커맨드큐를 지원하는 모바일 플래시 메모리 저장장치의 I/O 성능 향상

한규화 ○ 신동군
성균관대학교
Hgh6877@skku.edu, Dongkun@skku.edu

Improving I/O Performance of Command-Queue-Supporting Mobile Flash Memory Storage

Kyuhwa Han○ Dongkun Shin
Sungkyunkwan University

요 약

차세대 모바일 플래시메모리 저장장치들은 저장장치 수준의 병렬성을 활용하기 위해 커맨드큐를 적극적으로 지원한다. 하지만, 커맨드큐를 사용할 경우, 호스트 I/O 스케줄러와 커맨드큐에서 중첩된 스케줄링이 발생하여 사용자가 정의한 I/O 명령의 우선순위가 지켜지지 않아, 스케줄링의 정확도가 감소한다. 본 연구에서는 커맨드큐에서 높은 우선순위의 읽기/동기 쓰기 명령이 낮은 우선순위의 미리-읽기/쓰기 명령에 의해 지연되는 현상을 해결하기 위해 LAQ (Latency-aware queueing) 스케줄러를 제안한다. LAQ 스케줄러는 읽기 명령을 읽기 명령(read request)과 미리-읽기 명령(read-ahead request)으로 구분하여 관리하고, 미리-읽기 명령에 낮은 우선순위를 할당하여 높은 우선순위 명령에 대한 사용자 반응성을 향상시킨다. 또한, 낮은 우선순위 I/O 명령을 커맨드큐에 전달하는 양을 제한하여, 높은 우선순위 I/O의 응답시간을 향상시키는 LAT (Latency-aware Throttling) 기법을 제안한다. UFS(Universal Flash Storage)를 사용하는 삼성 갤럭시 S6에서 백그라운드 읽기를 수행하면서 주기적인 읽기를 수행할 때, 주기적인 읽기의 응답시간이 평균 60% 개선되는 것을 확인했다.

1. 서 론

커맨드큐는 저장장치에게 여러 개의 I/O 명령을 동시에 요청하는 기법이다. 이 기법은 HDD 장치에서 커맨드큐로 전달된 여러 I/O 명령의 순서를 재배치하여 디스크 탐색시간(seek time)을 줄이고 I/O 대역폭을 향상시키기 위해 고안되었다[1]. 반면 플래시메모리 저장장치는 저장장치 수준의 병렬성을 활용하기 위해 커맨드큐를 사용한다. 예를 들어, 대표적인 플래시메모리 저장장치인 SSD(Solid-State Drive)는 다수의 플래시 메모리 칩이 병렬적으로 구성되어 있어, 여러 칩에서 동시에 I/O를 병렬적으로 처리할수록 높은 대역폭을 얻을 수 있다.

최근에는 커맨드큐를 제공하는 차세대 모바일 저장장치인 UFS(Universal Flash Storage)가 등장하였으며[2], NVMe SSD에서는 커맨드큐를 보다 적극적으로 활용하기 위해 specification 상 최대 64k개의 커맨드큐를 생성 가능하고 각 커맨드큐마다 최대 64k개까지의 I/O 명령을 전달할 수 있게 정의하고 있다[3]. 하지만, 커맨드큐를 사용할 경우, 호스트 O/S의 I/O 스케줄러가 결정한 I/O 명령의 처리 순서가 저장장치의 커맨드큐에서 다시 한번 스케줄링되는 중첩된 스케줄링[4]이 발생하여 스케줄링 정확도가 저하될 수 있다.

리눅스는 다양한 I/O 명령들의 특성에 알맞은 서비스를 제공하기 위해서 I/O 명령의 우선순위를 정의하고 이를 기반으로 I/O 명령의 처리 순서를 결정하는 I/O 스케줄러를 제공한다. 리눅스에서 기본으로 사용되는 CFQ I/O 스케줄러[5]는 I/O 명령을 동기 I/O 명령(읽기, 동기 쓰기)과 비동기 I/O 명령(비동기 쓰기)으로 구분하고, 동기 I/O 명령을 우선적으로 저장장치에게 전달한다. 반면 커맨드큐에서는 읽기 명령과 쓰기 명령으로 I/O 명령을 구분하며, 독자적인 알고리즘에 의해 I/O 명령의 처리 순서가 결정된다[6].

