The Journal of The Institute of Internet, Broadcasting and Communication (IIBC)
Vol. 23, No. 1, pp.71-76, Feb. 28, 2023. pISSN 2289-0238, eISSN 2289-0246

https://doi.org/10.7236/JIIBC.2023.23.1.71
JIIBC 2023-1-11

읽기 참조와 쓰기 참조의 특성을 구분하는 메모리 관리 정책의 설계

Design of a Memory Management Policy Separating the Characteristics of Read and Write References

반효경*

Hyokyung Bahn*

요 약 최근 메모리 관리의 효율성을 높이기 위해 읽기 참조와 쓰기 참조의 기록을 별도로 활용하는 전략이 주목받고 있다. 이는 읽기/쓰기 시간이 비대칭적인 저장 매체의 출현과 읽기/쓰기 참조의 소프트웨어적 특성이 상이한 점을 반영하기 위해 필요한 전략이다. 한편, 기존의 연구들은 메모리 페이지에 읽기와 쓰기 중 어떤 참조가 발생했는지 운영체제가 구분할 수 있다는 가정을 하고 있으나, 대부분의 메모리 아키텍처는 이들을 구분할 수 있는 방안을 지원하지 않는다. 본 논문에서는 기존 연구와 달리 소프트웨어적인 방법으로 메모리 페이지에 발생하는 읽기 쓰기 참조의 특성을 반영하는 방법을 제안한다. 제안하는 방법은 참조 비트와 수정 비트를 이용해 각 페이지의 읽기 및 쓰기 기록을 추정하며, 시뮬레이션을 통해 하드웨어적인 지원이 있는 기존 연구와 거의 유사한 효과가 있음을 보인다.

Abstract Recently, a memory management strategy that utilizes read and write references separately is attracting attention. This is due to the emergence of new storage media with asymmetric read/write latencies and different read/write access characteristics of software. Existing research assumes that operating systems can differentiate between read/write references that occur on each memory page, but most memory architectures do not support a way to distinguish them. Unlike previous studies, this paper proposes a software method that reflects the read/write characteristics of page references by utilizing the reference and modified bits of each page. Simulations show that the proposed policy has almost similar effects to existing studies with hardware support.

Key Words : Memory management, read, write, memory reference, CLOCK

Ⅰ. 서 론

메모리 시스템에서 발생하는 페이지 참조는 시간 지역성(temporal locality)이 강한 것으로 알려져 있다[1]. 즉, 최근에 참조된 메모리 페이지가 다시 참조될 가능성이 높다는 의미이다. 이러한 성질을 활용하는 메모리 관리 정책으로 CLOCK 알고리즘 및 이를 변형한 방식이 널리 사용되고 있다[2]. 그러나, 이러한 방식은 읽기 참조와 쓰기 참조의 구분 없이 최근에 사용된 페이지를 메모리에 유지하는 단순한 전략을 가지고 있다.

*정회원, 이화여자대학교 컴퓨터공학과
접수일자 2022년 12월 30일, 수정완료 2023년 1월 30일
게재확정일자 2023년 2월 3일

Received: 30 December, 2022 / Revised: 30 January, 2023 / Accepted: 3 February, 2023
*Corresponding Author: bahn@ewha.ac.kr
Dept. of Computer Engineering, Ewha University, Korea

한편, 최근 읽기 참조와 쓰기 참조를 분리해서 메모리 관리의 효율성을 높일 수 있는 전략이 연구되고 있다[2, 3]. 이는 비대칭적 읽기/쓰기 시간을 가진 플래시메모리나 NVRAM 등의 스토리지 매체가 도입되면서 활성화된 연구 주제이다[4, 5]. NVRAM은 메모리 혹은 스토리지 계층으로 모두 사용 가능하지만 DRAM 대비 느린 읽기쓰기 속도로 인해 메모리로는 주로 저전력 임베디드용으로 언급되고 있으며[6, 7], 스토리지 성능을 가속하는 용도로 더 많이 주목받고 있다[8]. 한편, 플래시메모리나 NVRAM이 스토리지로 사용되는 시스템에서는 스토리지에서 메모리로 읽어오거나 메모리 내의 데이터를 스토리지로 반영할 때 소요되는 시간이 크게 달라 이를 고려한 메모리 관리 정책이 필요하다.

