논문 2017-12-16

NVMe 드라이버 구현 방식에 따른 I/O 응답시간 분석

(Analysis of I/O Response Time Throughout NVMe Driver Implementation Architectures)

강 인 구, 주 용 수, 임 성 수*
(Ingu Kang, Yongsoo Joo, Sung-Soo Lim)

Abstract: In recent years, non-volatile memory express (NVMe), a new host controller interface standard, has been adapted to overcome performance bottlenecks caused by the acceleration of solid state drives (SSD). Recently, performance breakthrough cases over AHCI based SATA SSDs by adapting NVMe based PCI Express (PCIe) SSD to servers and PCs have been reported. Furthermore, replacing legacy eMMC-flash storage with NVMe based storage is also considered for next generation of mobile devices such as smartphones. The Linux kernel includes drivers for NVMe support, and as the kernel version increases, the implementation of the NVMe driver code has changed. However, mobile devices are often equipped with older versions of Android operating systems (OSes), where the newest features of NVMe drivers are not available. Therefore, different features of different NVMe driver implementations are not well evaluated on Android OSes. In this paper, we analyze the response time of the NVMe driver for various Linux kernel version.

Keywords : NVMe, Storage stack, Latency, Linux

Ⅰ. 서 론

현대 컴퓨터 시스템이 요구하는 입출력 성능을 만족하기 위해 플래시 메모리 기반 SSD (solid state drive)가 기존 HDD (hard disk drive)를 대체하게 되었고, 빅데이터, 기계학습 등의 새로운 애플리케이션이 요구하는 높은 수준의 입출력 성능을 제공하기 위해 플래시 SSD 자체의 성능도 지속적으로 개선되어 왔다. 최근에는 플래시 SSD 자체의 입출력 성능이 기존 AHCI (advanced host controller interface)/SATA (serial advanced technology attachment) 기반 입출력 인터페이스 성능을 추월하는 정도에 이르러 컴퓨터 시스템의 성능 향상을 방해하는 병목현상의 원인으로 입출력 인터페이스가 지목되는 상황을 맞이하게 되었다. 따라서 최신의 플래시 SSD 성능을 제대로 활용할 수 있도록 하기 위해 PCIe 기반의 차세대 입출력 인터페이스 표준인 NVMe (non-volatile memory express) 기술이 발표되었고 NVMe 기반의 SSD가 시장에 출시되기 시작하였다. NVMe SSD는 서버, 데스크톱 및 랩톱 PC에 도입되어 기존의 SSD보다 높은 성능을 확보할 수 있음이 입증되었다 [2].

NVMe는 기존의 ACHI 인터페이스에 비해 내부 구조 및 구현이 단순하고 프로토콜 오버헤드가 작은

*Corresponding Author (sslim@kookmin.ac.kr)
Received: May 12 2017, Revised: May 23 2017, Accepted: May 26 2017.
I. Kang, Y. Joo, S. Lim: Kookmin University
※ 이 논문은 2015년도 정부 (교육부)의 재원으로 한국연구재단의 지원을 받아 수행된 기초연구사업임 (No. 2015R1D1A1A0105831).
※ 본 연구는 미래창조과학부 및 정보통신기술진흥센터의 대학ICT연구센터육성 지원사업의 연구결과로 수행되었음 (IITP-2015-0-00363).
※ 본 연구는 미래창조과학부 및 정보통신기술진흥센터의 SW중심대학지원 사업의 연구결과로 수행되었음 (R7116-16-1031).
※ 이 논문은 2016 한국정보과학회 제43회 정기총회 및 동계학술발표회에서 “NVMe 드라이버 구현 방식에 따른 I/O 응답시간 분석”이라는 제목으로 발표된 논문 [1]을 확장한 것임.

ⓒ IEMEK J. Embed. Sys. Appl. 2017 June 12(3) 139-147
ISSN : 1975-5066
http://dx.doi.org/10.14372/IEMEK.2017.12.3.139

