고성능 NVMe 장치를 위한 입출력 인터페이스의 읽기 성능 분석

정지현º 서동주 주용수 임성수 임은진
국민대학교 소프트웨어융합대학
{xcm1321, commisori, ysjoo, sslim, ejim}@kookmin.ac.kr

Read Performance Analysis of I/O Interface for High Performance NVMe Device

Jihyeon Jungº, Dongjoo Seo, Yongsoo Joo, Sung-Soo Lim, Eun-Jin Lim
School of Computer Science Kookmin University

요 약

리눅스의 기존 입출력 인터페이스는 인터럽트 기반으로 설계되어 있어 최근의 고성능 저지연 NVMe 저장장치에서 최대 성능을 끌어내지 못하는 비효율성이 지적되었다. 이를 극복하기 위해 폴링 기반의 새로운 입출력 인터페이스가 등장하고 있는데, 아직 이들에 대한 성능 분석은 충분히 이루어지지 않았다. 본 연구에서는 저지연시간을 가진 최신 NVMe 저장장치에 대해 최대 120개의 CPU 코어가 읽기 요청을 발생시키는 환경에서 최신 입출력 인터페이스들의 읽기 성능을 비교, 분석한다.

1. 서 론

최근 NVMe 장치가 널리 사용되기 시작하면서 이를 사용하는 소프트웨어 계층에 대한 연구가 많이 진행되고 있다. 대표적으로 Intel의 3D Xpoint나 삼성의 Z-NAND와 같은 낮은 입출력 지연 시간을 보장하는 기술들이 등장함에 따라 리눅스의 기존 비동기 입출력 인터페이스가 비효율성을 보이는 현상이 알려지게 되었다. 이러한 문제점을 보완하기 위해 최근 들어 io_uring, SPDK와 같은 비동기 입출력 인터페이스가 새로 제안되고 있는데, 실제 저장장치에 대해 해당 기술들이 어떻게 동작하는지에 대한 분석은 아직 충분하지 않은 상황이다. 본 논문에서는 삼성의 Z-NAND 기반 983 ZET SSD가 장착된 매니코어 시스템에서 다양한 입출력 인터페이스의 읽기 성능을 분석하고 성능에 영향을 미치는 요인을 분석한다.

2. 입출력 인터페이스별 동작 방식

2.1 Interrupt 방식

libaio[1]는 리눅스에서 비동기 입출력을 지원하기 위해서 만들어진 인터페이스로, linux 2.6에 통합되어 현재 리눅스에서 기본값으로 사용되고 있다. libaio는 입출력 요청 후 문맥을 교환하여 다른 작업을 수행하다가 인터럽트가 발생하면 입출력 요청을 마저 처리하는 방식으로 동작한다.

이러한 구현 방식은 지연시간이 수 ms에서 수십 ms에 달하는 HDD에 대해서는 효율적으로 동작하지만 수십 us의 저지연 시간을 제공하는 최근의 NVMe 저장장치에서는 실제 입출력 처리에 걸리는 시간보다 문맥 교환에 걸리는 시간이 훨씬 큰 비효율성을 보이게 된다.

2.2 Polling 방식

NVMe 기반 저지연 저장장치에서의 인터럽트 기반 비동기 입출력 인터페이스의 비효율성을 개선하기 위해 polling 방식을 기반으로 하고 driver를 user space로 옮기는 등의 다양한 최적화 기법을 적용한 입출력 인터페이스가 제안되고 있다.

설정값	설명
RWF_DSYNC	pwritev2 요청 시에 가능한 설정으로, 요청이 완료될 때에 메타데이터를 제외한 데이터 부분의 동기화가 보장된다.
RWF_HIPRI	설정되었을 경우 커널이 polling이 가능한지 확인 후 가능하다면 poll 아니면 요청의 우선순위를 높인다.
RWF_SYNC	RWF_DSYNC와 동일하며 메타데이터의 동기화가 추가로 보장된다.
RWF_NOWAIT	파일 블록 할당, 더티 페이지 정리, 뮤텍스 잠금, 정체된 블록 장치의 작업을 기다리지 않고 바로 진행하는 설정이다. 만약, 위의 조건에 걸리면 -EAGAIN을 반환한다.
RWF_APPEND	pwritev2 요청 시에 가능한 설정으로, 파일의 끝에 데이터를 추가하는 방식이다. 파일의 오프셋은 변경되지 않는다.

표 1: pvsync2 입출력 요청 시 추가 설정값

- pvsync2는 Linux 4.6에서 추가된 preadv2 / pwritev2 시스템 콜을 사용하는 입출력 인터페이스이다. preadv2 / pwritev2는 lseek과 read / write를 결합하고 배열 기반의 요청에서 벡터

* 이 논문은 2019년도 정부(과학기술정보통신부)의 재원으로 정보통신기획평가원의 지원을 받아 수행된 연구임 (No. 2014-0-00035, 매니코어 기반 초고성능 스케일러블 OS 기초연구)
* 이 성과는 정부(과학기술정보통신부)의 재원으로 한국연구재단의 지원을 받아 수행된 연구임(No.2018R1D1A1B05044558)

기반의 요청으로 바뀐 preadv / pwritev를 기반으로 만들어진 시스템 콜이다. preadv2 / pwritev2가 preadv / pwritev와 다른 점은 표 1에 나와있는 설정들을 추가로 지원한다는 것이다. 본 논문에서는 RWF_HIPRI를 polling과 함께 사용하여서 pvsync2를 사용하는 요청이 polling 기반으로 움직일 수 있도록 설정하였다.