이러한 관점에서 메모리 상의 페이지들은 수정 페이지와 무수정 페이지로 나누어 볼 수 있다. 무수정 페이지는 스토리지로부터 메모리에 올라온 이후 읽기 참조만 발생한 페이지로 이미 스토리지에 원본 데이터가 존재하므로 메모리에서 방출될 때 그냥 지워버리면 된다. 반면, 수정 페이지는 메모리에 올라온 이후 적어도 한번 이상의 쓰기 참조가 발생한 페이지로 이러한 페이지가 메모리에서 방출될 때에는 스토리지에 먼저 반영한 후 삭제해야 한다. 즉, 수정 페이지는 스토리지에 쓰기 연산을 동반하므로 플래시메모리나 NVRAM처럼 쓰기 연산이 느린 매체에서는 메모리 상의 우선순위를 높일 필요가 있다[2].

한편, 메모리 페이지의 읽기 참조와 쓰기 참조는 그 특성이 상이한 것으로 밝혀진 바 있다[1, 2]. 읽기 참조의 경우 시간 지역성이 강하여 최근에 참조된 페이지의 재참조 가능성이 높은 반면, 쓰기 참조는 시간 지역성이 비교적 약하고 불규칙적인 것으로 확인되었다. 이러한 특성을 반영하는 메모리 관리 정책이 연구된 바 있으나 이는 하드웨어적으로 읽기 참조와 쓰기 참조를 구분할 수 있는 기능이 지원되는 것을 전제하고 있다[2]. 그러나, 메모리 시스템에서 각 페이지의 읽기와 쓰기 참조 발생 정보를 구분해내는 것은 쉬운 일이 아니다. 이는 메모리 참조 시 하드웨어적으로 발생하는 비트 세팅이 읽기와 쓰기를 구분하지 않기 때문이다. 메모리 참조를 관리하는 하드웨어는 읽기와 쓰기의 구분 없이 모든 메모리 참조 시 해당 페이지의 참조 비트(reference bit)를 1로 세팅한다. 또한, 쓰기 참조가 발생하는 경우 수정 비트(modified bit)를 1로 세팅한다. 따라서, 이러한 연산별 참조 성향을 구분하기 위해서는 하드웨어 아키텍처의 변경을 필요로 한다.

본 논문에서는 하드웨어 아키텍처의 변경 없이 소프트웨어적인 방법으로 메모리 페이지의 읽기/쓰기 특성 차이를 메모리 관리 정책에 반영할 수 있는 정책을 제안한다. 제안하는 방식은 현재 메모리 시스템의 하드웨어가 제공하는 참조 비트와 수정 비트를 그대로 사용하여 각 페이지의 읽기 및 쓰기 기록을 추정하며, 읽기 비트와 쓰기 비트가 제공되는 하드웨어 아키텍쳐를 필요로 하는 기존 연구와 유사한 효과를 낼 수 있는 장점을 가진다. 다양한 워크로드의 메모리 참조 트레이스를 이용한 시뮬레이션 실험을 통해 제안하는 방식이 CLOCK 알고리즘 대비 평균 23.5%의 성능 개선 효과를 보이며, 이는 하드웨어 아키텍처 지원이 있는 기존 연구와 거의 흡사한 효과가 있음을 확인할 수 있었다.

