입출력 폴링 기법의 매니코어 확장성 분석

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

Analyzing Manycore Scalability of I/O Polling Schemes

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

요 약

운영체제는 상대적으로 느린 저장장치에 대한 입출력 요청을 효율적으로 처리하기 위해 인터럽트(interrupt) 기법을 사용해 왔다. 한편 최신 저장장치는 용량뿐만 아니라 성능 또한 획기적으로 개선되고 있으며, 최근 출시되는 NVMe SSD에서는 입출력 처리 지연 시간이 수십 마이크로초를 넘지 않는 성능을 보여주고 있다. 이러한 저지연 저장장치에서의 인터럽트 방식의 비효율성에 대한 해결방안으로 입출력 폴링(polling) 기법이 대안으로 제시되고 있다. 본 논문에서는 저지연 NVMe SSD가 장착된 매니코어 머신에서 최신 리눅스 운영체제가 제공하는 인터럽트 및 폴링 기법의 매니코어 확장성을 읽기 성능 관점에서 비교, 분석한다.

1. 서 론

최근 플래시 메모리의 발전으로 SATA SSD, NVMe SSD에 이르기까지 저장장치의 속도는 빠르게 발전하고 있다. 그중에서도 NVMe SSD는 최근 PCIe 4.0을 지원하며 순차 읽기 성능과 랜덤 읽기 성능 모두 SATA SSD 대비 10배 이상의 속도를 보여줄 정도로 발전하였다. 또한 일반적인 NVMe SSD가 수백 마이크로초 이상의 지연 시간을 갖는 반면에 최근의 Z-NAND, 3D Xpoint 등 저지연 기술을 제공하는 NVMe SSD는 30 마이크로초 이하의 지연 시간을 가진다.

인터럽트를 사용하여 I/O 요청을 처리하는 경우 문맥 교환이 일어나게 되는데, 이로 인해 발생하는 지연 시간이 I/O 지연 시간에 포함되며 많은 코어가 한 번에 읽기 요청을 하는 경우 오히려 성능이 저하되는 비효율성을 보인다. 이러한 한계를 극복하기 위한 방안으로 폴링 기반 I/O 처리 방식이 대안으로 제안되었으며 리눅스 psync I/O 및 Async I/O에 폴링이 차례로 통합되었다. 추가적으로는 io_uring[1] 및 SPDK[2] 등 폴링 방법을 사용하여 고성능을 얻고자 하는 새로운 입출력 엔진도 개발되고 있다. [3]

본 논문에서는 저지연 NVMe SSD가 장착된 매니코어 머신에서 최신 리눅스 운영체제가 제공하는 인터럽트 및 폴링 기법의 매니코어 확장성을 single file에 대한 랜덤 읽기 성능 관점에서 비교, 분석한다.

* 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)

2. NVMe 드라이버 동작 방식

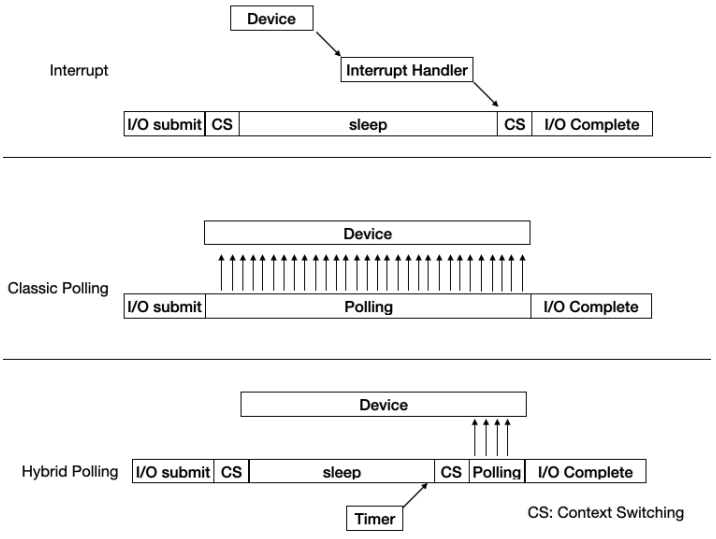

그림 1: 인터럽트, 클래식 폴링, 하이브리드 폴링 타임라인

2.1 인터럽트 방식

현재 리눅스에서 비동기 요청을 처리하기 위해 사용 중인 방식으로 I/O 요청을 장치로 보낸 후에 다른 일을 처리한다. 그 후에 인터럽트 Handler가 I/O 요청이 완료되었다고 알리는 경우 다시 I/O 요청을 처리한다.

