논문 2019-14-12

사용자 응답성 향상을 위한 멀티큐 블록계층 개선

(Improvement of Multi-Queue Block Layer for Fast User Response)

신 희 영, 김 태 석*
(Heeyoung Shin, Taeseok Kim)

Abstract : Multi-queue I/O block layer has been recently employed in Linux kernel to support fast storage devices such as NVMe SSDs, but it lacks differentiated I/O services yet. In this paper, we propose an I/O scheduling scheme that can improve the user responsiveness of foreground processes, which are closely related to user satisfaction. To this end, we redesign the existing multi-queue block layer to classify the I/O requests from foreground processes and schedule them by exploiting the feature of NVMe interface. Experimental results show that latency and launch time of the foreground processes have been significantly improved compared to original Linux kernel.

Keywords : NVMe SSDs, User responsiveness, Block I/O layer, IO scheduler

Ⅰ. 서 론

NVMe (Non-Volatile Memory Express)는 기존 SATA보다 훨씬 빠른 PCIe로 연결된 비휘발성 저장장치를 위한 통신규약으로, 기존 SATA와 AHCI 조합으로 제 속도를 내기 어려운 SSD 장치를 위해 현재 널리 사용되고 있다 [1-3]. 하나의 명령큐를 가지는 AHCI와는 달리 NVMe는 최대 65,535개의 큐를 가질 수 있고, 큐 하나 당 다시 65,536개의 명령을 처리하는 높은 병렬성을 통해 고속의 입출력 서비스를 지원한다 [4].

이러한 NVMe SSD를 효율적으로 지원하기 위하여 운영체제 또한 변화하고 있고, 최근 리눅스 커널에서도 멀티큐 블록 계층 (multiblock queue layer)을 도입하였다 [5]. 멀티큐 블록 계층은 싱글 큐 구조로 인해 멀티코어 CPU 환경에서 존재하던 락경쟁 문제를 해결하기 위해 코어 당 하나씩 소프트웨어큐를 두는 한편, SSD와 같이 병렬성을 제공하는 저장장치를 위해 여러 하드웨어큐를 사용한다. 이는 단위시간당 처리하는 입출력 수를 크게 증가시킬 수 있어 멀티코어와 NVMe SSD 장치를 사용하는 컴퓨팅 환경에 적합하다.

본 논문에서는 이러한 멀티큐 블록 계층을 개선해 높은 입출력 대역폭과 IOPS (Input/Output Operations Per Second)를 그대로 유지하면서 사용자 응답성까지 향상시키는 기법을 제안한다. 특히 다량의 파일복사, 대용량 파일 다운로드, 프로그램 업데이트 등 후위 (background)에서 다수의 프로세스들이 실행되어 시스템에 부하를 주고 있는 상황에서도 사용자가 직접 사용하고 있는 프로세스가 영향을 받지 않게 하여 사용자의 만족도를 향상시키고자 한다. 이를 위해 사용자가 작업 중인 전위 (foreground) 프로세스를 식별하여 CPU 우선순위를 임시로 높이고, 우선순위 정보를 멀티큐 블록 계층까지 전달하여 차별화된 입출력 서비스를 제공한다. 또한 NVMe SSD에서 라운드로빈 방식으로 입출력 요청을 디스패치하는 사실을 활용해 입출력 요청을 가급적 대기가 적은 큐에 배정하여 응답성을 높이고자 한다. 이러한 기법을 리눅스 커널에 구현하여 입출력 처리시간 개선과 응용프로그램 실행시간 개선 여부를 측정하였다.

본 논문은 다음과 같이 구성되어 있다. 2장에서는 NVMe SSD의 동작방식과 멀티큐 블록 계층에 대한 배경지식과 관련연구를 살펴본다. 3장에서는 본 논문에서 제안하는 사용자 응답성 개선을 위한 차

*Corresponding Author (tskim@kw.ac.kr)
Received: Feb. 22, 2019, Revised: Mar. 19, 2019, Accepted: Apr. 01, 2019.
H. Shin, T. Kim: Kwangwoon University.
※ 이 성과는 2019년도 정부(미래창조과학부)의 재원으로 한국연구재단의 지원을 받아 수행된 연구임(No. 2017R1A2B4008536).