II. 관련 연구

1. CLOCK 알고리즘

가상 메모리 환경에서의 페이지 참조는 시간지역성이 강하게 나타나는 것으로 알려져 있다[2]. 시간지역성이란 최근에 참조된 페이지일수록 다시 참조될 가능성이 높은 성질을 말한다. 시간지역성을 가진 참조 패턴에 대해서는 LRU(least recently used) 알고리즘이 최적의 알고리즘으로 알려져 있다[9]. LRU 알고리즘은 메모리 공간이 부족한 경우 가장 오래 전에 참조되었던 페이지를 메모리에서 방출한다. LRU 알고리즘은 메모리 상의 페이지들을 참조 순서에 따라 리스트에 줄 세우고 메모리 참조가 발생할 때마다 해당 페이지를 리스트 상의 최우선순위 위치로 이동시키는 방식을 사용한다. 이러한 방식은 구현이 단순하므로 파일 캐싱이나 웹 캐싱 등 다양한 컴퓨팅 환경에서 널리 활용되고 있다[10, 11, 12]. 그러나, 메모리 시스템에서는 참조가 발생할 때마다 해당 페이지의 참조 시각 정보를 업데이트하거나 소프트웨어적으로 리스트 상의 위치를 변경하는 연산이 사실상 불가능하다. 이는 모든 메모리 참조에 운영체제 커널의 개입이 불가능할 뿐 아니라 하드웨어적인 지원에도 한계가 있기 때문이다.

따라서, 현재의 메모리 시스템 아키텍처에서는 메모리 참조 발생 시 참조된 정확한 시각을 기록하는 대신 페이지의 참조 비트를 세팅하는 정도의 지원이 이루어지고 있다. 이러한 참조 비트를 바탕으로 운영체제는 메모리 상에 빈 공간이 필요할 때 최근에 참조되지 않은 페이지를 방출하기 위해 모든 페이지들을 순차적으로 스캔하면서 참조 비트가 1인 페이지의 경우 이를 0으로 리셋시키

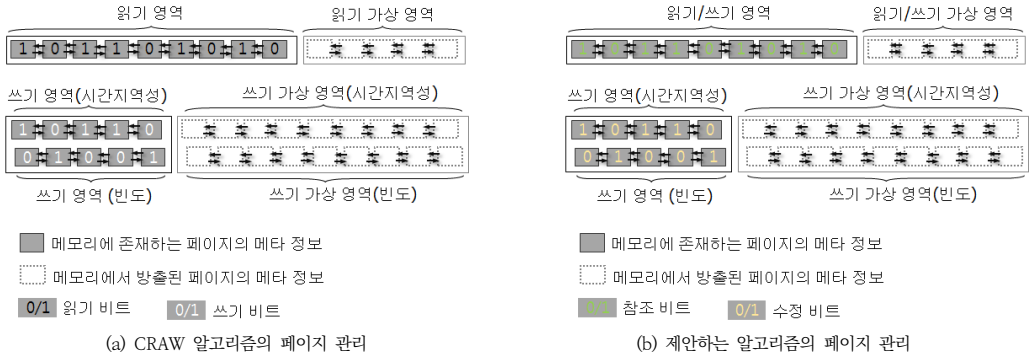

그림 1. CRAW와 제안하는 알고리즘의 페이지 관리 비교
Fig. 1. Comparison of page management in CRAW and the proposed algorithms.

고 참조 비트가 0인 페이지가 발견되면 이를 방출하는 CLOCK 알고리즘을 사용한다.

2. CRAW 알고리즘

읽기 참조와 쓰기 참조를 별도로 활용하는 메모리 관리 알고리즘으로 CRAW(Clock for read and write) 알고리즘이 제안된 바 있다[2]. CRAW 알고리즘은 가상 메모리의 스왑 영역으로 플래시메모리를 사용할 경우 읽기 연산과 쓰기 연산의 비용이 비대칭적임을 메모리 관리에 활용하기 위해 메모리 영역을 읽기 영역과 쓰기 영역으로 나누어 관리한다. 이 때, 플래시메모리에 발생하는 입출력 비용을 최소화하기 위해 읽기 영역과 쓰기 영역의 기여도를 조사하여 두 영역의 크기를 동적으로 조절한다.