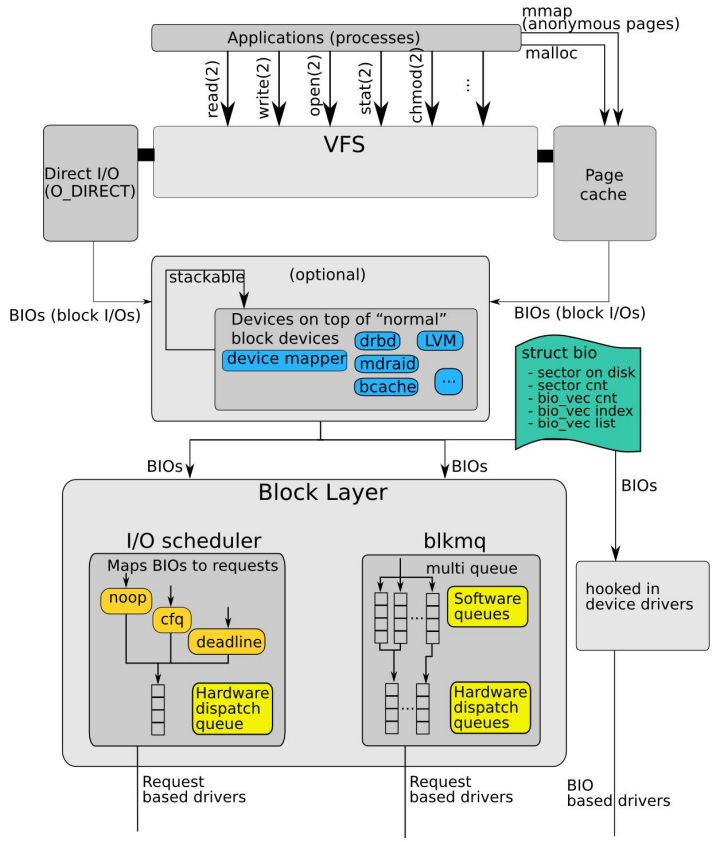

그림 1. 리눅스 스토리지 계층의 구조 ([5]의 다이어그램을 수정함)

Fig. 1 Architecture of Linux storage stack (Modified version of diagram from [5])

장점 [2]을 가지고 있어 서버 및 데스크탑 환경뿐만 아니라 모바일 기기에서도 기존의 eMMC를 대체하는 차세대 입출력 인터페이스로 고려되고 있으며 일부 스마트폰에 도입되기 시작하였다 [3]. 대표적인 모바일용 운영체제인 안드로이드는 리눅스 커널을 기반으로 하는데, 리눅스 커널 버전 3.3부터 NVMe 드라이버 구현이 포함되어 배포되고 있다.

NVMe 드라이버의 최초 설계에서는 기존의 리눅스 블록 레이어가 가지고 있는 한계 및 단점을 피하기 위해 블록 I/O 요청이 리눅스 블록 레이어를 우회하여 별도의 NVMe 드라이버에로 직접 전달되어 처리되는 방식으로 구현되었다 [2]. 이후 기존의 리눅스 블록 레이어의 단점을 개선하여 플래시 기반 차세대 저장장치에 적합한 구조인 Block Multi-Queue (blk-mq) [4]가 적용되었고, NVMe 드라이버 또한 blk-mq를 이용하는 방식으로 재작성되었다. 이후에도 커널 버전이 올라감에 따라 NVMe 드라이버도 지속적으로 변경 및 개선이 되고 있다.

하지만 최신 커널 버전 도입에 대해 보수적인 입장을 취하고 있는 안드로이드 운영체제에는 이러한 NVMe 드라이버 코드 개선 사항이 적시에 반영되지 못하고 있다. 그동안의 드라이버 코드 변경점은 데스크톱 및 서버 시스템에 맞추어 작성된 것이므로 최신 버전의 커널 드라이버를 백포팅하는 것이 모바일 장치에서도 더 좋은 특성을 나타낼 지에 대해서는 면밀한 분석이 필요하다. 따라서 본 연구는 NVMe 드라이버의 구현 변화에 따른 I/O 응답시간 관점에서의 성능 차이를 분석하고자 한다.

본 논문의 구성은 다음과 같다. 2장에서는 배경지식인 리눅스 스토리지 계층과 block-MQ에 대하여 설명하고 3장에서는 NVMe I/O 계층 구현의 변경점을 분석한다. 4장에서는 응답시간 측정 방법과 사용된 도구를 설명하며 5장에서는 측정 결과를