 2019 Apr. 14(2) 97-102
ISSN : 1975-5066
http://dx.doi.org/10.14372/IEMEK.2019.14.2.97

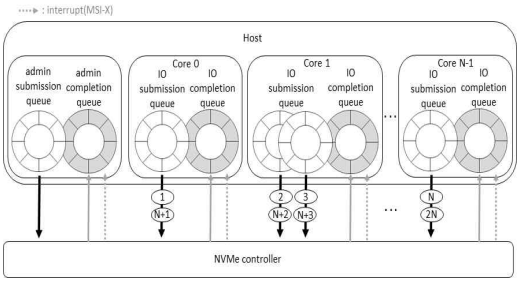


그림 1. 라운드로빈 방식의 명령어 디스패치
Fig. 1 Commands dispatched in round-robin manner

별화된 입출력 지원 기법에 대해 서술하고, 그 실험 결과를 4장에서 보인다. 마지막 5장에서는 연구 결과에 대한 요약과 함께 향후 연구 방안을 제시한다.

Ⅱ. 배경지식 및 관련 연구

1. NVMe의 동작방식

NVMe는 병렬성을 높이기 위해 멀티큐를 제공하는데, 다양한 디바이스 설정 명령어를 처리하는 관리큐 (admission queue)와 실제 입출력 명령어를 처리하는 입출력큐(I/O queue)로 크게 나누어진다. 이들은 다시 각각 호스트가 요청하는 명령어를 대기시키는 제출큐 (submission queue)와 컨트롤러가 명령어 처리 결과를 대기시키는 완료큐 (completion queue)의 조합으로 이루어지고 조합마다 ID가 부여된다. 관리큐는 NVMe 컨트롤러 당 하나의 조합이 생성되고 ID 0을 부여받는다. 입출력큐 조합은 코어 개수만큼 생성되는데, 입출력 제출큐는 디바이스가 허용하는 범위에 한해 추가로 생성할 수 있다.

NVMe 컨트롤러는 그림 1과 같이 다수의 큐로부터 하나씩 입출력 요청을 처리하는 라운드로빈 방식을 지원한다. 호스트는 제출큐에 입출력 요청을 삽입한 후 도어벨 레지스터를 업데이트 하고, 컨트롤러는 도어벨 레지스터가 업데이트된 제출큐를 순차적으로 방문하여 입출력 요청을 처리한다. NVMe가 가중치 기반 라운드로빈 (Weighted Round Robin)방식도 지원하는 것으로 되어 있지만, 아직 대부분의 NVMe SSD 장치들이 라운드로빈 방식을 구현하고 있다. 만일 NVMe SSD 장치가 가중치 기반 라운드로빈 방식을 지원한다면 입출력 제출큐에 우선순위를 부여하여 가중치 별로 입출력 요청을 차별화하여 처리할 수 있다. [6]에서는 NVMe 드라이버의 제출큐를 추가로 생성하고 각각의 제출큐에 우선순위를 부여하여 WRR 중재방식을 활용하여 입출력 요청의 우선순위별 서비스를 제공하였다.

2. 멀티큐 블록 계층

싱글코어 CPU 환경에서는 리눅스 커널의 블록 계층에 입출력 요청 큐가 하나여도 문제가 없었고, 하드디스크나 SATA방식의 SSD를 저장장치로 사용하는 환경에서도 디스패치큐 하나면 충분했다. 따라서 오랫동안 리눅스 커널에서는 한 개의 입출력 요청 큐와 한 개의 디스패치큐로 구성된 형태의 싱글큐 블록 계층을 사용해왔다. 그러나 최근 많은 컴퓨터 시스템에서 멀티코어 CPU와 NVMe SSD 저장장치를 사용함으로써 지금까지의 블록 계층 구조는 병목현상의 원인이 되었다.