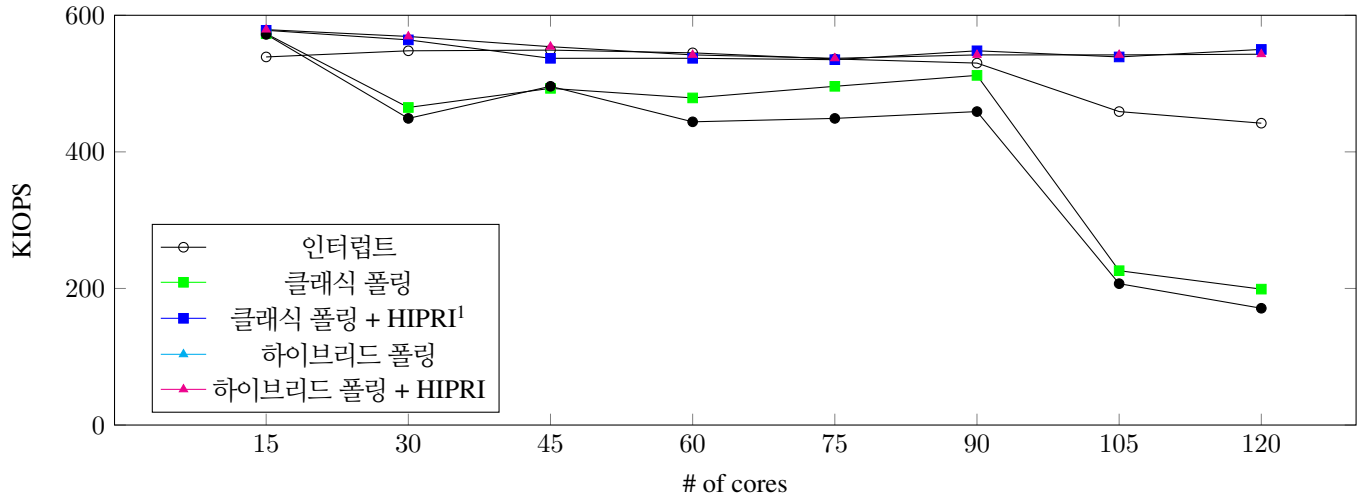

그림 2: CPU 코어 개수 증가에 따른 4k 임의 읽기 성능

2.2 폴링 방식

폴링 방식[4]은 동작 방식에 따라서 크게 클래식 폴링과 하이브리드 폴링[5] 두 가지로 나눌 수 있다.

- 클래식 폴링은 유저 공간에서 입출력 요청을 보내는 즉시 지속적으로 요청에 대한 작업이 완료되었는지 커널이 검사하는 작업을 시작한다. 그렇기 때문에 해당 작업을 위해 일하고 있는 CPU가 다른 일을 하지 못하게 되는 단점이 있다.
- 하이브리드 폴링은 입출력 요청을 커널이 받는 순간, 커널은 요청을 보낸 프로세스를 설정된 시간만큼 sleep 상태로 만든다. 특정 시간이 지난 후 timer 인터럽트에 의해서 깨어난 프로세스는 요청이 완료되었는지 확인하기 위해 폴링을 진행한다. 이를 통해 폴링에 비해 CPU 점유율은 감소하고 I/O latency는 비슷하게 유지할 수 있는 장점을 가진다.

3. 매니코어 확장성 분석

IBM X3950 X6	
OS	Ubuntu 18.04
Kernel	Linux 5.8
CPU	Intel Xeon E7 8870 v2 (2.30GHz) * 8
Memory	768GB
NVMe	Intel Optane SSD 900P 480GB
fio[6]	3.23

표 1: 실험 환경

1) fio 벤치마크에서 사용할 수 있는 옵션 중 하나로 pvsync2의 경우 RWF_HIPRI 플래그를 설정하는 데 사용한다. 이 플래그가 활성화되면 높은 우선순위를 갖도록 하고 폴링을 시도하도록 한다.

3.1 실험 환경

본 논문에서는 120개의 코어를 가진 매니코어 머신에 저지연 특성을 가진 Intel Optane SSD를 설치하여 표 1과 같은 실험 환경을 구축하였다. 매니코어 머신의 CPU 코어는 15개 단위로 활성화하여 15, 30, 45, 60, 75, 90, 105, 120개의 CPU 코어를 가진 8 종류의 매니코어 머신을 모사하였다.

읽기 workload는 fio 벤치마크를 이용하여 single file에 대해 랜덤하게 읽도록 발생시켰으며 I/O engine으로는 pvsync2를 사용하였다. 병목점 분석은 perf 및 btt를 이용하여 인터럽트, 클래식 폴링, 하이브리드 폴링에 대해 수행하였고, 폴링 모드에 대해서는 HIPRI 옵션 활성화에 따른 입출력 성능을 각각 측정하였다. NVMe 드라이버의 poll queue는 인터럽트 사용 시에는 비활성화하고 클래식 및 하이브리드 폴링에 대해서는 활성화하여 실험을 진행하였다. 또한 하이브리드 폴링 실험에 대해서는 io_poll_delay를 0으로 설정하여 커널이 입출력 지연시간에 맞추어 동적으로 동작하도록 설정하였다.

io_poll_delay 값을 0으로 설정할 경우 최근 I/O latency의 절반만큼 sleep했다가 깨어나서 폴링을 진행하고, 0보다 큰 값일 경우 해당 시간만큼 sleep 후 깨어나서 폴링을 진행한다. 또한 io_poll_delay를 -1로 설정하면 클래식 폴링으로 동작한다.

