The Journal of The Institute of Internet, Broadcasting and Communication (IIBC)
Vol. 16, No. 3, pp.7-12, Jun. 30, 2016. pISSN 2289-0238, eISSN 2289-0246

http://dx.doi.org/10.7236/JIIBC.2016.16.3.7
JIIBC 2016-3-2

스마트폰 저장장치의 성능개선을 위한 비휘발성메모리 기반의 버퍼캐쉬 관리

Buffer Cache Management based on Nonvolatile Memory to Improve the Performance of Smartphone Storage

최현경*, 반효경**

Hyunkyoung Choi*, Hyokyung Bahn**

요 약 스마트폰의 메모리로 사용되고 있는 DRAM은 고집적화의 한계로 인해 더 이상 용량 증대가 어려울 뿐 아니라 배터리 소모의 상당 부분을 차지하는 것으로 분석되고 있다. 이에 비해 페이스북 등의 소셜 네트워크 서비스는 점점 많은 메모리를 필요로 하고 메모리 용량 부족시 스토리지를 접근하게 되어 상당히 느린 응답 시간을 나타내고 있다. 본 논문은 스마트폰 저장장치의 성능 개선을 위해 차세대 비휘발성메모리를 버퍼캐쉬로 탑재하고 이를 효율적으로 관리하는 알고리즘을 제안한다. 제안하는 기법은 파일의 쓰기 연산이 발생한 데이터를 비휘발성메모리에 보관하여 스토리지 접근 횟수를 크게 줄이는 동시에, 읽기 연산과 쓰기 연산의 기록을 별도로 유지하여 두 연산 모두 성능 저하가 발생하지 않도록 한다. 트레이스 기반 시뮬레이션 실험을 통해 제안한 기법이 기존 방식에 비해 성능이 개선되는 것을 보인다.

Abstract DRAM is commonly used as a smartphone memory medium, but extending its capacity is challenging due to DRAM's large battery consumption and density limit. Meanwhile, smartphone applications such as social network services need increasingly large memory, resulting in long latency due to additional storage accesses. To alleviate this situation, we adopt emerging nonvolatile memory (NVRAM) as smartphone's buffer cache and propose an efficient management scheme. The proposed scheme stores all dirty data in NVRAM, thereby reducing the number of storage accesses. Moreover, it separately exploits read and write histories of data accesses, leading to more efficient management of volatile and nonvolatile buffer caches, respectively. Trace-driven simulations show that the proposed scheme improves I/O performances significantly.

Key Words : Smartphone, Nonvolatile memory, Buffer cache, Storage.

Ⅰ. 서 론

스마트폰의 메모리와 스토리지로 각각 DRAM과 플래시메모리가 활용되고 있다. DRAM은 고집적화의 한계로 인해 더 이상 용량 증대가 어려운 상황에 이르렀을 뿐 아니라 배터리 소모의 상당 부분을 차지하는 것으로 분석되고 있다[1],[2]. 이에 비해 페이스북 등의 소셜 네트워크 서비스는 점점 많은 메모리를 필요로 하고 메모리 용량 부족시 스토리지를 접근하게 되어 상당히 느린 응답 시간을 나타내고 있다.

*준회원, 이화여자대학교 컴퓨터공학과
**정회원, 이화여자대학교 컴퓨터공학과(교신저자)
접수일자 : 2016년 4월 24일, 수정완료 : 2016년 5월 24일
게재확정일자 : 2016년 6월 10일

Received: 24 April, 2016 / Revised: 24 May, 2016 / Accepted: 10 June, 2016
**Corresponding Author: bahn@ewha.ac.kr
Dept. of Computer Engineering, Ewha University, Korea

최근 MRAM, PRAM 등의 차세대 비휘발성메모리 기술이 기존의 DRAM과 플래시메모리에 의한 성능 저하를 막을 수 있는 새로운 매체로 부각되고 있다[3],[4]. 차세대 비휘발성메모리는 바이트 단위 접근이 가능해 DRAM 대체용 메인 메모리로 사용될 수 있으면서 비휘발성이기 때문에 파일에 대한 쓰기 연산들을 스토리지까지 곧바로 보내지 않고 최대한 메모리 단에서 흡수할 수 있어 성능 개선효과를 얻을 수 있다[5].