CFQ I/O 스케줄러는 읽기 명령과 미리-읽기 명령의 우선순위를 구분하지 않는다. 이로 인해 비동기 명령인 미리-읽기 명령은 동기 명령과 같은 우선순위로 커맨드큐에 전달된다. 하지만 커맨드큐에서는 읽기 명령이 쓰기 명령보다 높은 우선순위를 가지기 때문에, 비동기 명령인 미리-읽기 명령이 동기 쓰기 명령보다 빠르게 처리되어 동기 쓰기에 대한 사용자의 반응성의 저하가 발생한다. 또한 커맨드큐에서는 각 I/O 명령이 동기 명령인지 비동기 명령인지 구분할 수 없으므로, 미리 전달된 비동기 명령에 의해 동기 명령의 처리가 지연되는 현상이 발생한다.

본 연구에서는 앞서 소개한 문제를 해결하기 위해 LAQ (Latency-aware queuing) 스케줄러를 제안한다. LAQ 스케줄러는 기존의 읽기 명령을 읽기 명령(read request)과 미리-읽기 명령(read-ahead request)으로 구분하고, 미리-읽기 명령에 낮

이 논문은 2016년도 정부(미래창조과학부)의 재원으로 한국연구재단의 지원을 받아 수행된 연구임 (No. 2016R1A2B2008672)

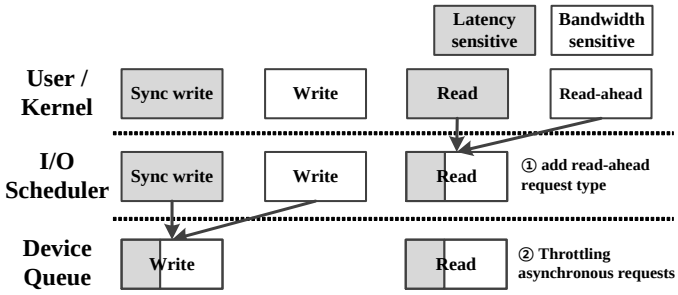


Figure 1. Semantic disappearing through the vertical I/O stack

은 우선순위를 할당하여 높은 우선순위 명령에 대한 사용자 반응성을 향상시킬 수 있다. 또한 커맨드큐에서 호스트가 정의한 우선순위가 지켜지지 않는 문제를 해결하고 높은 우선순위를 가진 I/O 명령의 응답시간을 보장하는 LAT(Latency-aware Throttling) 기법을 제안한다. 본 연구에서 제안된 기법은 UFS를 탑재하고 있는 스마트폰 환경에서 구현하였다. 백그라운드 읽기를 수행하면서 주기적인 읽기를 수행할 때, 주기적인 읽기의 응답시간이 평균 60% 개선되는 것을 확인했다.

2. Latency-aware queueing

2.1 기존 I/O 스케줄러의 문제점

Figure1과 같이 리눅스 I/O 스케줄러는 사용자와 커널(kernel)에게서 응답시간이 중요한 읽기/동기 쓰기 명령과 대역폭이 중요한 미리-읽기/비동기 쓰기 명령을 전달받는다. 하지만 현재의 리눅스 I/O 스케줄러는 읽기 명령과 미리-읽기 명령을 구분하지 않는다. 이는 기존의 I/O 스택이 임의 읽기가 느린 장치인 HDD를 바탕으로 설계되어 있어, 읽기 명령과 미리-읽기 명령을 연속적으로 처리하기 위함이다. 하지만 플래시 메모리 저장장치는 임의 접근이 빠르다는 장점을 가지고 있어, 읽기 명령과 미리-읽기 명령을 구분하여 관리할 필요가 있다.