이를 해결하기 위해 도입된 멀티큐 블록 계층은 소프트웨어큐와 하드웨어큐로 구성된다 [5]. 소프트웨어큐는 blk_mq_ctx라는 자료구조로 기술되고 CPU코어 개수만큼 생성된다. 각 코어에서 실행중인 스레드로부터의 입출력 요청은 각각 미리 할당된 소프트웨어큐로 삽입되므로, 동시에 여러 개의 스레드가 입출력 요청을 큐에 삽입하려 해도 병목현상이 발생되지 않는다. 하드웨어큐는 NVMe 드라이버 내의 제출큐와 연결되는 디스패치큐로, blk_mq_hw_ctx라는 자료구조로 기술된다. 각 큐에 대응되는 자료구조들이 초기화될 때 소프트웨어큐와 하드웨어큐를 연결하는 과정 또한 이루어진다. 최신 리눅스 커널은 싱글큐 블록 계층과 멀티큐 블록 계층을 모두 가지고 있어 NVMe SSD 뿐만 아니라 하드디스크와 SATA SSD도 함께 지원하고 있다.

[7]에서는 이러한 멀티큐 블록 계층 구조에서 읽기 성능을 저하시키는 쓰기 간섭 현상을 제거하기 위해 입출력 요청 큐를 읽기용과 쓰기용으로 분리하였다. 이와 유사하게 [8]에서도 읽기 요청과 쓰기 요청 비율을 적절한 비율로 처리하여 가비지 컬렉션이 집중적으로 수행되는 것을 방지함으로써 쓰기 간섭 현상을 완화하여 읽기 응답속도를 개선하였다. [9]는 멀티 SSD 볼륨 환경에서 멀티 블록 계층에서도 입출력 요청이 볼륨 단위로 처리되어 제성능을 발휘할 수 없음을 지적하고, SSD 장치 단위로 입출력 요청이 처리될 수 있도록 멀티큐 블록 계층을 수정하였다. 이를 위해 그들은 입출력 스택 구조를 볼륨 수준에서 들어오는 입출력 요청을 배치시키는 과정과 SSD 장치 수준으로 입출력 요청을 병렬화해서 처리하는 과정으로 구분하였다.

Ⅲ. 응답성 향상을 위한 멀티큐 블록 계층 개선

사용자 입장에서는 단위 시간당 처리하는 입출력 요청 수나 처리량보다 전위에서 사용 중인 프로세스의 입출력 지연시간이 더 중요하다 [10]. 여기에서는 사용자가 직접 사용하고 있는 전위 프로세스의 응답성 개선을 위한 기법을 다음과 같이 세 부분으로 나눠 기술한다.

먼저, 전위 프로세스의 응답성을 높이기 위해 먼저 전위 프로세스를 식별하여 그 우선순위를 임시로 높였다. 리눅스 커널에서는 CFS (Completely Fair Scheduler) 등 우선순위 기반의 CPU 스케줄링 기법이 사용되고 있으므로 이것만으로도 전위 프로세스의 응답선 개선을 기대할 수 있다. 사용자에게 투명성을 제공하기 위해서는 전위 프로세스를 스스로 식별하여 우선순위를 조정할 필요가 있으므로, 이를 위해 프로세스의 전위 및 후위 실행을 제어할 수 있는 쉘에 전위 프로세스를 식별하는 코드를 추가하였다. 프로그램이 처음 전위로 실행될 때 그리고 후위 프로세스가 전위 프로세스로 변경될 경우 임시로 높은 우선순위를 받을 수 있도록 하였고, 전위 프로세스가 후위 프로세스로 변경될 때는 원래의 우선순위로 환원되도록 하였다. 이를 위해 쉘의 해당 부분에 setpriority() 함수를 이용해 우선순위를 조정하였다.