본 연구에서는 스마트폰 시스템에 차세대 비휘발성메모리를 버퍼캐쉬로 탑재하여 스마트폰 앱들의 메모리 요구량을 충족하면서 성능을 개선할 수 있는 새로운 관리 기법을 제안한다. 제안하는 기술은 2차 저장장치로는 플래시메모리를 사용하고, 버퍼캐쉬로는 기존의 휘발성 DRAM과 비휘발성메모리를 같이 사용한다. 제안하는 캐쉬 관리 기법은 3가지 특징을 가지고 있다. 첫째, 파일 연산 중 읽기 연산과 쓰기 연산 기록을 별도로 유지함으로써 각 연산의 미래 발생 가능성에 대한 예측의 정확도를 높여준다. 둘째, 쓰기 연산이 발생한 모든 파일 참조 블록들을 비휘발성메모리에 저장하여 크래쉬 발생 시 스토리지의 최신 상태로의 복원을 가능하게 한다. 마지막으로 캐쉬 내 블록들의 저장 위치(즉, 휘발성 DRAM 혹은 비휘발성메모리)와 관계없이 연산 종류에 따라 2가지 리스트로 데이터를 관리하여 효율성을 높인다. 즉, 캐쉬 내 블록들의 읽기 및 쓰기 연산 기록을 관리하기 위해 LRR(least recently read) 리스트와 LRW(least recently written) 리스트를 두어 미래에 발생할 연산에 대한 예측의 정확성을 높인다. 트레이스 기반 시뮬레이션 실험을 통해 제안한 기법이 기존 버퍼캐쉬 시스템에 비해 큰 성능 개선을 얻을 수 있음을 확인하였다.

본 논문의 구성은 다음과 같다. 2장에서는 비휘발성메모리를 이용하는 버퍼캐쉬 기법에 대한 문제 정의를 기술한다. 3장에서는 제안하는 기법에 대해 구체적으로 설명한다. 4장에서는 제안하는 기법의 효과를 검증하기 위한 성능 평가 및 실험 결과를 기술한다. 끝으로 5장에서 본 논문의 결론을 제시한다.

Ⅱ. 비휘발성 버퍼캐쉬의 문제 정의

본 장에서는 비휘발성 버퍼캐쉬 문제를 정의하기에 앞서 전통적인 버퍼캐쉬 문제에 대해 설명한다. 버퍼캐쉬의 크기를 S라 하고, 총 데이터 요청 횟수를 r, 캐쉬 적중 횟수, 즉 스토리지 접근없이 캐쉬에서 직접 서비스된 횟수를 h라 하자. 그러면, 버퍼캐쉬 문제는 캐쉬 용량 S를 초과하지 않으면서 가장 최근에 요청된 데이터 블록을 캐쉬에 보관해야 하는 문제를 의미한다. 본 논문에서는 미래의 데이터 요청 정보를 모르는 상태에서 매 시점 요청되는 데이터를 버퍼캐쉬로 적재하는, 가장 전형적인 버퍼캐쉬 문제를 가정한다. 만약 데이터의 총량이 버퍼캐쉬의 크기인 S를 초과하는 경우 캐쉬에서 방출해야 할 데이터를 캐쉬 교체 알고리즘이 결정한 후 쫓아낸다. 이때, 교체 알고리즘의 목표는 총 데이터 요청 중 스토리지 접근 없이 버퍼캐쉬에서 직접 처리되는 요청의 비율을 극대화하는 것이다. 즉, 전통적인 버퍼캐쉬 환경에서는 캐쉬 적중률(hit ratio)인 h/r이 버퍼캐쉬의 효율성을 평가하는 지표로 사용된다. 이러한 캐쉬 환경에서는 Belady의 MIN 알고리즘이 최적의 알고리즘으로 알려져 있다[6]. MIN 알고리즘은 캐쉬 용량 초과 상황 발생시 가장 먼 미래에 접근될 데이터 블록을 캐쉬에서 방출한다. 실제 시스템 환경에서는 미래의 데이터 접근 여부를 미리 알 수 없기 때문에 MIN 알고리즘은 실제 시스템에 적용 가능한 알고리즘은 아니다. 그러나, 실제 활용 가능한 온라인 알고리즘들의 성능에 대한 상한값을 제공한다는 측면에서 의미가 있다.