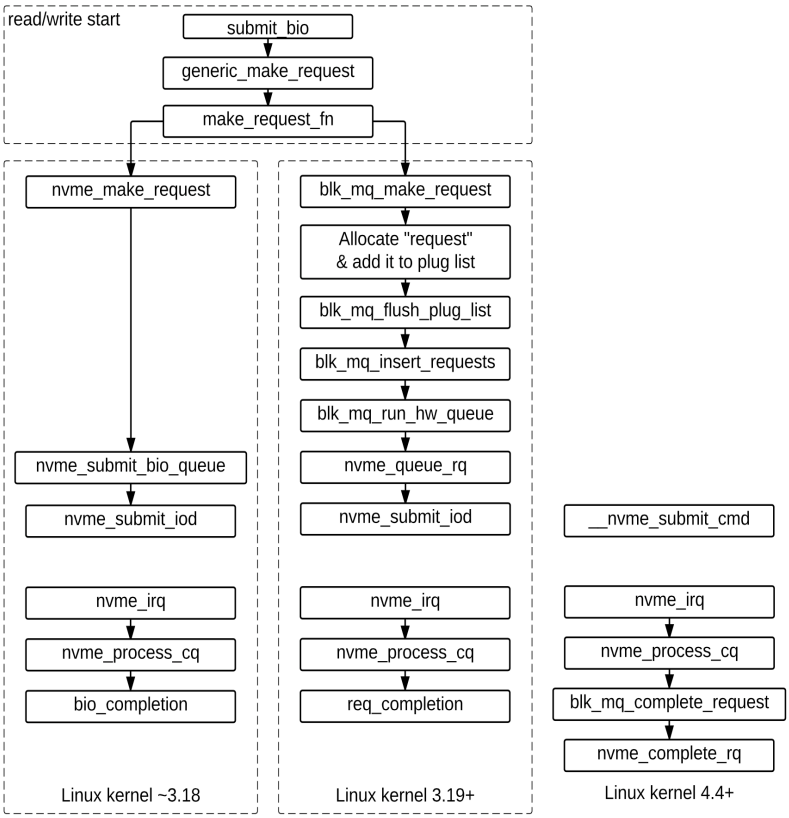

그림 2. 커널 버전 별 NVMe I/O 처리 함수 흐름
Fig. 2 NVMe I/O procsssing function flow by kernel version

보이고 그 원인에 대해 논한다. 마지막으로 6장에서는 결론 및 향후 연구에 대하여 설명한다.

II. 배 경

1. 리눅스 스토리지 계층

리눅스 시스템에서 I/O 요청은 사용자 애플리케이션 수준에서 발생해서 커널을 거쳐 디바이스로 전달된다. 커널 수준에서 처리되는 단계는 가상 파일시스템, 블록 레이어, 디바이스 드라이버로 크게 세 단계로 구분할 수 있다. 블록 레이어는 과거부터 대부분의 장치를 위해 사용되어 온 I/O 스케줄러를 통해 처리되는 경로와 최근에 추가된 blk-mq 방식의 경로 (그림 1)가 있다.

2. Block-MQ

blk-mq [4]는 최근에 만들어지고 적용된 블록 레이어의 부분으로, 기존의 블록 레이어에서 I/O 요청들이 I/O 스케줄러를 통해 한 개의 엘리베이터 큐로 모아지고 저장장치로 보내지는 과정으로 인한 병목 현상을 제거하였다.

blk-mq에는 소프트웨어 큐와 하드웨어 디스패치 큐로 구성된 두 단계가 존재하며 사용자 수준에서 발생한 I/O 요청들은 소프트웨어 큐로 삽입된다. 요청들은 드라이버로 요청을 전달하기 위한 하드웨어 디스패치 큐로 옮겨지며, 드라이버가 요청들을 처리하여 하드웨어 장치로 명령을 전송한다. 하드웨어 디스패치 큐는 저장 장치가 지원하는 최대 개수에 맞추어 만들어져 동작한다.