이러한 우선순위는 CPU 스케줄러에서만 효과가 있는 것으로 아무리 전위 프로세스의 우선순위가 높아졌다하더라도 입출력 블록계층에서 이에 대한 지원이 적절히 이루어지지 않으면 전위 프로세스의 응답성 개선이 제한적일 수밖에 없다. 따라서 입출력 요청의 자료구조인 request에 우선순위 정보를 추가하여 입출력 블록 계층에서도 프로세스의 우선순위 정보를 활용할 수 있도록 하였다. 그런 다음 기존에 코어마다 하나씩 있던 소프트웨어큐 (sw q)를 그림 2와 같이 각각 전위 프로세스와 후위 프로세스를 위해 구분하여 후위 프로세스로부터 많은 입출력 요청이 들어온다 하더라도 전위 프로세스로부터의 입출력 요청을 우선적으로 처리할 수 있도록 하였다.

소프트웨어큐에서 대기중인 입출력 요청이 하드웨어큐 (hw q)를 거쳐 일단 NVMe의 제출큐 (sq)로 넘어가면 호스트 입장에서 더 이상 제어할 수가 없다. 즉, 제출큐에서의 스케줄링은 더 이상 불가능하므로 전위 프로세스의 입출력 요청을 여러 제출큐 중 어디로 보내느냐에 따라 지연시간이 달라질

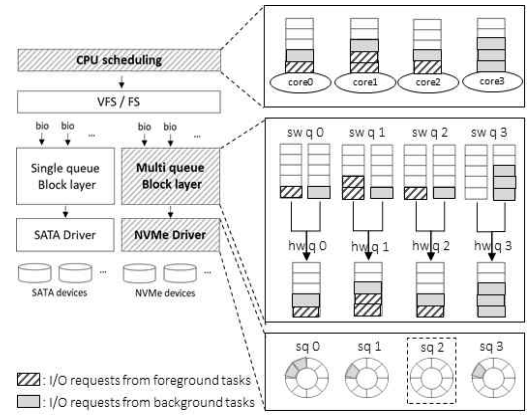


그림 2. 전위 프로세스의 응답성 향상
Fig. 2 Fast response of foreground processes

수 있다. 본 논문에서는 제출큐들을 라운드로빈 방식으로 번갈아가며 디스패치하는 NVMe 인터페이스의 특성상, 여러 제출큐 중 대기 중인 입출력 요청 수가 적은 곳으로 전위 프로세스의 입출력 요청을 보낸다. 그러면 NVMe 인터페이스가 라운드로빈으로 입출력 요청들을 디스패치할 때 전위 프로세스로부터 온 입출력 요청을 비교적 빨리 처리할 수 있으므로 전위 프로세스의 입출력 지연시간이 개선된다.

하드웨어큐는 제출큐와 일대일로 하나씩 연결되므로, 소프트웨어큐에서 하드웨어큐로 입출력 요청을 전달할 때 대기 중인 입출력 요청의 수가 가장 적은 제출큐에 대응되는 하드웨어큐로 전위 프로세스로부터의 입출력 요청을 넘긴다. 그림 2에서는 세 번째 제출큐 (sq 2)의 대기가 가장 짧기 때문에 두 번째 소프트웨어큐 (sw q1)에 있는 전위 프로세스의 입출력 요청을 세 번째 하드웨어큐 (hw q2)로 보낸다. 소프트웨어큐에서 하드웨어큐로 입출력 요청을 전달하는 과정에서 락경쟁 문제가 발생할 수도 있어 이를 위한 동기화가 필요하다. 또한, NVMe 드라이버에서 제출큐의 head나 tail의 위치를 알 수 있으므로, 각 제출큐에 대기중인 명령어 수를 파악할 수 있다. 그림 3은 제안하는 기법의 순서도를 나타낸다.

Ⅳ. 성능 평가

다수의 프로세스들을 후위에서 실행하여 시스템에 입출력 부하를 충분히 준 후, 전위 프로세스의 응답지연시간이 기존 커널 대비 얼마나 개선되지 확인하는 실험을 수행하였다. 이를 위해 입출력 요

