오픈 채널 SSD에서 호스트 수준 FTL의 적응적 우선순위 설정 기법

Adaptive Priority Boosting Technique of the Host Level FTL in Open-Channel SSDs

요 약

오픈 채널 SSD(Open-Channel SSD)는 새로운 낸드 플래시 기반의 디바이스로, FTL이 디바이스 내부에서 동작하는 것이 아닌 호스트 시스템에서 동작한다. FTL이 호스트 시스템에서 동작함에 따라 FTL 프로세스는 호스트 시스템에서 동작하는 다른 여러 유저 프로세스와 경쟁 상태에 놓이게 되어 두 종류의 프로세스 모두 성능 저하를 겪게 된다. FTL 프로세스가 동작하는 방식을 고려하였을 때, 효율성을 위해 유저 프로세스보다 FTL 프로세스가 우선적으로 수행되어야 하는 상황들이 존재한다. 하지만 호스트 시스템의 CPU 스케줄러는 이와 같은 사실을 인지하지 못 해 두 프로세스 간 스케줄링 충돌이 발생한다. 본 연구는 실험적인 관찰을 통해 이러한 스케줄링 충돌이 존재함을 증명하고 이로 인해 발생하는 성능 저하를 관찰하였다. 또한, FTL 프로세스의 우선순위를 조정함으로써 CPU 스케줄러 내에서 발생하는 FTL 프로세스와 유저 프로세스 간의 경쟁을 완화할 수 있음을 보이며 성능 향상에 효과적임을 확인하였다.

1. 서 론

솔리드 스테이트 드라이브(SSD)는 낸드 플래시 메모리 기반의 스토리지 장치이다. SSD가 지니는 쓰기 전 지우기(erase-before-write), 쓰기 단위와 삭제 단위의 불일치와 같은 하드 디스크 드라이브(HDD)와 다른 특성의 처리를 위해 FTL(Flash Translation Layer)이라는 특수 펌웨어가 SSD 디바이스 내부에서 동작한다. FTL은 L2P(logical-to-physical) 주소 변환, 가비지 컬렉션(garbage collection), 웨어 레벨링(wear-leveling) 등의 기능을 수행하여 SSD가 HDD와 동일한 블록 I/O 인터페이스를 사용 가능하게 한다.

기존 SSD의 경우, FTL이 디바이스 내부에서 구현되어 펌웨어로 동작한다. 반면, FTL이 호스트 시스템에서 동작하는 오픈 채널 SSD라는 새로운 SSD가 등장하였다. 오픈 채널 SSD는 디바이스의 물리적 내부가 호스트 시스템에게 공개되어 호스트가 물리적 I/O 스케줄링 및 공간 관리를 제어할 수 있다. 디바이스 내부의 동작을 호스트가 제어하거나 알 수 없었던 기존 SSD와 달리 오픈 채널 SSD는 호스트가 디바이스와 함께 디바이스 제어의 의무를 나눠 갖는다. 그보다 더 중요한 것은 호스트가 SSD를 제어할 수 있다는 점을 활용하여 이미 잘 알려진 SSD의 단점을 더욱 개선할 수 있다는 것이다[2].

LightNVM[3]은 리눅스 환경에서 커널 모듈의 형태로 구현된 오픈 채널 SSD 서브시스템이다. pblk라는 LightNVM 내 인터페이스는 호스트 수준에서 동작하는 FTL의 역할을 수행한다. 호스트 수준 FTL은 호스트 시스템에서 구현되어 있기에 호스트의 CPU 스케줄러에 의해 스케줄링 되어야 호스트의 프로세서에서 동작 가능하다. 따라서 호스트 수준 FTL에 의해 호스트 시스템에서 동작하는 다른 프로세스에 대해서 간섭하는 현상이 발생한다. 본 연구에서는 호스트 수준 FTL에 의해 발생하는 호스트 시스템에서의 CPU 스케줄링 충돌이 존재함을 밝히며 프로세스의 우선순위 조정을 통해 CPU 스케줄러에서의 이러한 충돌을 완화할 수 있음을 보인다.