III. NVMe I/O 계층 구현 변경점

1. Block-MQ 도입 전후 주요 차이점

Block-MQ 도입을 전후로 한 Linux 버전에 따른

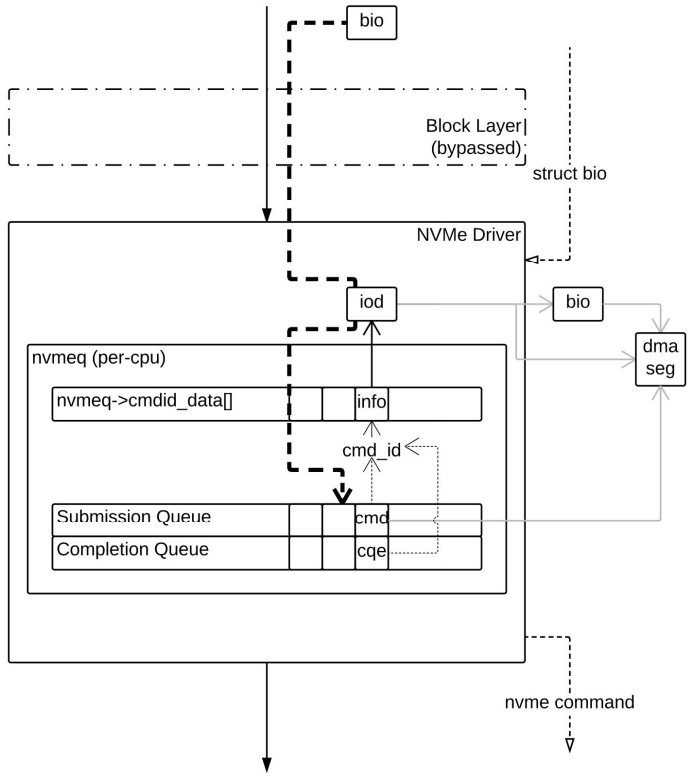

그림 3. Linux 3.18 이하에서의 NVMe I/O 요청 처리
Fig. 3 NVMe I/O request processing in Linux ≤3.18

NVMe 드라이버 구현의 주요 변화는 다음과 같이 정리할 수 있다.

- Linux 3.18 이전: 초기 NVMe 드라이버는 블록 레이어를 우회하여 bio 구조체를 직접 처리한다.
- Linux 3.19 이상: bio가 blk-mq를 거쳐 변환된 request 구조체로 변환되어 처리되도록 부분적으로 재작성되었다. 요청 정보를 저장하기 위한 cmd_info, iod 구조체 중 cmd_info는 array로 사전에 할당 된 pool에서 가져오며, iod는 필요에 따라 동적으로 할당된다.
- Linux 4.4 이상: 코드가 최적화 및 오류 수정을 위해 수정된 부분이 있으며, cmd_info와 iod 구조체가 통합되었고 blk-mq의 PDU (프로토콜 데이터 유닛)으로 사전 할당된다.

위에 정리된 구현 변화에 따른 함수 호출 흐름을 비교해 보면 그림 2와 같다.

다음 절들에서 더 상세히 설명할 I/O 계층 구현 구조의 차이점 중에서 성능에 영향을 줄 만한 부분을 간략히 살펴보면 리눅스 커널 3.18 이전에서 bio를 직접 처리하는 구조일 때 각 bio 처리를 위한 NVMe 드라이버의 메타데이터 저장 구조체를 매번 할당하였던 부분이 최신 커널로 오면서는 드라이버 초기 적재 시에 blk-mq의 API를 통하여 사전 할당됨으로써 개별 할당 작업 오버헤드가 감소한 점, 그리고 I/O 처리를 위한 request 구조체와 드라이버의 메타데이터 저장을 위한 PDU 영역의 메모리 공간상 근접성으로 인한 메모리 캐싱 효율 개선 가능성, 그리고 직접 관리하였던 큐 구조에서 blk-mq의 큐 관리를 사용하는 구조로 바뀌면서 CPU별 큐 처리 및 NVMe 장치에서 돌아온 응답에 대한 처리구조 개선 등이 있다고 사료된다.

2. Block-MQ 도입 전 NVMe Driver 구조

Linux 3.18 이하에서의 흐름 (그림 3)은 다음과 같다.