또한 저장장치의 커맨드큐는 동기 쓰기와 비동기 쓰기 명령을 구분하지 않는다. 이는 블록 I/O 인터페이스를 통해 저장장치에 전달할 수 있는 정보가 한정적이어서 저장장치는 호스트 I/O 스케줄러에서 관리하는 다양한 우선순위 정보를 알 수 없어 I/O명령을 읽기와 쓰기로만 구분하기 때문이다. 하지만 이와 같은 구분으로는 호스트 I/O 스케줄러가 지정한 높은 우선순위를 가진 명령과 낮은 우선순위를 가진 명령을 정확히 판별할 수 없기 때문에 호스트가 정의한 우선순위가 지켜지지 않는 문제가 발생한다.

본 연구에서는 블록 I/O 계층에서 읽기 I/O 명령이 일반읽기 명령인지 미리-읽기 명령인지 구분하기 위하여 *is_rahead* 필드를 추가하였다. VFS(Virtual File System) 계층은 미리-읽기 명령을 수행하는 경우, 해당 필드를 마킹하여 I/O 스케줄러가 읽기 명령과 미리-읽기 명령을 구분할 수 있도록 한다. 또한, 커맨드큐 내에서 I/O 명령의 순서가 뒤바뀌는 문제를 해결하기 위해, I/O 스케줄러에서 동시에 커맨드큐로 전달하는 비동기 I/O 명령의 수를 제한하여, 비동기 I/O 명령이 커맨드큐를 불필요하게 채우는 것을 방지한다.

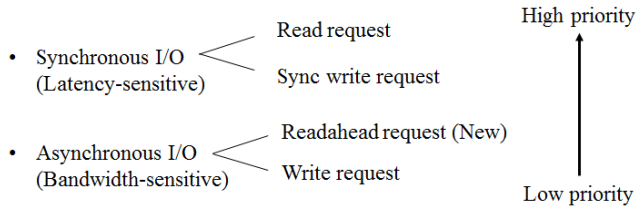


Figure 2. Request type classification

2.2 미리-읽기 명령의 I/O 우선순위

본 연구에서는 CFQ I/O 스케줄러에 미리-읽기 명령 속성을 추가하였으며, Figure2와 같이 동기 명령인 읽기와 동기 쓰기 명령을 높은 우선순위로 설정하고, 비동기 명령인 미리-읽기와 비동기 쓰기 명령을 낮은 우선순위로 정의한다. 미리-읽기 명령은 프로세스가 빠른 시간 안에 접근할 데이터를 포함하고 있으며, 프로세스가 데이터를 접근할 때까지 I/O가 완료되지 않으면 프로세스가 이를 기다리기 위해 블록되므로 write-back으로 요청된 비동기 쓰기 명령보다 높은 우선순위를 부여한다.

LAQ 스케줄러는 이러한 구분을 기반으로 높은 우선순위명령인 동기 I/O 명령을 우선적으로 저장장치의 커맨드큐에게 전달하며, 동기 I/O 명령이 경우 낮은 우선순위 명령인 비동기 I/O 명령을 전달한다. 비동기 I/O 명령 중 미리-읽기 명령은 비동기 쓰기 명령보다 높은 우선순위를 가지고 있으므로 우선적으로 전달하고 비동기 쓰기 명령은 다른 명령들이 없는 경우에만 저장장치에 전달한다.