CRAW는 각 영역에서 방출할 페이지의 선정을 위해 CLOCK 알고리즘과 유사한 방식을 사용한다. 다만, 참조 비트 대신 읽기 비트와 쓰기 비트를 사용하여 읽기 참조와 쓰기 참조의 특성을 별도로 활용한다. 읽기 참조의 경우 시간지역성이 강하다는 점에 착안하며, 쓰기 참조의 경우 약한 시간지역성을 보완하기 위해 쓰기 빈도를 함께 활용하여 방출 대상 페이지를 선정한다. 쓰기 영역에 속한 페이지들은 방출될 때 플래시메모리에 쓰기 연산을 발생시키며, 이는 읽기 연산에 비해 2~8배의 비용을 유발하므로 CRAW는 쓰기 영역의 페이지들을 읽기 영역의 페이지에 비해 메모리 보관 우선순위를 높게 부여한다. 하지만, 읽기 영역의 페이지들 또한 재참조 가능성이 높은 경우 상대적인 성능 기여도를 고려해 메모리 내에 보존할 수 있도록 한다.

CRAW는 읽기 영역과 쓰기 영역의 크기 조절을 위해 두 개의 가상 영역인 가상 읽기 영역과 가상 쓰기 영역을 두며, 이때 가상 영역은 메모리에서 방출된 페이지의 메타 정보를 한시적으로 보존하여 해당 페이지가 최근에 방출되었다는 사실을 기억해 둔다. 가상 영역에서의 페이지 참조 정보를 통해 해당 영역의 크기를 확대했을 때 얻게 되는 성능 개선 효과를 예상할 수 있다. 예를 들어 읽기 가상 영역에 속한 페이지가 빈번히 참조된다면 읽기 영역의 크기를 확대하여 성능 개선을 도모할 수 있다. 이 때, CRAW는 가상 영역에서의 페이지 적중뿐 아니라 스토리지에 발생하는 연산의 비용을 함께 고려해서 메모리 영역의 크기를 조절한다.

그림 1(a)는 CRAW가 관리하는 읽기 영역과 쓰기 영역, 그리고 그 가상 영역들의 모습을 간단히 보여주고 있다. 한편, 메모리 참조에서는 동일한 페이지에 대해 읽기 참조와 쓰기 참조가 모두 발생할 수 있다. 따라서, CRAW는 최근에 읽기/쓰기가 모두 발생한 페이지의 경우 두 영역에 동시에 유지한다. 이 경우 실제 데이터는 물리적으로 하나의 페이지에 존재하나 해당 페이지의 메타 정보는 읽기 영역과 쓰기 영역에 모두 연결되어 각 영역의 관리에 활용된다. 즉, 하나의 페이지가 두 영역에 동시에 존재할 수 있으므로 해당 페이지가 물리적 메모리에서 방출되려면 그 페이지가 어느 영역에도 연결돼 있지 않아야 한다.

읽기 영역에서 빈 공간이 필요한 경우 CRAW는 읽기 비트를 체크하면서 CLOCK 알고리즘과 유사한 형태로 페이지들을 스캔한다. 이때, 읽기 비트가 1인 페이지는 0으로 클리어시키고 읽기 비트가 0인 페이지를 만나면 해당 페이지를 읽기 영역에서 방출한다. 쓰기 영역에서 빈 공간이 필요한 경우 읽기 비트가 아닌 쓰기 비트를 활용하며, 알고리즘 자체의 동작은 동일하다.