- io_uring[2]은 Linux 5.1에서 새롭게 추가된 비동기 입출력 인터페이스이다. io_uring은 응용 프로그램의 입출력 요청을 처리하기 위해 원형 입력 요청 큐와 요청 완료 큐를 활용한다. 요청 완료 큐는 저장장치의 입출력 요청 완료 여부를 확인하는 방법으로 interrupt 방식과 polling 방식을 모두 지원하는데, 저지연시간을 가진 최신 NVMe 저장장치에서는 polling 방식을 사용하는 것이 더 나은 성능을 얻을 수 있다고 언급되어 있다[2][3]. io_uring의 동작 방식을 지정하기 위해 IORING_SETUP_IOPOLL 플래그 설정을 통해 요청 완료 큐에 대한 polling 적용 여부를 선택할 수 있고, 또한 IORING_SETUP_SQPOLL 플래그 설정을 통해 원형 입력 요청 큐의 사용 여부를 결정할 수 있다.

- SPDK[4][5]는 인텔에서 시작한 오픈소스 프로젝트로, 최신 NVMe 장치에서 고성능을 얻기 위해 새롭게 제안된 라이브러리 및 도구 모음이다. SPDK는 입출력 성능 개선을 위해 다음과 같은 최적화를 적용하였다. 첫째, 입출력 요청 처리를 담당하는 드라이버를 모두 유저 영역으로 옮겨 시스템 콜 호출을 피하고 유저 영역에서 직접 접근을 가능하게 하였다. 둘째, interrupt 대신 polling을 사용하여 총 지연시간과 지연시간의 변동폭을 줄였다. 셋째, 입출력 요청 경로에 있는 잠금을 모두 없앴다.

3. 실험 결과

IBM X3950 X6	
OS	Ubuntu 18.04
Kernel	Linux 5.6
CPU	Intel Xeon E7 8870 v2 (2.30GHz) * 8
Memory	768GB
NVMe	Samsung SSD 983 ZET 480GB
fio[6]	3.19
SPDK	v20.04-pre

표 2: 실험 환경

3.1 실험 환경

본 논문에서는 120개의 CPU 코어를 가진 매니코어 시스템에서 fio를 이용하여 읽기 요청에 대한 입출력 성능을 측정하기 위해 표 2와 같은 실험 환경을 구축하였다.

실험은 libaio, pvsync2, io_uring, SPDK 4가지 입출력 인터페이스를 사용하여 진행하였으며 libaio를 제외한 3개의 인터페이스는 모두 polling 방식으로 동작하도록 설정하였다.

또한 측정값의 변동폭을 줄이기 위하여 Intel Turbo Boost를 비활성화하였으며, Linux에서 0번 CPU 코어에 인터럽트가 집중되는 현상을 방지하기 위해 smp_affinity를 0번 마스크에 몰지 않고 여러 코어에 나누어 주었다. 마지막으로 CPU governor 설정은 performance로 하여 CPU가 설계 최고 클럭인 2.3 GHz를 유지할 수 있도록 하였다.

위 설정을 마친 후, fio를 사용해서 장치에 파일 시스템을 올리지 않고 직접 접근하여 job 개수의 증가에 따른 저장장치 별 4k 임의 읽기 성능이 어떻게 변화하는지 알아보았다. 본 논문에서 제시하는 모든 데이터는 같은 조건으로 5회 실험 후 중앙값을 사용하였다.

3.2 실험 결과 분석

그림 1은 Job 개수에 따른 4k 임의 읽기 성능을 측정한 결과를 보여준다. Interrupt 방식으로 동작하는 libaio의 경우는 job 개수와 상관없이 성능이 550 KIOPS에서 더 이상 올라가지 않는 모습을 보였다. Polling 방식으로 동작하는 pvsync2와 io_uring, SPDK는 libaio에 비해 상대적으로 우월한 성능을 보였지만, job 수에 따른 성능 변화는 각각 다른 양상을 보였다.

io_uring의 경우 job 개수가 2개일 때부터 곧바로 저장장치의 사양상 최대 성능을 보이며 120개까지 job 개수가 늘어나는 동안 해당 성능을 꾸준히 유지하였다. 다만 job 개수가 75를 넘는 구간에서 저장장치의 사양상 성능을 초과하여 900K IOPS가 넘는 값이 측정되었는데, 이는 fio가 많은 스레드를 한 번에 띄우지 못하여 발생한 오류로 확인되었다.

pvsync2의 경우는 8개까지의 job 개수에 대해서는 libaio보다도 낮은 성능을 보였지만 15개 이후부터 libaio보다 높은 성능을 확보하는 것을 확인하였다. 또한 45개까지의 job 개수에 대해서는 IOPS가 꾸준히 증가하여 io_uring의 최대치에 근접하지만 그 이후에는 IOPS가 오히려 소폭 감소하는 경향을 보였다.