2. 배경 및 연구 동기

2.1. LightNVM와 pblk

LightNVM은 리눅스 환경에서의 오픈 채널 SSD용 서브시스템이다. NVMe 장치 드라이버, LightNVM 서브시스템 그리고 상위레벨 I/O 인터페이스의 세 가지 계층으로 구성되어 있다. 가장 하위 계층인 NVMe 장치 드라이버는 오픈 채널 SSD 디바이스 자체를 사용자 공간에 노출하는 역할을 담당하며 두 번째 계층인 LightNVM 서브시스템은 디바이스 내부 형상을 호스트 수준 FTL에게 노출한다. 상위레벨 I/O 인터페이스는 오픈 채널 SSD를 특정 호스트 수준 FTL로 동작하도록 구현된 타겟의 형태로 액세스하는 추상화를 제공한다.

pblk(Physical Block Device)는 LightNVM 타겟의 대표적인 예로, 기존 블록 I/O 인터페이스를 따르는 호스트 수준 FTL을 채택한다. pblk가 수행하는 작업으로는 L2P

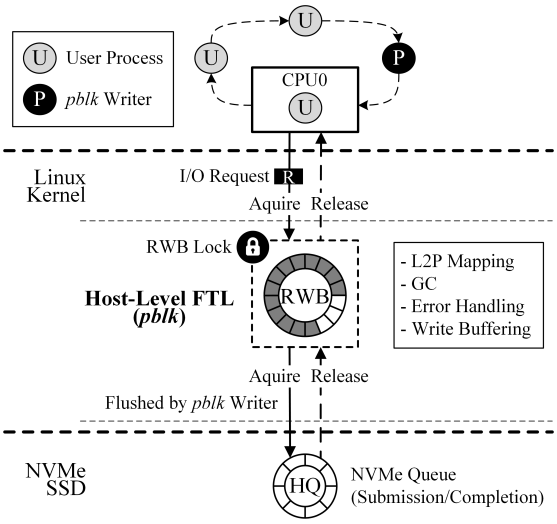

그림 1 pblk에서의 I/O 요청 처리

주소 변환, 가비지 컬렉션, 에러 핸들링, 플러시(flush) 핸들링 및 쓰기 버퍼링이 있다. 쓰기 버퍼링이 필요한 이유는 물리적 낸드 플래시 메모리의 페이지 크기와 호스트 시스템에 의해 정의된 섹터 크기(일반적으로 4KB) 간의 차이 때문이다. 이러한 데이터 버퍼링은 기존에 SSD 디바이스 내에서 수행되었으나 오픈 채널 SSD에서는 호스트 시스템의 메모리 내의 단일 원형 링 버퍼(Ring Write Buffer)가 데이터 버퍼링의 목적으로 사용된다. 호스트의 쓰기 I/O 요청이 해당 버퍼에 데이터를 삽입 성공하면 해당 요청에 대한 완료 응답이 전송된다.

버퍼 내에 데이터가 충분하거나 플러시 명령이 실행된 경우 버퍼 내의 일부의 엔트리는 버퍼에서 꺼내져 저장 장치에 기록된다. 이러한 작업은 pblk의 writer 데몬에 의해 수행된다. pblk writer 데몬은 1초마다 주기적으로 깨어나며 버퍼에 충분한 데이터가 쌓였을 때 유저 I/O 스레드에 의해 깨워지기도 한다.

2.2. pblk와 유저 I/O 스레드 간의 스케줄링 충돌

유저 I/O 스레드와 pblk writer 데몬 모두 호스트 시스템에서 동작하는 특성으로 인해 두 스레드 간에 스케줄링 종속성이 존재한다. 그림 1은 pblk의 동작 방식을 묘사한다. 유저 I/O 스레드와 pblk writer 데몬 모두 버퍼 전체에 대한 락(lock)을 획득한 후에야 버퍼에 액서스 가능하다. 버퍼에 공간이 있는 경우 유저 I/O 스레드는 버퍼에 데이터를 삽입할 수 있다. 그러나 버퍼에 남은 공간이 없을 때 유저 I/O 스레드는 데이터 엔트리를 삽입하지 못하며 pblk writer 데몬이 버퍼 엔트리를 소비하기까지 기다리면서 불필요하게 스케줄링 된다. 이러한 사실을 CPU 스케줄러는 알지 못해 두 스레드 모두의 효율성과 성능이 저하되는 현상이 발생한다. 본 연구에서는 실험을 통해 pblk writer 데몬과 유저 I/O 스레드 간의 스케줄링 종속성과 충돌의 영향을 관찰하였다.

