑 방식은 53%의 평균적인 성능 저하가 발생했으며, 이는 킬링의 지원 없이 스와핑만을 사용하는 것이 실행 앱의 수가 증가함에 따라 확장성 있는 결과를 나타내지 못함을 뜻한다. 기존 스와핑 방식과 달리 제안하는 스와핑 정책은 거의 성능 저하가 나타나지 않는 것을 확인할 수 있다. 제안하는 정책의 기존 스와핑 방식 대비 성능 개선 효과는 평균 43%에 이르렀으며, 스와핑 미지원 정책에 비해서도 16%의 평균적인 성능 개선 효과를 나타내었다. 이는 제안하는 정책이 메모리가 부족할 경우에도 공유 라이브러리나 안드로이드 서비스와 같은 필수적인 내용을 담는 메모리 페이지를 방출하지 않고 장기간 미사용된 앱을 종료시켜 메모리 공간을 확보하며, 그러고도 메모리가 부족할 때에 최근 미사용된 페이지를 스와핑하는 단계적인 정책을 펼치기 때문이다.

그림 4는 3가지 정책에 대한 실험 결과 앱들의 평균적인 초기 지연 시간을 비교해서 보여주고 있다. 그림에서는 스와핑의 대상이 되는 높은 우선 순위 클래스의 앱들(클래스 I)과 킬링의 대상이 되는 낮은 우선 순위의 앱들(클래스 II)에 대한 결과를 구분해서 보여주고 있다. 그림에서 보는 것처럼 높은 우선 순위의 앱들은 본 논문이 제안하는 정책 하에서 가장 우수한 성능을 나타내었다. 이는 스와핑 대상이 되는 앱의 수를 일정 수준 이내로 유지하고 낮은 우선순위의 앱들을 먼저 종료시켜 스와핑으로 인한 빈번한 스토리지 접근을 막았기 때문이다. 이를 통해 낮은 우선순위의 앱들에 대해서도 일정 수준 이상의 성능을 유지할 수 있었으며, 가용 메모리 공간이 소진되더라도 높은 우선순위의 앱에 대한 메모리 데이터를 가급적 메모리에 유지시켜 우수한 성능을 나타낼 수 있었다.

V. 결 론

본 논문에서는 앱의 특성과 메모리 상황을 고려하는 스마트폰의 스와핑 정책을 제안하였다. 제안한 정책은 스와핑의 대상이 되는 앱의 수가 증가할 경우 스마트폰의 성능이 크게 저하되는 것에 근거하여 메모리 상황에 맞게 스와핑 대상 앱의 수를 조절하도록 하였다. 이를 위해 스마트폰의 앱들을 그 기능적 특성에 따라 우선순위화하고 높은 우선순위의 앱들에 대해서는 스와핑을 선별적으로 지원하였다. 안드로이드 레퍼런스 디바이스를 활용한 실험을 통해 제안하는 스와핑 정책이 기존의 스와핑 정책 대비 앱의 시작 지연시간을 크게 줄이는 것을 확인할 수 있었다. 또한, 스와핑을 지원하지 않는 시스템과 비교해서도 성능저하가 발생하지 않는 것을 확인할 수 있었다.

References

[1] 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

[2] I. Nayeem and R. Want, “Smartphones: past, present, and future,” IEEE Pervasive Computing, vol. 13, no. 4, pp. 89-92, 2014
DOI: https://doi.org/10.1109/MPRV.2014.74.

[3] B. Lee and C. Son, “Improving evaluation metric of mobile application service with user review data,” Journal of the Korea Academia-Industrial cooperation Society, vol. 21, no. 1, pp. 380-386, 2020.
DOI: https://doi.org/10.5762/KAIS.2020.21.1.3

[4] S. Kim, J. Jeong, J. Kim, and S. Maeng, “SmartLMK: a memory reclamation scheme for improving user-perceived app launch time,” ACM Trans. Embed. Comput. Syst., vol. 15, no. 47, pp. 1-25, 2016.
DOI: https://doi.org/10.1145/2894755

[5] J. Kim and H. Bahn, "Analysis of smartphone I/O characteristics — toward efficient swap in a smartphone," IEEE Access, vol. 7, pp. 129930-129941, 2019. DOI: https://doi.org/10.1109/ACCESS.2019.2937852.

[6] S. Yoon, H. Park, K. Cho, and H. Bahn, "Supporting swap in real-time task scheduling for unified power-saving in CPU and memory," IEEE Access, vol. 10, pp. 3559-3570, 2022. DOI: https://doi.org/10.1109/ACCESS.2021.3140166

[7] S. Perez, "Report: Smartphone owners are using 9 apps per day, 30 per month," Technical Report, http://goo.gl/gnrvsF, 2017.

[8] 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

[9] S. Lee, H. Bahn, and S. 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. DOI: https://doi.org/10.1109/TC.2013.98

저 자 소 개

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

김 지 선(준회원)

- 2011년 2월: 한신대학교 컴퓨터공학과 학사
- 2014년 3월 ~ : 이화여자대학교 컴퓨터공학과 대학원생
- 주관심분야 : 운영체제, 스토리지 시스템

※ This work was partly supported by the National Research Foundation of Korea (NRF) grant funded by the Korea government (MSIT) (No. 2019R1A2C1009275) and the Institute of Information & communications Technology Planning & Evaluation (IITP) grant funded by the Korea government (MSIT) (No.2021-0-02068, Artificial Intelligence Innovation Hub).