지금부터는 비휘발성 버퍼캐쉬에 대한 문제를 정의하도록 하겠다. 용량 및 성능 상의 이유로 비휘발성 버퍼캐쉬가 단독으로 사용되기보다는 휘발성 버퍼캐쉬와 함께 사용될 가능성이 높기 때문에 본 논문에서는 휘발성 및 비휘발성 버퍼캐쉬가 함께 사용되는 환경을 가정한다. 이때, 모든 쓰기 연산은 비휘발성 버퍼캐쉬 상에서 이루어지는 것으로 가정한다. 이는 전원이 나가더라도 최신 데이터가 보존되는 것을 보장하기 위함이다. 따라서, 2차 저장장치에 대한 쓰기 연산 발생은 비휘발성 버퍼캐쉬에서 데이터 블록이 방출되는 경우로 한정된다. 한편, 모든 클린 블록(clean block)은 휘발성 버퍼캐쉬에 존재하는 것으로 가정한다. 클린 블록이란 데이터가 버퍼캐쉬에 적재된 이후 쓰기 연산이 한번도 발생하지 않고 읽기 연산만 발생한 블록을 뜻한다. 이러한 블록은 캐쉬에서 방출될 때 단순히 지워버리기만 하면 되므로 스토리지 쓰기 절감 효과를 얻기 위한 비휘발성 버퍼캐쉬 대신 휘발성 버퍼캐쉬를 사용하는 것이다. 반면, 쓰기 연산이 1번이라도 발생한 수정 블록(dirty block)의 경우 캐쉬에서

방출될 때 2차 저장장치에 수정된 부분을 반영한 후 방출해야 한다.

비휘발성 버퍼캐쉬의 크기를 S_{NV}, 휘발성 버퍼캐쉬의 크기를 S_V, 총 읽기 요청 횟수를 r, 총 쓰기 요청 횟수를 w, 캐쉬 적중 횟수를 h라 하자. 그러면, 전통적인 버퍼캐쉬 문제에서와 달리 캐쉬적중률, 즉 h/(r+w)는 비휘발성 버퍼캐쉬 환경에서 더 이상 좋은 성능 척도가 되지 못한다. 대신에, 이 환경에서 더 의미 있는 성능 척도는 스토리지 접근 횟수이다. 이는 비휘발성 버퍼캐쉬 환경의 경우 캐쉬 적중 상황에서도 스토리지 접근이 발생할 수 있기 때문이다. 예를 들어 쓰기 요청이 발생했고 해당 데이터 블록이 비휘발성 버퍼캐쉬가 아닌 휘발성 버퍼캐쉬에 존재한다고 하자. 이 경우 캐쉬 적중에 해당하지만 데이터를 비휘발성 버퍼캐쉬로 옮기는 작업이 필요하다. 이때, 비휘발성 버퍼캐쉬에 빈 공간이 없는 경우 방출할 블록을 선정한 후 이를 스토리지에 쓰는 작업을 선행해야 한다. 이는 캐쉬 적중 상황에서도 스토리지 접근이 필요할 수 있다는 것을 의미한다.