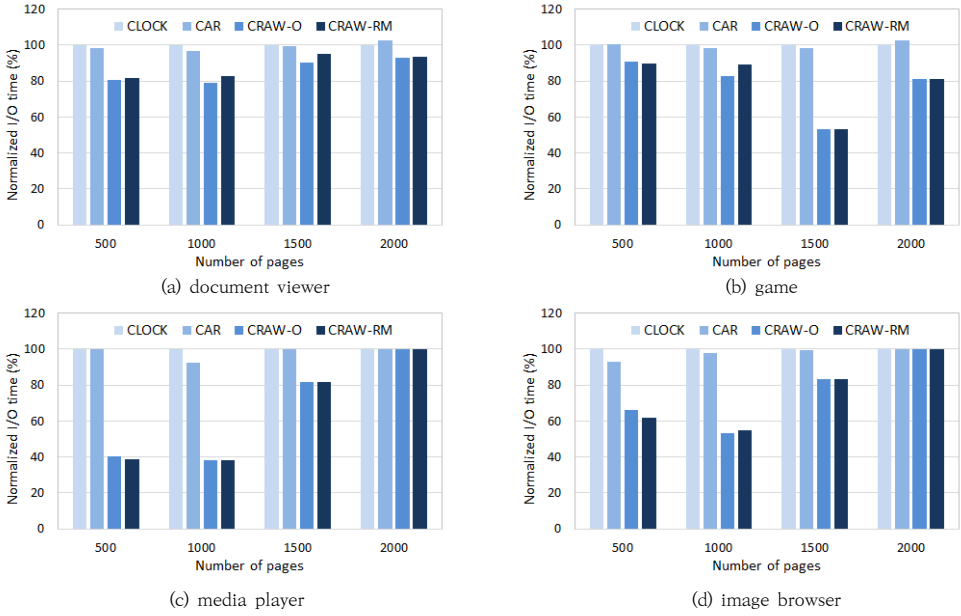

그림 2. 제안하는 메모리 관리 정책의 성능 평가
Fig. 2. Performance evaluation of the proposed memory management policy.

III. 제안하는 메모리 관리 정책

CRAW 알고리즘처럼 읽기 참조와 쓰기 참조를 분리해서 메모리 관리를 수행하는 기법들은 페이지별 읽기와 쓰기 기록을 구분해낼 수 있다는 가정을 한다. CRAW 알고리즘의 경우 읽기/쓰기 참조가 발생할 때 하드웨어적으로 읽기 비트와 쓰기 비트가 세팅되는 것을 가정하고 있다. 이는 페이징 시스템에서 참조 비트와 수정 비트가 세팅되는 것과 유사한 원리이지만 실제 메모리 관리 하드웨어는 읽기/쓰기 비트를 지원하지 않고 있다. 따라서, 읽기/쓰기 참조를 별도로 활용하는 메모리 관리 기법의 구현을 위해서는 하드웨어적인 변경을 통해 읽기/쓰기 비트가 지원되어야만 실현 가능하다.

본 논문에서는 이러한 하드웨어 계층의 변경을 동반하지 않고 운영체제 수준에서 소프트웨어적으로 읽기와 쓰기 참조를 고려한 메모리 관리 기법을 제안한다. 즉, 하드웨어적으로는 전통적으로 지원되는 참조 비트와 수정 비트를 그대로 사용해서 읽기 참조 또는 쓰기 참조가 발생할 때 참조 비트가 1로 세팅되고 쓰기 참조가 발생할 때 수정 비트가 1로 세팅되는 것을 전제로 메모리 관리 기법을 설계한다.

한편, 참조 비트와 수정 비트를 통해 읽기 참조와 쓰기 참조를 분리해내는 것은 현실적으로 불가능하다. 이는 참조 비트가 읽기 또는 쓰기 참조 발생시에 모두 세팅이 되기 때문에 참조 비트로부터 읽기 참조의 발생 정보만을 추출하는 것이 불가능하기 때문이다. 따라서 이러한 상황에서의 유일한 대안은 참조 비트 및 수정 비트가 제공하는 의미에 충실하게 읽기 및 쓰기 참조 정보를 추출하는 것이다. 즉, 참조 비트의 세팅을 통해 발생하는 이벤트는 읽기와 쓰기 모두이므로 페이지의 참조 비트를 통해서는 읽기/쓰기를 포함한 전체 참조의 특성을 추출하고 수정 비트의 세팅을 통해 발생하는 이벤트가 쓰기 참조이므로 수정 비트를 통해 쓰기 참조의 특성을 분리해내는 것이다.