2.2.1. 실험 환경

실험은 Intel Xeon E5-2630(2.40GHz, 16코어)가 장착된 서버에서 수행되었다. 오픈 채널 SSD 디바이스는 QEMU 기반 에뮬레이터[4]를 사용하여 8GB 에뮬레이션하였다.

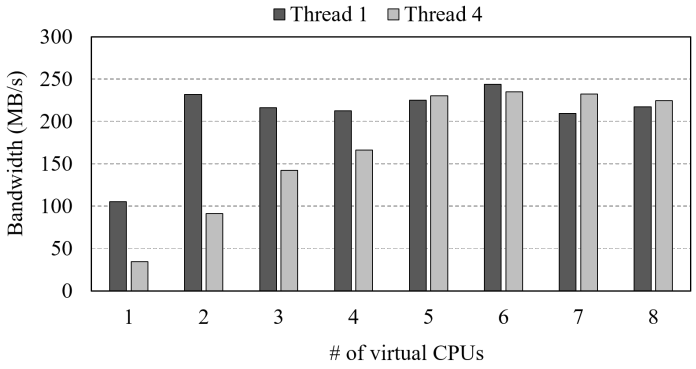

그림 2 vCPU 개수에 따른 pblk 성능

게스트 시스템은 16GB 메모리, 최대 8개의 가상 CPU로 설정되었다. 실험은 fio 벤치마크[1]를 4MB 블록 크기, direct I/O, 랜덤 쓰기 워크로드로 수행하였다.

2.2.2. 실험 결과

pblk writer 데몬과 유저 I/O 스레드 간의 스케줄링 경쟁에 의해 야기된 스케줄링 충돌을 관찰하기 위해 프로세서 수의 부족으로 인해 서로 다른 두 스레드가 경쟁하는 상황을 조성하였다. 그림 2를 통해 CPU 수가 증가할수록 대역폭이 향상하나 CPU 수가 2개(Thread 1) 또는 5개(Thread 4) 이상인 시점부터는 더 이상 대역폭이 개선되지 않는 모습을 보인다. pblk writer 데몬까지 합해서 유저 I/O 스레드 1개일 때는 총 2개, 4개일 때는 총 5개의 프로세스가 동작하므로 해당 시점에서 pblk writer 데몬과 유저 I/O 스레드 간의 경쟁이 멈춘다. 이와 같이 CPU 수를 늘리면 스레드 간의 경쟁 문제를 해결할 수 있으나 유저 I/O 스레드 수와 달리 CPU 수를 늘리는 데에는 한계가 있으므로 완전한 해결책이 되지는 못한다.

그림 3은 기존 pblk와 pblk writer 데몬에게 높은 우선순위를 부여하였을 때 발생하는 유저 I/O 스레드의 버퍼 삽입 실패 횟수를 비교한다. 해당 실험은 1개의 유저 I/O 스레드를 가상 CPU 수가 1인 환경에서 수행하였다. 우선순위가 높여진 경우는 0보다 더 큰 값을 가질 때만 표시하였다. pblk writer 데몬의 우선순위가 높여졌을 때 pblk writer 데몬은 유저 I/O 스레드보다 CPU 스케줄러에 의해 선호된다. 따라서 pblk writer 데몬은 버퍼에서 더 자주, 그리고 필요한 때에 적절히 엔트리를 소비할 수 있다. pblk writer 데몬이 기존 pblk 구현에서보다 더 적극적으로 활동하기 때문에 버퍼가 가득 차 있을 확률이 줄어든다. 따라서 유저 I/O 스레드는 비교적 적은 횟수의 시도로 버퍼에 데이터를 삽입하는 데에 성공할 수 있다. pblk writer 데몬의 우선순위가 높여진 경우는 대역폭 측면에서 기존의 pblk에 비해 두 배 이상 높은 성능을 보였다. 또한, 기존 pblk의 총 버퍼 삽입 실패 횟수는 우선순위가 높여졌을 때보다 70배 이상 많았다.