따라서, 비휘발성 버퍼캐쉬에서는 총 스토리지 접근 횟수를 성능 척도로 정의하는 것이 적절하다. 한편, 읽기 연산 시의 캐쉬 적중 발생 시에는 해당 데이터가 휘발성 버퍼캐쉬에 존재하든 비휘발성 버퍼캐쉬에 존재하든 스토리지 접근은 필요하지 않다.

Ⅲ. 제안하는 버퍼캐쉬 관리 기법

1. 시스템 아키텍쳐

그림 1은 제안하는 버퍼캐쉬의 메모리 및 스토리지 구조도이다. 2차 저장장치로 플래시메모리가 사용되고, 휘발성 버퍼캐쉬로는 DRAM, 비휘발성 버퍼캐쉬로는 PRAM과 같은 비휘발성메모리(NVRAM)를 사용한다. 모든 수정 데이터들은 NVRAM에 유지함으로써 전원이 나가더라도 데이터의 최신성을 유지할 수 있도록 한다. DRAM과 달리 NVRAM 버퍼캐쉬 자체가 비휘발성이므로 기존의 비휘발성 2차 저장장치인 플래시메모리에까지 쓰기 연산이 이어지는 경우는 NVRAM 용량이 부족하여 데이터가 쫓겨나는 경우로 한정되어 이로 인한 성능 향상을 기대할 수 있다.

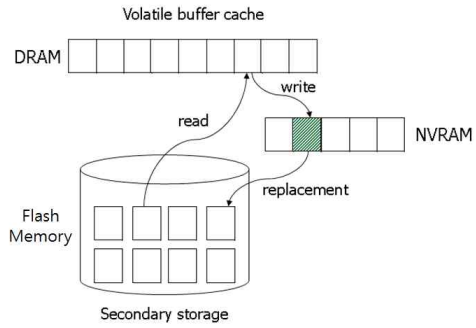

그림 1. 제안하는 버퍼캐쉬 및 스토리지 구조
Fig. 1. Proposed buffer cache and storage architectures

2. 제안하는 버퍼캐쉬 관리 알고리즘

제안하는 버퍼캐쉬 관리 알고리즘은 파일의 블록 접근 시 읽기 연산과 쓰기 연산의 기록을 별도로 유지해서 각 연산에 대한 미래 발생 가능성을 정교하게 예측한다. 즉, 최근의 읽기 연산 발생 기록에 근거한 LRR(least recently read) 리스트와 최근의 쓰기 연산 발생 기록에 근거한 LRW(least recently written) 리스트를 두어 미래의 읽기 쓰기 연산 발생 여부를 정교하게 예측한다.

이 두 리스트에 있는 블록들은 실제 데이터가 아니라 메타데이터, 즉 과거 접근 기록 및 블록의 식별자만 관리되는 것이기 때문에 휘발성 및 비휘발성 버퍼캐쉬에 있는 실제 데이터와 일치할 필요는 없다. 즉, 제안하는 알고리즘은 메타 데이터와 실 데이터를 분리해서 관리함으로써 각각의 버퍼캐쉬 공간을 효율적으로 활용할 수 있게 한다.

예를 들어 최근 읽기와 쓰기가 동시에 발생한 블록이 있다면, 그 블록의 메타데이터는 LRR 리스트나 LRW 리스트에 모두 존재할 수 있지만, 실 데이터는 비휘발성 버퍼캐쉬에만 존재한다. 따라서, LRR 리스트에 있는 일부 블록들은 휘발성메모리에 존재하지 않을 수도 있다.

이로 인해 휘발성 버퍼캐쉬가 더 많은 블록들을 저장할 수 있어 더 높은 캐쉬 적중률을 기대할 수 있다. 휘발성 버퍼캐쉬에 없는 데이터에 대한 읽기 기록을 LRR 리스트에 유지하는 이유는 비휘발성 버퍼캐쉬에서 공간 부족으로 해당 데이터가 방출될 경우 그 블록이 다시 휘발성 버퍼캐쉬로 돌아와야 하는지를 읽기 기록을 토대로 결정할 수 있도록 하기 위한 것이다. 그림 3은 제안하는 알고리즘의 의사코드를 나타낸 것이다.