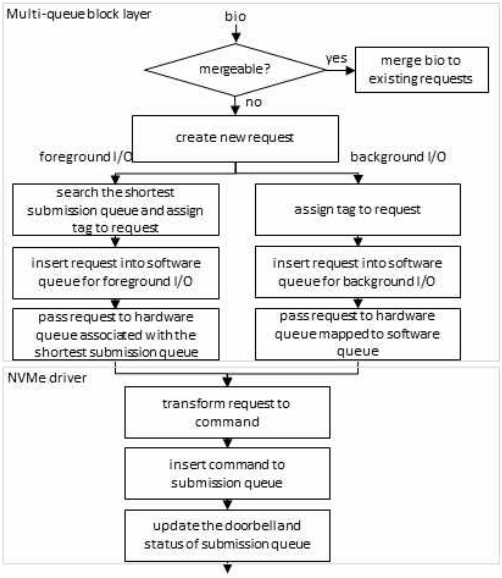


그림 3. 개선된 멀티큐 블록 계층의 순서도
Fig. 3 Flow chart of improved multi-queue block layer

청을 집중적으로 수행하는 fio 벤치마크를 50여개 실행하였고, 그 중 전위로 수행되는 프로세스의 성능을 측정하였다. fio는 다양한 설정이 가능한데, 그 설정내용과 실험에 사용된 환경은 각각 표 1, 표 2와 같다.

제안한 기법 중 세 가지 사항 즉, 전위 프로세스에 높은 우선순위 부여를 통한 CPU 스케줄링 효과, 전위 프로세스로부터 들어온 입출력 요청의 차별화를 통한 입출력 스케줄링 효과, 그리고 NVMe 인터페이스의 라운드로빈 디스패치방식을 고려한 제출 큐 정보 활용 효과가 각각 전위 프로세스의 응답성 개선에 어떠한 영향을 끼치는지 확인하기 위해 표 3과 같이 몇 가지 경우로 나누었다.

여러 실험 결과, 전위 프로세스의 성능은 CPU 스케줄링 효과만으로도 상당한 성능 개선이 되고, 이에 더하여 멀티큐 블록 계층에서 수정한 두 가지 스케줄링 기법도 효과가 있음을 입증하였다. 그림 4 (a)에서와 같이 전위 프로세스가 CPU 스케줄링의 효과 (CPU)만 얻어도 기존 커널 대비 약 9%의 성능 개선이 있었다. 여기에 전위 프로세스의 입출력 요청을 분리해 우선적으로 처리하는 경우 (CPU+IO) 약 11%의 성능 개선을 보였고, NVMe 드라이버 내 제출 큐 정보를 활용할 경우 (CPU+SIZE)에는 기존 커널 대비 약 12%의 성능

표 1. fio 워크로드 설정
Table 1. fio workload configurations

ioengine	libaio
size	15G
I/O pattern	random read
numjobs	512
iodepth	1

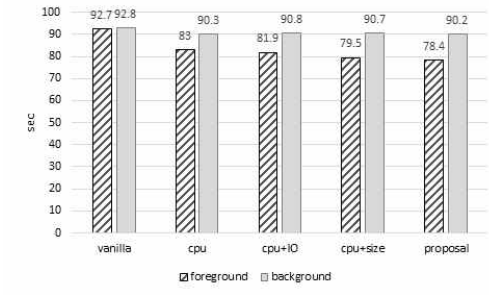
표 2. 실험 환경
Table 2. Experimental environment

CPU	Intel(R) Core(TM) i7-8700K CPU @ 3.70GHz
Storage	Samsung SSD 970 PRO 512GB
OS	Ubuntu 14.04 LTS 64bit
Kernel	Linux 4.13.10
Bash shell	4.4.18
fio	3.6

표 3. 성능 평가를 위한 비교 대상
Table 3. Options for performance evaluation of the proposed

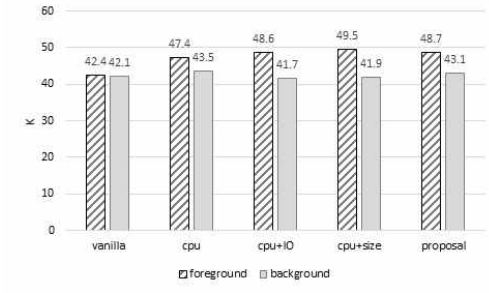
options	description
Vanilla	original kernel.
CPU	foreground tasks are given high priority.
CPU+IO	foreground tasks are given high priority and their I/O requests are differentiated.
CPU+SIZE	foreground tasks are given high priority and their I/O requests are passed to the shortest submission queue.
Proposed	foreground tasks are given high priority, their I/O requests are differentiated and passed to the shortest submission queue.