즉, CRAW가 메모리를 읽기 영역과 쓰기 영역을 분리 운영하기 위해 읽기/쓰기 비트를 필요로 하는 것에 비해, 이를 읽기/쓰기 영역과 쓰기 영역으로 분리해서 운영할 경우 참조 비트와 수정 비트를 활용하는 것이 가능하다는 의미이다. 이 경우 어떤 페이지에 쓰기 참조가 발생한 경우 이 페이지는 읽기/쓰기 영역과 쓰기 영역에 모두 포함시키게 되는 것이다. CRAW 알고리즘에서도 하나의 페이지가 두 영역, 즉 읽기 영역과 쓰기 영역에 동시에 포함되는 것이 가능했기 때문에 페이지를 논리적으로 두 영역에 포함시키는 것은 비트의 의미를 훼손하지 않으면

서 영역에 대한 정확한 기여도에 따라 크기 결정을 가능하게 한다. 실제로 CRAW 알고리즘의 분석에 따르면 읽기 참조와 쓰기 참조의 구분 없이 모든 메모리 참조에 대한 시간지역성 분석이 읽기 참조만 추출한 메모리 참조의 시간지역성 분석과 크게 다르지 않다는 점을 통해서도 이러한 방식은 합리화될 수 있다. 실제로 전체 메모리 참조와 특성이 상이한 것은 쓰기 참조만 별도로 추출한 시간지역성이고 이에 대해서는 수정 비트를 통해 운영되는 쓰기 영역의 관리에 활용할 수 있다. 즉, CRAW에서 쓰기 참조의 시간지역성이 약하다는 특성을 메모리 관리에 반영하기 위해 쓰기 영역의 관리에 시간지역성과 쓰기 빈도를 같이 활용한 부분을 본 논문에서도 그대로 활용할 수 있다는 의미이다. 그림 1(b)는 이와 같이 참조 비트와 수정 비트를 지원하는 하드웨어 아키텍처 상에서 읽기/쓰기 연산의 특성을 활용한 메모리 관리 방법을 그림으로 보여주고 있다.

IV. 성능 평가

본 논문에서 제안한 페이지 교체 정책의 성능평가를 위해 메모리 참조 트레이스를 재현하는 시뮬레이션 실험을 수행하였다. 트레이스는 Valgrind 툴셋의 Cachegrind를 통해 추출하였으며[13], 리눅스 상의 4종 워크로드인 document viewer, game, media player, image browser로 구성된다. 그림 2는 메모리 내의 페이지 수가 500개에서 2000개까지 변함에 따라 알고리즘별 전체 입출력 소요 시간을 보여주고 있다. 그래프에서는 CLOCK 알고리즘의 입출력 시간을 100%로 했을 때 다른 알고리즘들의 상대적인 입출력 시간을 보여주고 있다. 그림에서 CAR는 시간지역성과 참조 빈도를 함께 고려하는 메모리 관리 알고리즘이고[14], CRAW-O는 읽기/쓰기 비트가 하드웨어적으로 제공되는 것을 가정하는 오리지널 CRAW 알고리즘을 나타낸다. 본 논문이 제안하는 방식은 CRAW-RM(CRAW with Reference & Modified bits)으로 이는 현재 메모리 아키텍처가 제공하는 참조 비트와 수정 비트를 사용했음을 뜻한다. 그림에서 보는 것처럼 CRAW-O/RM이 CLOCK과 CAR에 비해 우수한 성능을 나타내었으며, media player나 image browser처럼 워크로드의 규모가 작은 경우 2000개의 메모리 페이지 하에서는 모든 알고리즘이 동일한 성능으로 수렴하는 것을 확인할 수 있다. 한편, CRAW-O에서 읽기 비트와 쓰기 비트를 사용하는 대신 참조 비트와 수정 비트를 사용하는 방식인 CRAW-RM은 CRAW-O와 거의 유사한 결과를 나타내는 것을 확인할 수 있다. CLOCK과 비교할 때 CRAW-RM은 평균 23.5%의 성능 개선을 나타내었으며, CAR와 비교한 성능 개선은 평균 22.8%였다.