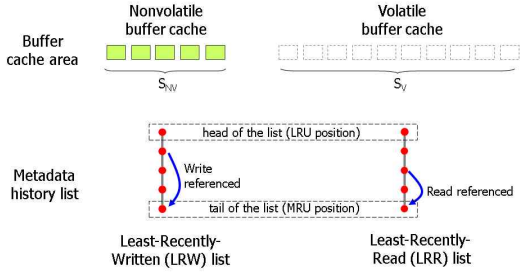

그림 2. 제안하는 알고리즘의 구현을 위한 자료구조
Fig. 2. Data structure for the proposed algorithm

Ⅳ. 성능 평가

제안하는 버퍼캐쉬 관리기법(NBM: nonvolatile buffer management)의 성능 검증을 위해 트레이스 기반 시뮬레이션 실험을 수행하였다. 실험에 사용한 트레이스는 UC 버클리대학의 NOW 프로젝트에서 제공하는 공개 트레이스이며 구체적인 내용은 표 1과 같다[7].

성능 비교를 위해, 기본 LRU 알고리즘을 비휘발성 버퍼캐쉬의 특성에 맞게 수정한 NVLRU라는 알고리즘을 비교대상으로 사용하였다. NVLRU는 NVRAM에 수정 블록을 유지시킴으로써 크래쉬 발생시에도 데이터의 최신 상태를 유지할 수 있다는 점에서 NBM과 마찬가지로 비휘발성 버퍼캐쉬의 특성을 적절히 활용한다. 다만, 새로운 수정 블록의 진입을 위해 NVRAM에 빈 공간이 없는 경우 LRU 방식으로 블록을 방출한다는 점에서 쓰기연산의 기록을 별도로 활용하는 NBM과 차별화된다.

그림 4는 NBM과 NVLRU의 성능을 총 입출력횟수로 비교해서 보여준다. 그림에서 보는 것처럼 NBM이 모든 경우에서 NVLRU에 비해 우수한 성능을 나타내는 것을 확인할 수 있다. 특히, 휘발성메모리(VRAM)의 용량이 일정 수준 이상인 상황(즉, 실제 상황에 좀 더 유사한 환경 설정)에서 제안하는 기법이 NVLRU에 비해 더 큰 성능 향상을 나타내는 것을 확인할 수 있다. 참고로 NVLRU 역시 본 논문이 제안하는 비휘발성 버퍼캐쉬를 사용하는 시스템이지만 알고리즘 설계가 NBM에 비해 잘 설계되지 않은 설정이다.

한편, NVLRU는 최근 읽기기록은 있으나 쓰기기록이 거의 없는 블록도 NVRAM에 유지한다. 이는 NVLRU가 읽기와 쓰기 연산 기록을 별도로 유지하지 않기 때문에 발생하는 현상이다. 즉, 비휘발성메모리에 쓰기연산과 읽기연산의 구분없이 모든 과거 접근 기록을 LRU로 통합하여 관리하고 있기 때문이다. 또한, NBM은 휘발성 버퍼캐쉬의 크기가 충분히 클 때 성능 개선 효과가 더욱 뚜렷하게 나타낸다. 이는 NVRAM에서 캐쉬 삭제가 필요

```
procedure NBM (block p, operation op)
{  if(p is in the buffer cache) /* cache hit */
   {  if(op is read)
      { if(p∉ read_list) add p at tail of read_list;
        else move p to tail of read_list;
      }
      else    /* write */
      { if(p∉ write_list) add p at tail of write_list;
        else move p to tail of write_list;
        if(p is not in NVRAM)
        {  if(no free block in NVRAM)
              replace_NVRAM ();
           move p from VRAM to NVRAM;
        }
      }
   }
   else /* cache miss */
   {  if(op is read)
      {  if(no free space in VRAM)
            replace_VRAM ();
         add p to VRAM;
         add p at tail of read_list;
      }
      else     /* write */
      {  if(no free space in NVRAM)
            replace_NVRAM ();
         add p to NVRAM;
         add p at tail of write_list;
      }
   }
}
procedure replace_VRAM ()
{  evict the least recently read block p in VRAM;
   evict the read history of p from read_list;
}
procedure replace_NVRAM ()
{  select the least recently written block p in
     NVRAM;
   evict the write history of p from write_list;
   if(p∈read_list)
   {  if(p is more recently read than the least
         recently read block q in VRAM)
      {  evict q from VRAM;
         evict read history of q from read_list;
         move p from NVRAM to VRAM;
      }
   }
   else evict p from NVRAM;
}
```