개선을 보였다. 그리고 세 가지 사항을 모두 적용했을 경우 (Proposed) 기존 커널 대비 약 15%의 성능이 개선된 것을 확인하였다. 즉, 시스템에 입출력 부하가 충분히 주어진 상황에서 CPU 스케줄러, 멀티큐 블록 계층, NVMe 인터페이스 전 계층에 걸쳐 전위 프로세스로부터의 입출력 요청 대기시간을 최소화하는 노력이 효과가 있음을 알 수 있었다. 후위에서 실행되는 프로세스들의 성능도 실험 결과, 크게 나빠지지 않거나 오히려 좋아지는 경우가 있었다. 이는 본 실험에서 전위 프로세스가 완료된 후 남은 자원들을 후위 프로세스들이 사용할 수 있었기 때문이다. 또한 IOPS와 입출력 대역폭 관점에서도 이와 유사한 결과를 보였다 (그림 4 (b), 4 (c)).



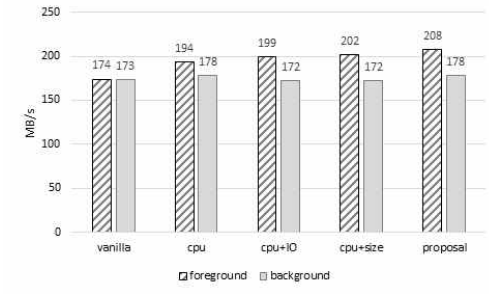
(a) 실행시간

(a) Execution time



(b) IOPS

(b) IOPS



(c) 입출력대역폭

(c) I/O bandwidth

그림 4. 전위 프로세스의 성능 개선

Fig. 4 Improved performances of foreground processes

프로그램의 구동시간 (launch time)도 사용자 입장에서는 매우 중요한 성능지표이므로, 다수의 fio를 후위에서 실행하면서 널리 사용되는 대표적인 응용프로그램들의 구동시간을 평가하였다. 각 응용프로그램들의 구동시간을 스톱워치로 10회씩 측정하여 그 평균값을 그림 5에 제시하였다. 브라우저 (firefox), 동영상 플레이어 (xdg), 그리고 리브레오피스 제품군 중 워드프로세서 (writer), 스프레드시트 (calc), 프리젠테이션 (impress) 등 총 5개의 응용프로그램을 대상으로 실험하였다. firefox에서는 로딩되는 항목이 많은 복잡한 웹페이지를 홈으로 설정하여 웹브라우저 구동시간을 측정하였고, 그 결과 기존 커널 대비 약 28%의 성능 개선 효과가 있음을 확인하였다. 또한 xdg의 구동시간은 기존 커널 대비 약 49%의 개선이 있었고, writer, calc, impress는 각각 기존 커널 대비 64%, 60%, 65% 가량 성능이 개선됨을 확인하였다.