V. 결 론

최근 읽기 참조와 쓰기 참조를 별도로 활용하는 메모리 관리 정책이 연구되고 있으나, 기존의 연구들은 읽기/쓰기 구분을 위해 메모리 아키텍처의 변화를 필요로 한다. 본 논문에서는 기존 연구와 달리 하드웨어적인 지원 없이 소프트웨어적인 방법으로 메모리 페이지에 발생하는 읽기/쓰기 참조의 특성을 메모리 관리 정책에 반영하는 방법을 제안하였다. 제안하는 방식은 메모리 시스템이 이미 제공하고 있는 참조 비트와 수정 비트를 이용해 메모리 영역을 읽기/쓰기 영역과 쓰기 영역의 분할하고 이를 통해 하드웨어적인 지원이 필요한 기존 연구와 거의 유사한 효과가 있음을 보였다. 또한, 제안하는 방식이 CLOCK 알고리즘 대비 평균 23.5%의 성능 개선 효과를 있음을 시뮬레이션 실험을 통해 확인하였다.

References

[1] 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

[2] H. Lee and H. Bahn, "Characterizing virtual memory write references for efficient page replacement in NAND flash memory," Proc. IEEE MASCOTS Conf., pp.1-10, 2009.
DOI: https://doi.org/10.1109/MASCOT.2009.5366768

[3] S. Lee and H. Bahn, "Characterization of Android memory references and implication to hybrid memory management," IEEE Access, vol. 9, pp. 60997-61009, 2021.
DOI: https://doi.org/10.1109/ACCESS.2021.3074179

[4] I. Shin, "Performance evaluation of applying shallow write in SSDs with internal cache," The Journal of KIIT, vol. 17, no. 1, pp. 31-38, 2019.
DOI: https://doi.org/10.14801/jkiit.2019.17.1.31

[5] H. Bahn and K. Cho, "Implications of NVM based storage on memory subsystem management," Applied

Sciences, vol. 10, no. 3, 2020.
DOI: https://doi.org/10.3390/app10030999

[6] S. Yoo, Y. Jo, and H. Bahn, "Integrated scheduling of real-time and interactive tasks for configurable industrial systems," IEEE Trans. Industrial Informatics, vol. 18, no. 1, pp. 631-641, 2022.
DOI: https://doi.org/10.1109/TII.2021.3067714

[7] 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

[8] Y. Park and H. Bahn, "Modeling and analysis of the page sizing problem for NVM storage in virtualized systems," IEEE Access, vol. 9, pp. 52839-52850, 2021.
DOI: https://doi.org/10.1109/ACCESS.2021.3069966

[9] E. Coffman and P. Denning, Operating Systems Theory, Prentice-Hall, pp. 241-283, 1973.

[10] H. Bahn, S. Noh, S. Min, and K. Koh, "Efficient replacement of nonuniform objects in web caches," Computer, vol.35, no.6, pp.65-73, 2002.
DOI: https://doi.org/10.1109/MC.2002.1009170

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

[12] T. Kang, "BIM geometry cache structure for data streaming with large volume," Journal of the Korea Academia-Industrial cooperation Society (JKAIS), vol. 18, no. 9, pp.1-8, 2017.
DOI: https://doi.org/10.5762/KAIS.2017.18.9.1

[13] N. Nethercote and J. Seward, "Valgrind: a framework for heavyweight dynamic binary instrumentation," ACM SIGPLAN Notices, vol. 42, no. 6, pp. 89-100, 2007.
DOI: https://doi.org/10.1145/1273442.1250746

[14] S. Bansal and D. Modha, "CAR: clock with adaptive replacement," Proc. USENIX FAST Conference, pp. 187-200, 2004.

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

※ 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 Korean government (MSIT) (No.RS-2022-00155966, Artificial Intelligence Convergence Innovation Human Resources Development (Ewha Womans University)).