그림 3. 제안하는 알고리즘의 의사 코드
Fig. 3. pseudo-code of the proposed algorithm

표 1. 실험에 사용한 트레이스의 특성
Table 1. Trace characteristics used in the experiments

	trace	INS	RES
System calls	Total	861168	393571
	Reads	733549	336963
	Writes	127619	56608
	Read:Write	5.75	5.95
Memory usage	VRAM	106MB	68MB
	NVRAM	152MB	133MB

한 경우 NBM이 최근에 읽기 연산이 발생한 데이터를 캐쉬에서 지워버리지 않고 DRAM으로 이동시켜 성능 개선을 도모할 수 있기 때문이다. 이는 NVRAM 공간을 적게 사용하면서 최근에 읽기연산이 발생한 모든 데이터를 버퍼캐쉬에 보관할 수 있도록 한다. 또한, 휘발성 버퍼캐쉬의 용량 압박이 비휘발성 버퍼캐쉬에 비해 더 극심한 상황에 도달할 경우 제안하는 기법은 읽기 연산이 발생할 가능성이 높은 데이터를 NVRAM에서 DRAM으로 이주시켜 더 높은 읽기 캐쉬 적중률을 유도할 수 있게 되는 것이다.

그림 5는 제안하는 기법과 NVLRU의 읽기입출력 횟수를 보여 준다. 그림에서 보는 것처럼 NVLRU에 비해 NBM의 읽기 입출력 횟수가 크게 줄어드는 것을 확인할 수 있으며, 이는 NVRAM 용량이 적은 경우 더욱 뚜렷이 나타나는 것을 확인할 수 있다. 그림 6은 제안하는 기법과 NVLRU의 쓰기 입출력 횟수를 보여주고 있다. 그림에서 보는 것처럼 모든 상황에서 두 알고리즘의 쓰기 횟수가 거의 동일하게 발생한 것을 확인할 수 있다. NBM은 비휘발성 버퍼캐쉬에서 휘발성 버퍼캐쉬로 수정 블록을 이동시킬 경우 데이터의 수정본을 스토리지에 반영하는 작업을 수행하게 된다. 그러나 이러한 작업은 NVLRU에서도 수행해야 하는 작업이므로 두 방법 사이의 쓰기 입출력 횟수는 크게 차이가 나지 않는 것을 확인할 수 있다.