2.3 Latency-aware throttling

본 연구에서는 비동기 I/O 명령이 커맨드큐를 점유하여, 이후에 전달된 동기 I/O 명령의 응답시간이 증가하는 문제를 해결하기 위해 LAT 기법을 제안한다. LAT 기법은 평상시에 OIS를 저장장치의 대역폭을 최대로 활용할 수 있는 OIS 값인 Th_{OIS_Max}로 설정하여, 저장장치의 대역폭을 최대한 활용할 수 있게 한다. 만약 OIS값이 Th_{OIS_Max}인 상황에서 동기 I/O 명령이 요청된다면, 일정 시간(Th_{wait}) 이내에는 커맨드큐에 동시에 전달할 수 있는 I/O(OIS: Outstanding I/O Size)의 크기를 Th_{OIS_Min}으로 제한하여 비동기 I/O 명령들이 커맨드큐를 불필요하게 채우는 것을 방지한다. Th_{OIS_Min}이 낮을수록 연속해서 발생하는 높은 우선순위 I/O 명령의 응답시간을 향상시킬 수 있지만, 커맨드큐의 사용을 제한함으로 인한 대역폭의 저하가 커지므로 이를 고려하여 Th_{OIS_Min}를 결정하는 것이 필요하다. 본 연구에서는 Th_{OIS_Min}을 Th_{OIS_Max}의 1/2로 사용하여 성능평가를 진행하였다.

성능 평가를 진행하기에 앞서, Th_{wait}, Th_{OIS_Max}의 값을 설정하기 위하여 UFS를 사용하는 스마트폰에서 I/O 크기를 바꿔가면서 임의 읽기/쓰기를 수행할 때의 대역폭을 측정하였다.

이 때, 읽기와 쓰기 명령 모두 2048 섹터의 I/O 명령이 커맨드큐에 큐잉되면 최대 대역폭을 보이는 것을 확인하였으며, 이를 기반으로 Th_{OIS_Max}값을 2048 섹터로 정의한다. 또한 크롬 브라우저, 카메라를 실행하는 사용자 시나리오에서 93%가량의 동기 I/O 명령이 1ms 이내에 다시 요청되는 것을 확인하였으며, 1ms를 Th_{wait}값으로 사용한다.

3. 실 험

3.1 실험환경

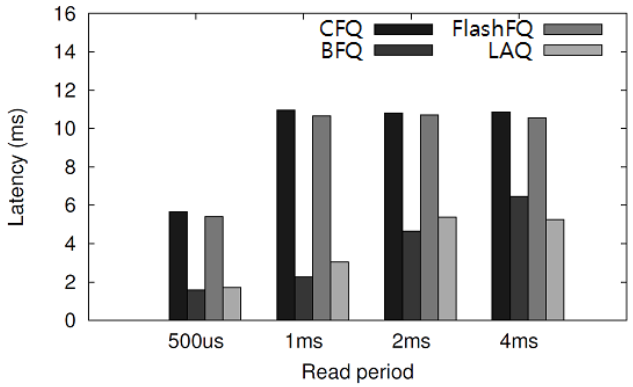
본 연구에서는 LAQ I/O 스케줄러성능을 평가하기 위해 32GB UFS를 탑재한 삼성 갤럭시 S6를 사용하였는데, 해당 스마트폰은 안드로이드 5.0.2 플랫폼 환경에서 리눅스 커널 3.10.61를 O/S로 사용한다. LAQ 스케줄러는 해당 O/S를 기반으로 VFS 계층과 CFQ I/O 스케줄러를 수정하여 구현하였다. 본 기법의 성능을 평가하기 위하여 기존의 CFQ I/O 스케줄러와 BFQ I/O 스케줄러[7], FlashFQ[8]에서 같은 실험을 수행하여 성능을 비교하였다. FlashFQ는 커맨드큐에 16개의 I/O 명령을 큐잉 할 수 있도록 설정하였으며, 우선순위 기반 I/O 스케줄링을 추가하여 다른 I/O 스케줄러 기법과 동등하게 비교할 수 있게 하였다.