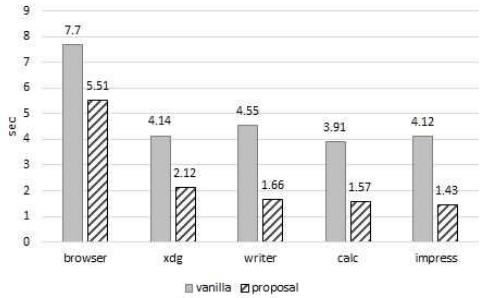


그림 5. 응용프로그램의 구동시간 개선

Fig. 5 Improved launch time of applications

Ⅴ. 결론 및 향후 연구

본 논문은 NVMe SSD를 사용하는 환경에서 사용자의 응답성 개선을 위한 복잡하지 않으면서 실용적인 기법을 제안하였다. 전위 프로세스를 식별한 후, 이 정보를 CPU 스케줄러뿐만 아니라 입출력 블록 계층까지 전달하여 우선적인 서비스를 받도록 하였다. 이를 위해 멀티큐 블록 계층에서 전위 프로세스의 입출력 요청을 분류하여 차별화하는 한편, 라운드로빈 방식으로 입출력 요청을 디스패치하는 NVMe 인터페이스의 특성을 이용해 전위 프로세스의 입출력 요청 의 디스패치시간을 최소화하였다. 입출력 부하를 충분히 주어 실험한 결과, 전위 프로세스의 실행시간 및 구동시간 등이 기존 커널 대비 크게 개선되었음을 확인하였다. 사용자 응답성은 모바일 환경에서 보다 중요하므로 향후 안드로이드 등의 모바일 플랫폼에도 적용, 연구할 예정이다.

References

[1] H. Kim, Y. Lee, J. Kim, “NVMeDirect: A User-space I/O Framework for Application-specific Optimization on NVMe SSDs,” Proceedings of

8th USENIX Workshop on Hot Topics in Stor age and File Systems, pp. 1-5, 2016.
[2] Q. Xu, H. Siyamwala, M,. Ghosh, T. Suri, M. Awasthi, Z. Guz, V. Balakrishnan, "Performance Analysis of NVMe SSDs and Their Implication on Real World Databases," Proceedings of the 8th ACM International Systems and Storage Conference, pp. 5, 2015.
[3] M. Bjørling, J. Gonzales, P. Bonnet, "LightNVM: The Linux Open-Channel SSD Subsystem," Proceedings of 15th USENIX Conference on File and Storage Technologies, pp. 359-374, 2017.
[4] K. Marks, "An NVM Express Tutorial," Flash Memory Summit, 2013.
[5] M. Bjørling, J. Axboe, D. Nellansm, P. Bonnet, "Linux Block IO:Introducing Multi-queue SSD Access on Multi-core Systems," Proceedings of the 6th International Systems and Storage Conference, pp. 22, ,2013.
[6] K. Josh, K. Yadav, P. Choudhary, i, "Enabling NVMe WRR Support in Linux Block Layer," Proceedings of the 9th USENIX Workshop on Hot Topics in Storage and File Systems, 2017.
[7] M. Lee, D. Kang, Y. Eom, "Utilizing Multi-queue Based NVMe SSD for Improving Read I/O Performance," Proceedings of Korea Computer Congress, pp. 1469-1471, 2016 (in Korean).
[8] J. Park, J. Lee, D. Seo, "Multi-Queue Block I/O Scheme for Improving User Responsivness in NVMe SSD," Proceedings of the Korean Institute of Communications and Information Sciences, pp. 1639-1640, 2017 (in Korean).
[9] P. Kumar, H.H. Huang, "Falcon: Scaling IO Performance in Multi-SSD Volumes," Proceedgins of the USENIX Annual Technical Conference, pp. 41-53, 2017.
[10] S. Hahn, S. Lee, I. Yee, D. Ryu, J. Kim, "Improving User Experience of Android Smartphones Using Foreground App-Aware I/O Management," Proceedings of the 8th Asia-Pacific Workshop on Systems, pp. 5, 2017.

Heeyoung Shin (신 희 영)

She received the B.S. degree in information and communications engineering from Hankuk University of Foreign Studies, Korea, in 2016. She is currently a M.S. student in the department of embedded software engineering, Kwangwoon University, Seoul, Korea. Her research interests include storage systems and embedded systems.

Email: hyshin@kw.ac.kr

Taeseok Kim (김 태 석)

He received the B.S. degree in computer science and the M.S. and Ph.D degrees in computer science and engineering from Seoul National University, South Korea, in 2000, 2002, and 2007, respectively. In 2007, he was a Senior Research Engineer with Samsung Electronics, Suwon, South Korea. In 2008, he joined the Department of Computer Engineering, Kwangwoon Univerisity, Seoul, South Korea, where he is currently a Professor. His research interests are primarily in computer systems such as operating systems, storage systems, multimedia systems, and embedded systems.

Email: tskim@kw.ac.kr