3. 스케줄링 종속성을 고려한 호스트 수준 FTL

3.1. pblk의 우선순위 조정

본 연구에서는 pblk writer 데몬의 우선순위를 조정함으로써 pblk writer 데몬과 유저 I/O 스레드 간의 스케줄링 종속성이 존재함을 CPU 스케줄러에게 암시하며 두

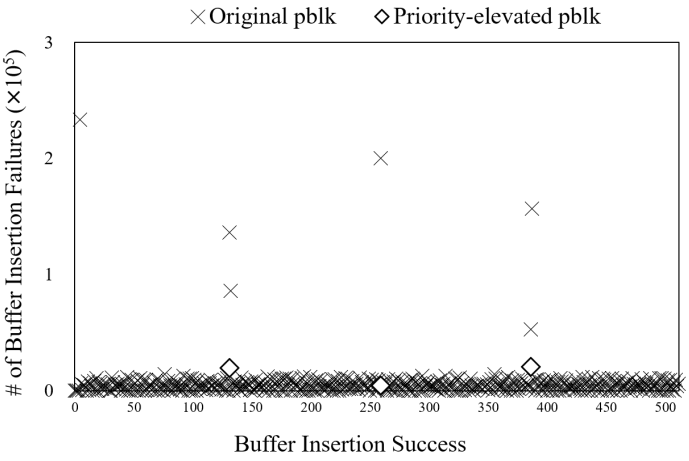

그림 3 버퍼 삽입 성공마다 발생한 버퍼 삽입 실패 횟수

스레드 간의 스케줄링 충돌을 완화할 수 있음을 보인다. pblk writer 데몬은 초기화 과정에서 0의 nice 값을 갖는다. pblk writer 데몬의 우선순위는 두 가지 상황에서 높여지도록 하였다. 첫 번째 상황은 유저 I/O 스레드가 버퍼 공간 부족으로 인해 버퍼 삽입 실패를 겪었을 때이다. 두 번째는 유저 I/O 스레드가 버퍼 삽입 성공 후 pblk writer 데몬을 깨웠을 때이다. 유저 I/O 스레드에 의해 pblk writer 데몬이 깨워지는 경우는 버퍼에 특정 임계값 이상의 엔트리가 쌓여있을 때이다. pblk writer 데몬이 CPU 스케줄러에 의해 스케줄링 되면, 즉 버퍼에서 엔트리를 빼내는 데에 성공하면 우선순위를 원래의 상태로 낮추었다. 이로써 CPU 스케줄러는 pblk writer 데몬의 우선순위를 통해 pblk writer 데몬이 우선적으로 실행되어야 하는 상황을 인식할 수 있어 버퍼가 꽉 찼을 때 버퍼를 더 빨리 비울 수 있게 된다.

3.2. 실 험

우선순위를 조정한 pblk의 성능 평가에는 2장에서의 실험 환경과 동일한 환경이 사용되었다. fio 벤치마크의 랜덤 쓰기 워크로드를 4개의 유저 I/O 스레드, direct I/O, 4MB 단위의 블록 크기로 수행하였다.

그림 4는 기존의 pblk와 우선순위를 조정한 pblk의 대역폭 성능을 보인다. 앞서 설명했듯이 기존 pblk는 CPU 수가 감소함에 따라 pblk writer 데몬과 유저 I/O 스레드 간의 경쟁이 심화되기 때문에 CPU 수가 적을수록 성능이 저하된다. 반면 우선순위를 조정한 pblk는 CPU 수와 관계없이 일정한 성능을 보인다. 경쟁이 사라진 시점과 비교했을 때, 우선순위를 조정한 pblk는 1.5~7.6%의 성능 저하를 겪은 반면 기존 pblk는 최대 84.1%, 최소 27.7%의 성능 저하를 겪었다.