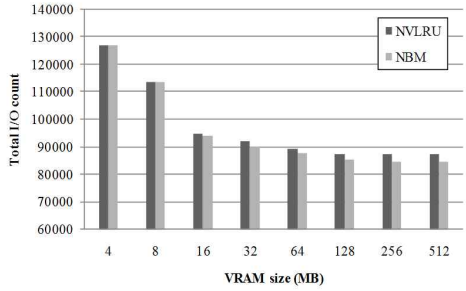

그림 4. 제안하는 기법의 성능 분석
Fig. 4. performance analysis of the proposed scheme

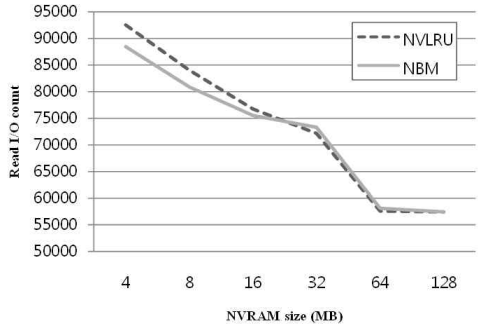

그림 5. 제안하는 기법의 읽기 성능
Fig. 5. read performance of the proposed scheme

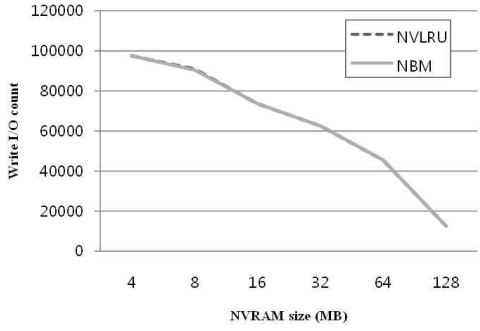

그림 6. 제안하는 기법의 쓰기 성능
Fig. 6. write performance of the proposed scheme

V. 결 론

본 논문은 스마트폰의 고성능화를 위해 휘발성메모리와 비휘발성메모리를 동시에 버퍼캐쉬로 사용하는 NBM이라는 알고리즘을 제안하였다. 제안하는 기법은 크래쉬 발생 시에도 데이터의 최신성을 보장하기 위해 수정 블록을 비휘발성메모리에 보관하여 스토리지 접근 횟수를 크게 줄이는 동시에, 읽기 기록과 쓰기 기록을 별도로 유지하여 두 연산 모두 성능 저하가 발생하지 않도록 하는 것이 주요한 아이디어라 할 수 있다. 비휘발성 캐쉬에서 삭제될 데이터를 선정할 때 제안하는 방법은 최근 쓰기 연산이 발생하지 않은 블록을 선정하고 삭제 대상인 데이터가 빈번한 읽기 연산을 유발하는 데이터일 경우 이

를 휘발성 버퍼캐쉬로 이주시킬 수 있도록 설계하였다. 트레이스 기반 시뮬레이션을 통해 제안한 기법이 NVLRU에 비해 입출력 횟수와 읽기 성능이 개선되는 것을 확인할 수 있었다.

References

[1] D. Kim and H. Bahn, "Buffer Cache Management of Smartphones Exploiting Write-Only-Once Characteristics," The Journal of The Institute of Internet, Broadcasting and Communication (JIIBC), Vol. 15, No. 6, pp. 129-134, 2015.

[2] 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.

[3] E. Lee et al., "Unioning of the Buffer Cache and Journaling Layers with Non-volatile Memory," Proc. USENIX FAST Conference, pp. 73-80, 2013.

[4] E. Lee et al., "Performance Evaluation and Analysis of NVM Storage for Ultra-Light Internet of Things," The Journal of The Institute of Internet, Broadcasting and Communication (JIIBC), Vol. 15, No. 6, pp. 181-186, 2015.

[5] E. Lee et al., "Eliminating Periodic Flush Overhead of File I/O with Non-volatile Buffer Cache," IEEE Trans. Computers, vol. 65, no. 4, pp. 1145-1157, 2016.

[6] L. Belady, "A Study of Replacement of Algorithms for a Virtual Storage Computer," IBM Systems Journal, vol.5, no.2, pp.78 - 101, 1966.

[7] D. Roselli et al., "A Comparison of File System Workloads," Proc. USENIX Annual Technical Conference, pp. 41-54, 2000.

저자 소개

최 현 경(준회원)

• 2008년 2월 : 강원대학교 컴퓨터공학과 학사
• 2013년 9월 ~ : 이화여자대학교 컴퓨터공학과 대학원생

<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 산업통상자원부 및 한국산업기술평가관리원의 산업원천기술개발사업(No.10041608)의 지원으로 수행됨.