SPDK의 경우 io_uring과 달리 job 개수가 하나일 때부터 저장장치의 최대 성능을 끌어낼 수 있음이 확인되었고, 이는 30개의 job까지 계속 유지되었다. 다만 SPDK는 job 하나당 HW Queue 하나가 할당되도록 설계되었다. 따라서 983 ZET의 경우 HW Queue가 32개이므로 그림 1에서는 30개까지의 job에 대한 실험 결과만 제시하였다.

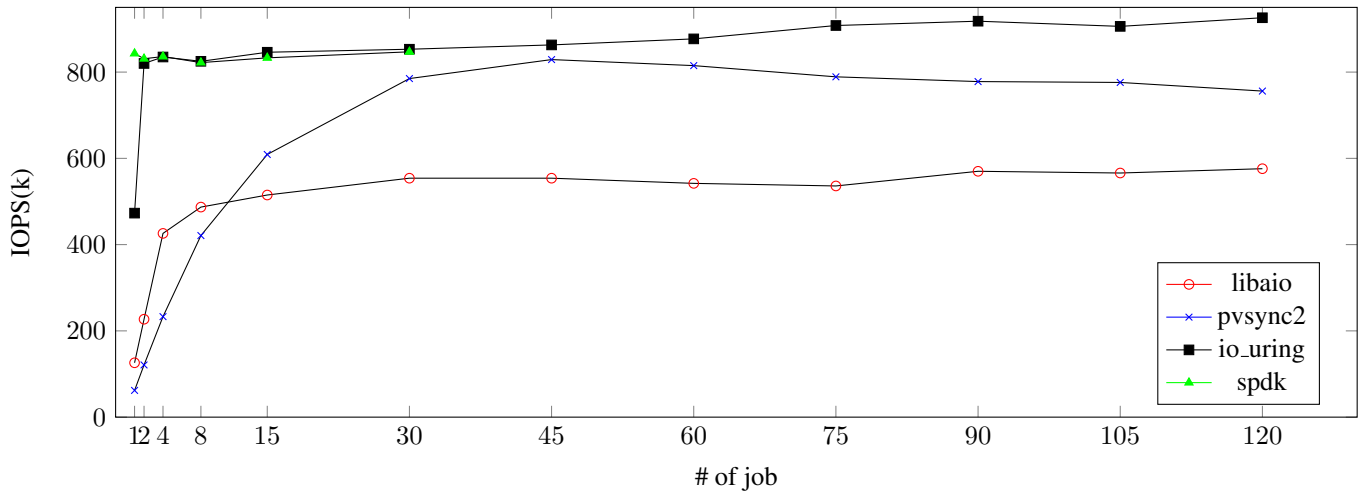

그림 1: Job 개수 증가에 따른 4k 임의 읽기 성능

4. 결론 및 향후 연구

본 논문에서는 최신의 저지연 NVMe 저장장치에서 interrupt 방식의 입출력 인터페이스보다는 polling 방식의 인터페이스가 뛰어난 읽기 성능을 확보할 수 있음을 확인하였다. 향후 연구로는 고성능 저지연 저장장치뿐만 아니라 보급형 대용량 NVMe 저장장치에서의 성능 측정 및 파일 시스템을 통한 읽기 성능에 대한 분석도 수행할 예정이다.

참고 문헌

[1] S. Bhattacharya, S. Pratt, B. Pulavarty, and J. Morgan, "Asynchronous I/O support in Linux 2.5," in *Proceedings of the Linux Sym..*, pp. 371–386, 2003.

[2] J. Corbet, "Ringing in a new asynchronous I/O API," *URL https://lwn.net/Articles/776703/*, 2019.

[3] J. Yang, D. B. Minturn, and F. Hady, "When Poll is Better than Interrupt," in *10th USENIX conference on File and Storage Technologies (FAST'12)*, vol. 12, pp. 3–3, 2012.

[4] 배진욱, 이성진, "인텔 spdk 및 관련 연구동향 소개," *정보과학회지*, vol. 37, no. 6, pp. 35–42, 2019.

[5] Z. Yang, J. R. Harris, B. Walker, D. Verkamp, C. Liu, C. Chang, G. Cao, J. Stern, V. Verma, and L. E. Paul, "SPDK: A Development Kit to Build High Performance Storage Applications," in *2017 IEEE International Conference on Cloud Computing Technology and Science (CloudCom)*, pp. 154–161, 2017.

[6] J. Axboe, "Fio-flexible io tester," *URL http://freecode.com/projects/fio*, 2014.