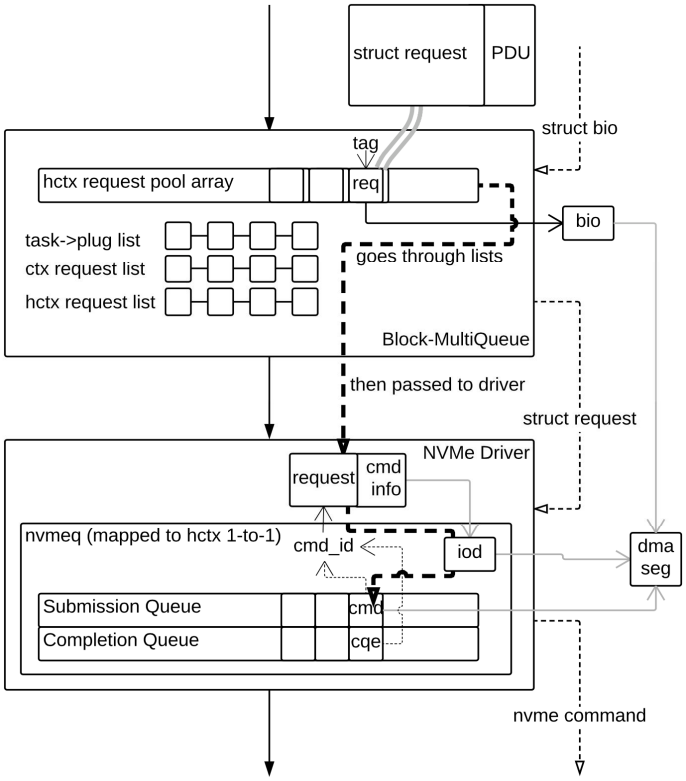

그림 4. Linux 3.19 이상에서의 NVMe I/O 요청 처리
Fig. 4 NVMe I/O request processing in Linux ≥ 3.19

1. nvme_make_request() 함수가 bio를 받아서 iod (I/O descriptor) 구조체를 만든다. 각각의 bio마다 iod가 하나씩 할당된다.

2. nvme_submit_iod() 함수가 iod를 받고, 현재 CPU를 위한 nvmeq 구조체로부터 이 iod를 위한 nvme_cmd_info 구조체 메모리 영역과 그 index를 할당받는다.

3. nvme_submit_iod() 함수에서 iod의 정보를 바탕으로 SQ (submission queue)에 명령어를 구성한다. 그와 동시에 미리 할당된 nvme_cmd_info 구조체에 iod와 callback 함수의 정보를 저장한다. 여기 저장된 정보들은 CQ (completion queue) 처리시에 사용된다. 마지막으로 SQ의 도어벨을 통해 NVMe 장치에게 알려준다.

iod와 callback 함수로의 포인터는 cmd_info에 저장하는데, 그 이유는 SQ/CQ entry가 command_id만을 저장할 수 있고 iod나 callback 포인터와 같은 다른 정보는 저장할 수 없기 때문이다.

3. Block-MQ 도입 후 NVMe Driver 구조

Linux 3.19 이상에서 요청이 처리되는 흐름 (그림 4)은 다음과 같이 정리할 수 있다.

1. blk_mq_make_request()가 bio를 받아 이를 바탕으로 request 구조체를 만든다. request 구조체의 instance는 NVMe device 초기화 단계에서 미리 할당되었던 hctx의 instance pool로부터 선택된 것이다. request->tag 필드는 pool 배열에서 request instance가 위치한 index를 저장한다. tag의 값은 나중에 nvme_submit_iod()에서 command_id로도 사용된다. 이 request 인스턴스는 plug->mq_list에 추가된다.