3.2 실험 결과

CPU 코어 수에 따른 각 입출력 기법의 입출력 성능 측정 결과를 그림 2에 나타내었다. 인터럽트의 경우 15 코어부터 거의 최대 성능에 가까운 모습을 보이며 90 코어까지는 성능이 유지되다가 105 코어와 120 코어에서 성능이 감소하는 결과를 보였다. 성능 저하의 이유를 분석한 결과 읽기 요청이 완료되는 과정에서의 오버헤드가 원인임을 확인하였다.

클래식 폴링에서는 HIPRI 옵션을 활성화하지 않을 경우 105 코

어 이상에서 급격한 성능 저하가 발생하였다. 해당 상황에서 입출력 명령의 처리과정을 분석한 결과 입출력 요청 완료 함수가 실행될 때 I/O가 폴링 대신 스케줄링으로 처리되었으며, 또한 요청을 보낸 후에도 spinlock이 걸려 그로 인한 오버헤드가 성능 저하를 발생시킴을 확인하였다. HIPRI 옵션을 활성화한 경우에는 모든 코어 수에 대해 저장장치의 사양상 최대 성능을 계속 유지하였으며 측정 대상 입출력 기법 중 가장 우수한 성능을 보였다.

다음으로 하이브리드 폴링에서 HIPRI 옵션을 비활성화한 경우에는 클래식 폴링과 같은 이유로 105 코어부터 심각한 성능 저하를 보임을 확인하였다. HIPRI 옵션을 활성화한 경우 모든 코어수에서 클래식 폴링과 동일한 입출력 성능을 확보할 수 있음을 확인하였다.

코어	인터럽트	클래식 폴링	하이브리드 폴링
15	50.6%	99.6%	59.2%
30	32.8%	99.3%	57.8%
45	29.6%	99.1%	59.1%
60	25.6%	99.0%	60.9%
75	26.0%	98.7%	63.5%
90	25.0%	98.6%	61.1%
105	36.4%	98.4%	60.6%
120	38.8%	98.1%	60.5%

표 2: 입출력 기법별 CPU 점유율

표 2는 인터럽트, 클래식 폴링과 하이브리드 폴링의 CPU 점유율을 나타낸다. 클래식 폴링은 100%에 가까운 CPU 점유율을 보이는 반면 하이브리드 폴링을 사용할 경우 약 40% 정도의 CPU 점유율 감소 효과를 얻을 수 있음을 확인하였다.

폴링을 사용하는 경우 적절한 옵션을 부여하지 않으면 성능이 오히려 기존 인터럽트보다 떨어짐을 확인하였다. 따라서 I/O를 요청하는 상황에 맞추어 적절한 조합이 요구됨을 확인하였고 폴링을 사용하는 경우 가장 우수한 성능을 얻을 수 있음을 확인하였다.

3.3 토의

폴링이 sync 읽기 요청에 대해서는 인터럽트에 비해 성능상 이점을 가지고 있음을 확인할 수 있었다. 또한 하이브리드 폴링은 sleep 시간 설정을 통해 클래식 폴링 기법의 과도한 CPU 점유율 문제를 유의미하게 개선할 수 있음을 보였다. 하지만 하이브리드 폴링의 CPU 점유율은 인터럽트에 비해 상대적으로 높아 추가 개선의 여지가 있음을 확인하였다.

4. 결론 및 향후 연구

본 논문에서는 저지연 NVMe SSD가 장착된 매니코어 머신에서 최신 리눅스 운영체제가 제공하는 인터럽트 및 폴링 기법의 매니코어 확장성을 비교, 분석하고 하이브리드 폴링의 효과와 한계 및 개선점을 살펴보았다. 향후 연구로 하이브리드 폴링의 CPU 점유율 추가 절감을 위한 sleep 시간 결정 알고리즘 개선을 시도하고자 한다.

참고 문헌

[1] J. Corbet, “Ringing in a New Asynchronous I/O API,” *URL https://lwn.net/Articles/776703/*, 2019.

[2] 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.

[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] D. Le Moal, “I/O Latency Optimization with Polling,” in *Proc. of USENIX Conference of Linux Storage and Filesystems Conference (VAULT)*, pp. 1–25, 2017.

[5] 이혜지, 이태형, 엄영익, “가상화 시스템에서의 하이브리드 폴링 방식 성능 분석,” *한국정보과학회 학술발표논문집*, pp. 1528–1529, 2019.

[6] J. Axboe, “FIO-Flexible IO Tester,” *URL http://freecode.com/projects/fio*, 2014.