pblk의 우선순위를 조정함으로써 호스트 시스템에서 실행되는 다른 프로세스를 방해하는 현상이 완화됨도 발견하였다. fio 벤치마크를 vCPU 1개인 환경에서 파이(π) 연산을 수행하는 멀티-스레드 CPU 연산 워크로드와 함께 실행하였다. 표 1에서 보이듯이 우선순위를 조정한 pblk는 CPU 연산 워크로드를 수행하는 데에 기존의 pblk보다 약 32% 더 적은 시간을 소요하였다. 이러한 현상이 발생하는 이유는 우선순위를 조정한 pblk가 버퍼 삽입

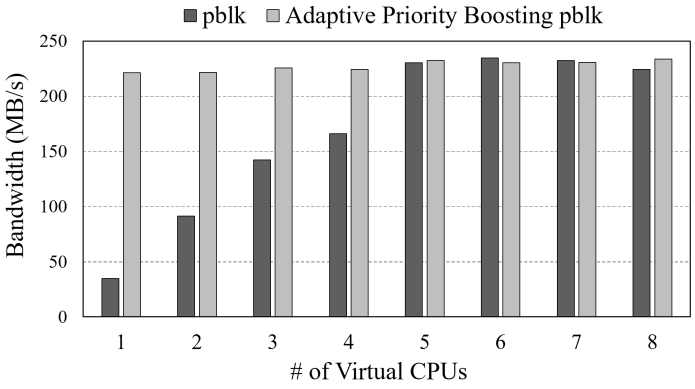

그림 4 pblk와 우선순위 조정한 pblk의 대역폭 성능

표 1 CPU 연산 워크로드와 함께 수행했을 때의 결과

	pblk	우선순위 조정 pblk
Bandwidth (MB/s)	20.11	204.53
Context Switch 횟수	4905	110
연산 워크로드 수행 시간 (sec)	246	167

실패를 훨씬 더 적게 경험해 문맥 전환(context switch)이 덜 발생하였기 때문이다. 따라서 I/O 수행 워크로드에 의한 불필요한 스케줄링이 덜 발생하였기 때문에 CPU 연산 프로세스가 수행하는 데에 더 적은 영향을 끼치게 되었다.

4. 결론 및 향후 연구

본 연구는 오픈 채널 SSD의 호스트 수준 FTL이 호스트 시스템 내에서 동작함으로써 발생하는 CPU 스케줄러에서의 충돌이 존재함을 증명하였다. 호스트 수준 FTL은 호스트 시스템 내의 다른 유저 스레드와 함께 처리되므로 두 스레드 간에 스케줄링 종속성이 존재한다. 이러한 종속성으로 인해 발생하는 스케줄링 충돌을 완화하는 데에 호스트 수준 FTL의 데몬의 우선순위를 조정하는 것이 효과적임을 확인하였다.

FTL이 동작해야 하는 상황에서 우선순위를 높여주고 그렇지 않은 상황에선 다시 우선순위를 낮추는 단순한 접근방식만으로도 성능 저하를 매우 완화할 수 있었다. 이를 통해 PID controller[5]의 도입으로 적응적으로 우선순위를 조정하거나 FTL 데몬과 유저 스레드 간의 완전한 격리를 달성하는 등의 추가적인 작업으로 성능을 더욱 향상할 수 있을 것으로 기대된다.

참고 문헌

[1] J. Axobe, “Fio-flexible io tester,” URL *http://freecode.com/projects/fio*, 2014.

[2] J. González and M. Bjørling, “Multi-tenant I/O Isolation with Open-Channel SSDs,” in Nonvolatile Memory Workshop (NVMW), 2017.

[3] M. Bjørling, J. González, and P. Bonnet, “LightNVM: the Linux Open-Channel SSD Subsystem,” in FAST, pp. 359-374, 2017.

[4] OpenChannelSSD, “qemu-nvme,” URL https://github.com/OpenChannelSSD/qemu-nvme, 2018.

[5] A. O’Dwyer, “Handbook of PI and PID controller tuning rules.”, Imperial College Press, 2009.