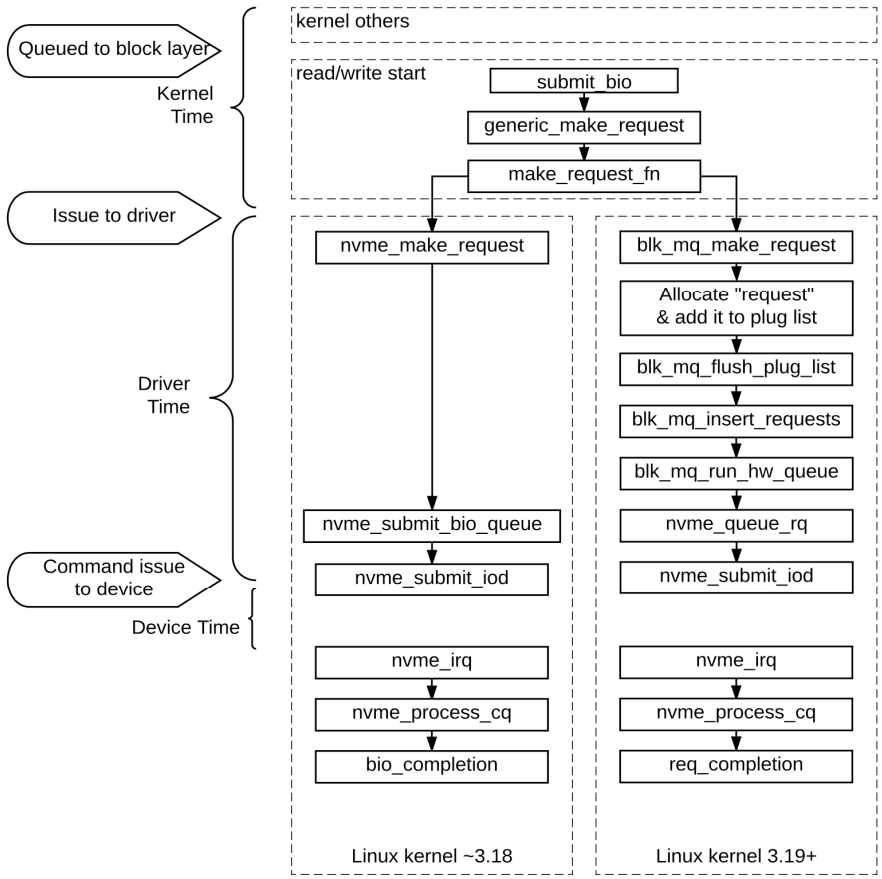

그림 5. 시간측정 구간과 이를 위한 트레이싱 지점
Fig. 5 Timing block and tracing points

2. blk_mq_flush_plug_list()와 blk_mq_insert_requests() 함수가 plug->mq_list에 있는 request들을 ctx->rq_list로 flush한다. 여기서 ctx->rq_list는 per-CPU 소프트웨어 큐를 지칭한다.

3. flush_busy_ctxs()와 __blk_mq_insert_request() 함수가 ctx->rq_list로부터 지역변수로 선언된 rq_list (일대일 HW dispatch queue로 사용됨)로 request들을 flush하고, nvme_queue_rq() 함수를 호출하여 처리한다.

4. nvme_queue_rq() 함수가 request에 대응되는 iod를 할당하고 정보를 저장한다.

5. nvme_submit_iod() 함수가 iod를 command로 변환한다. request->tag가 command_id로 사용된다. Completion 단계에서 사용되기 위해서 콜백 함수와 iod에 대한 정보가 nvme_cmd_info에 저장된다. 그리고 나서 SQ 도어벨을 울린다.

nvme_cmd_info는 미리 할당되어 있으며 구조체 할당 영역을 가리키는 포인터는 blk_mq_rq_to_pdu() 함수에서 request 구조체 인스턴스 주소에 sizeof (struct request)만큼 memory offset 연산을 하여 계산된다.

nvme_cmd_info와 nvme_iod는 최신 버전 Linux kernel driver에서 하나로 합쳐졌다. 모든

iod들은 이전의 nvme_cmd_info처럼 장치 초기화 시에 미리 할당되며, DMA 메모리 세그먼트 정보를 저장하는 배열인 iod->sg는 부분은 미리 할당되지 않는다.

Ⅳ. 구현 변경에 따른 응답시간 비교

1. 측정 방법

FIO [6]를 통해 I/O 요청을 발생시킴과 동시에 응용프로그램 수준에서의 응답속도를 측정한다. FIO를 실행하는 동안에는 blktrace [7]를 사용하여 커널 수준에서 I/O가 처리되는 시간을 기록한다.

blktrace에서는 블록 레이어로 요청 진입, 장치 드라이버로 전달, 디바이스로 명령 전달이 이루어지는 시점을 기록한다. 이를 위하여 기존 커널 소스코드에 구현되어 있지 않는, 드라이버로의 전달 시점을 기록하기 위한 blktrace 추적 지점을 커널에 추가하고 로그 분석기를 작성하였다. 추적 지점과 시간 측정 구간은 그림 5와 같다.

그림 6에서와 같이 두 도구로부터 얻은 데이터를 가공하여 최종적인 실험 데이터를 FIO에서 측정된 응답시간과 blktrace 로그에서 계산된 커널 레벨에서 단계별 시간 측정값을 이용하여 유저 레벨, 커널 레벨의 소요 시간을 계산한다. 커널 레벨의 소요 시간은 드라이버가 사용한 시간과 드라이버 외에 블록 레이어 등에서 사용된 시간으로 나누어 계산한다.

2. 워크로드

응답시간을 측정하기 위한 I/O 요청은 FIO를 통하여 생성하였으며, 이를 위해 정의한 FIO 워크로드는 다음과 같다.

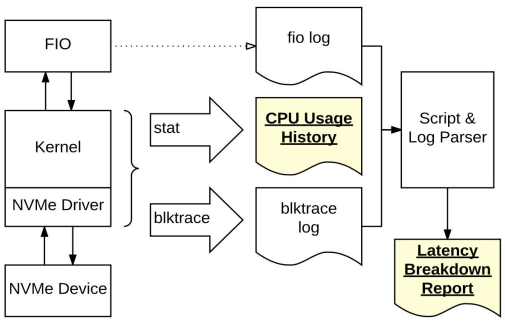

그림 6. FIO, blktrace를 통한 응답시간 측정
Fig. 6 Latency measurement using FIO, blktrace

표 1. 실험 환경
Table 1. Environment setup

Host Computer Hardware	
Target SSD	Intel 750 400GB
CPU/RAM	Intel Core i7-3770 / DDR3 16GB
Mainboard	ASUS P8H77-V LE
Boot Storage	HDD 250GB
Software Setup	
Ubuntu 14.04 LTS Server (minimal package)	
Linux Kernel 3.18, 4.2, 4.5 (from GitHub)	
FIO 2.1.3, blktrace 1.0.5 - main repo	
Go 1.6 - ppa:ubuntu-lxc/lxc-stable repo	

Low 워크로드는 동시에 1개만의 요청이 발생되는 워크로드이다. 이 워크로드는 가장 좋은 상황에서 코드 수행 시간을 측정하기 위한 것이다.

Normal 워크로드는 4-64KB 블록사이즈, 그리고 읽기/쓰기가 섞인 I/O 요청을 동시에 네 프로세스에서 큐 깊이 4로 생성한다. 이 워크로드는 일반적인 상황 [8, 9]에서 평균적인 코드 수행 시간과 그에 따른 응답시간이 어떻게 나오는지를 확인하여 동시다발적 요청이 잘 처리되는지를 보기 위한 것이다.

3. 실험 환경

본 연구의 실험 환경은 표 1과 같다.

Ⅴ. 측정 결과

Low 워크로드에 대한 측정 결과는 그림 7의 좌측 부분과 같다. 커널 버전 간의 차이가 2us 이내로 작았으며 경우에 따라서는 구 버전의 커널에서 오히려 더 짧은 처리 시간을 보이는 것을 알 수 있다.

이것은 I/O 요청의 수가 적은 상황에서 가장 좋은 케이스의 처리에서는 커널 버전 간의 큰 차이가 나지 않는다는 점을 볼 수 있다. NVMe I/O 소프트웨어 계층 자체가 매우 작기 때문에 구조적인 차이가 있음에도 이와 같은 상황에서는 작은 차이만을 보이는 것으로 추정된다.

Normal 워크로드에 대한 측정 결과는 그림 7의

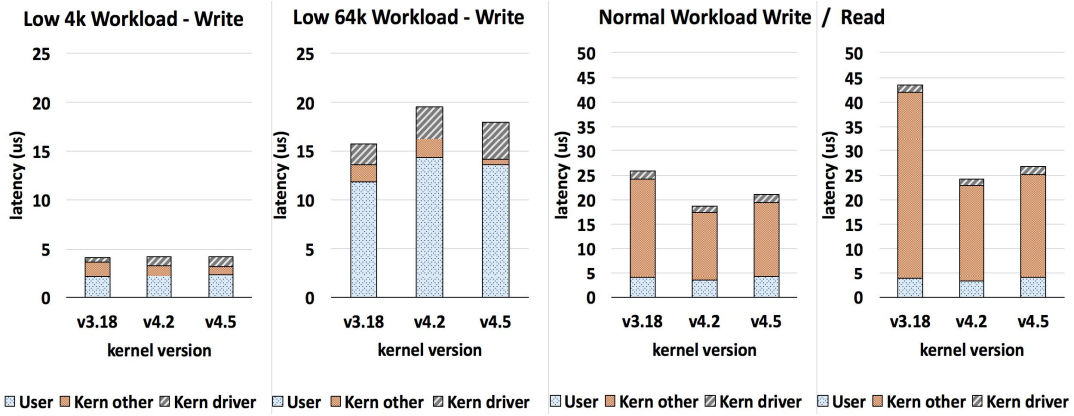

그림 7. 커널 버전 별로 측정된 I/O 계층별 소요시간 비교 (작을수록 좋음).
User: 유저 레벨 시간 / Kern driver, other: 커널 레벨 시간 - 드라이버, 드라이버 외 부분
Fig. 7 Comparison of spent time in each I/O layer by kernel version (lower is better).
User: time spent in user level / Kern driver, other: spent time in kernel level - in driver or others

우측 부분과 같다. blk-mq 적용 전후로 읽기 약 19us, 쓰기 약 8us의 차이가 났으며, 이는 3.18 버전의 소요시간을 기준으로 약 44% 단축된 것이다. 이 결과에서 더 I/O 요청이 집중되는 상황에서는 blk-mq가 적용된 새로운 구조가 초기의 블록 레이어 우회 방식보다 더 좋은 처리시간을 나타낸다는 것을 알 수 있다.

이는 커널 자체의 성능 향상과 함께, blk-mq의 소프트웨어-하드웨어 큐 구조가 초기에 NVMe 드라이버가 자체적으로 관리하였던 큐 구조보다 다중 코어 시스템에서 더 효율적이기 때문에 다수의 요청이 들어올 때 커널 수준에서 CPU를 점유하는 시간이 작아지고 그에 따라 성능 향상을 가져온 것으로 사료된다. 또한 앞서 3장에서 언급한 바와 같이 각각의 I/O에 대하여 메타데이터를 저장하기 위한 구조체 할당이 매번 따로 일어나던 이전의 구조에서 통합된 구조체를 드라이버 로드 시 큐 최대 크기만큼 사전 할당하여 사용하도록 바뀐 점 또한 성능 차이에 영향을 미친 것으로 보인다.

VI. 결론 및 향후 연구

본 논문은 리눅스 NVMe I/O 계층의 커널 버전별 차이를 구조 및 응답시간 측면에서 분석하였다. 커널 버전 및 그에 따른 NVMe 드라이버 구현방식의 차이에 따른 응답시간에서의 성능 차이가 확연함을 실험을 통해 확인하였다. 따라서 NVMe 저장장치를 사용하는 리눅스 커널기반의 모바일 시스템에서는 시스템 성능 최적화 과정에서 최신 NVMe 드라이버 구현과 관련 커널 코드 개선 부분을 적용함에 따른 성능 변화 요소를 중요하게 고려하여야 할 것으로 보인다. 향후 연구로는 NVMe I/O 계층과 기존에 모바일 장치에 사용되는 eMMC, UFS I/O 계층의 비교 분석을 수행하고자 한다.

References

[1] I. Kang, Y. Joo, S. Lim. “Analysis of I/O Response Time Throughout NVMe Driver Implementation Architectures,” Korean Institute of Information Scientists and Engineers, pp. 517-519. 2016 (in Korean).

[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. 6, 2015.

[3] E. Baram, “PCIe/NVMe in Mobile Devices,” Proceedings of the 2013 Flash Memory Summit 2015.

[4] M. Bjørling, J. Axboe, D. Nellans, P. Bonnet, “Linux Block IO: Introducing Multi-queue SSD Access on Multi-core Systems,”

Proceedings of the 6th ACM international. systems and storage conference, 2013.
[5] W. Fischer, G. Schönberger, “The Linux Storage Stack Diagram,” [Online] http://www.thomas-krenn.com/en/wiki/Linux_Storage_Stack_Diagram (License: CC-BY-SA 3.0, modified by Ingu Kang)
[6] J. Axboe, “Flexible I/O Tester,” [Online] https://github.com/axboe/fio
[7] J. Axboe, “Block Trace,” [Online] https://git.kernel.org/pub/scm/linux/kernel/git/axboe/blktrace.git/
[8] M. Kim, S. Lee, Y. Won, “Comparative Study on I/O Characteristics of Mobile web Browsers,” Proceedings of 5th IEEE International Conference on Consumer Electronics-Berlin, 2015.
[9] S. Jeong, K. Lee, S. Son, Y. Won, “I/O Stack Optimization for Smartphones,” Presented as part of the 2013 USENIX Annual Technical Conference, 2013.

Ingu Kang (강 인 구)

He received the B.S. degree in Computer Science from Kookmin University, Seoul, Korea, in 2015. He is currently pursuing M.S. degree in Computer Science at Kookmin Universty, Seoul, Korea. His research interests include storage stack, non-volatile memory storage, virtualization, human computer interaction.
Email: i@igk.me

Yongsoo Joo (주 용 수)

He received the Ph.D. degree in School of Computer Science and Engineering from Seoul National University, Seoul, Korea, in 2007. He is an assistant professor in the School of Computer Science in Kookmin University, Korea. His research interests include embedded systems and memory systems.
Email: ysjoo@kookmin.ac.kr

Sung-Soo Lim (임 성 수)

He received the Ph.D. degree in Electrical and Computer Engineering from Seoul National University, Seoul, Korea, in 2002. He is a professor in the School of Computer Science in Kookmin University, Korea. His research interests include virtualization, real-time systems, and mobile power management.
Email: sslim@kookmin.ac.kr