3.2 실험 결과

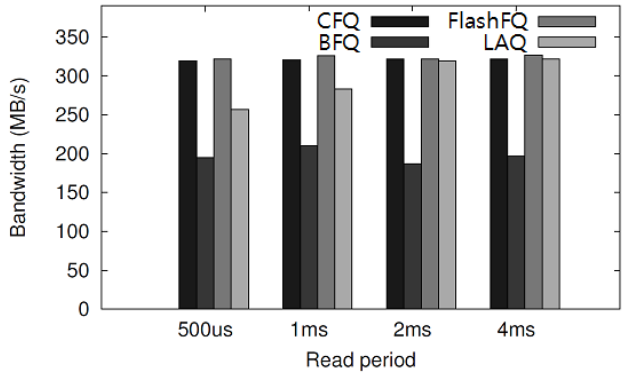
Figure3은 하나의 프로세스가 주기적으로 4KB의 임의 읽기 명령을 수행하고, 동시에 백그라운드 프로세스가 fio 밴치마크를 사용하여 순차 읽기 명령을 수행하면서 측정한 주기적 읽기의 응답시간과 백그라운드 프로세스의 대역폭을 보여준다. 최대 미리-읽기의 크기는 기본값인 256KB로 사용하였으므로 백그라운드 프로세스는 512개 섹터 크기의 순차-읽기를 반복적으로 수행한다.

Figure3의 (a)는 주기적 읽기 명령의 응답시간을 보여주는데, *CFQ*와 *FlashFQ*는 커맨드큐에게 미리-읽기 명령을 모두 전달하였으므로 이후에 발생한 주기적 읽기 명령에 대해 긴 응답시간을 가진다. 반면 *LAQ*의 경우, 커맨드큐에 전달하는 미리-읽기 명령의 크기를 Th_{OIS_Max}로 제한하였기 때문에 *CFQ* 대비 60%의 응답시간을 향상시켰다. 또한 주기적인 읽기의 발생시간이 1ms 이내인 경우, 커맨드큐에 전달하는 미리-읽기 명령의 크기를 Th_{OIS_Min}까지 제한함으로서 좀 더 향상된 응답시간을 보였다. *BFQ*에서는 주기적인 I/O를 요청하는 프로세스를 soft real-time 프로세스로 가정하고 보다 많은 버짓을 할당해 주는데, 이로 인해 *LAQ* 기법과 비슷한 응답시간을 보인다.

Figure3의 (b)는 백그라운드 읽기의 대역폭을 보여준다. *CFQ*와 *FlashFQ*는 미리-읽기 명령을 커맨드큐에 제한 없이 전달하기 때문에 높은 대역폭을 보인다. 반면 *BFQ*는 주기적인 읽기를 발생시키는 프로세스에게 높은 버짓을 할당했기 때문에, 상대적으로 백그라운드 읽기의 처리가 늦어져 대역폭이 낮은 것을 볼 수 있다. *LAQ* 기법에서는 주기적인 읽기의 주기가 1ms



(a) I/O latency of periodic read



(b) Bandwidth of background read

Figure 3. Periodic read and background read

보다 작은 경우, Th_{OIS_Min}까지 백그라운드 읽기를 제한하기 때문에 *CFQ*와 *FlashFQ*보다 낮은 성능을 보이지만 *BFQ*보다는 높은 성능을 보인다

4. 결론

본 연구는 호스트 I/O 스케줄러와 커맨드큐에서 중첩된 I/O 스케줄링으로 인한 스케줄링 정확도가 저하되는 문제를 해결하였다. 커맨드큐가 제공하는 대역폭을 충분히 활용하면서도 높은 우선순위를 가지는 I/O 명령의 응답시간을 향상시키기 위한 LAQ 스케줄러를 제안하여 실제 모바일 장치에서 평가를 진행하였다. 제안된 기법은 높은 우선순위를 가진 I/O 명령의 응답 시간을 보장하여 백그라운드 I/O가 방해하는 상황에서 주기적 읽기의 응답 시간을 CFQ 대비 60% 개선하였다.

참고문헌

[1] SATA Native Command Queuing, https://www.sata-io.org/native-command-queuing

[2] UNIVERSAL FLASH STORAGE (UFS), Version 2.1, http://www.jedec.org/standards-documents/docs/jesd220c

[3] Y. Son, H. Kang, H. Han, and H. Y. Yeom, "An Empirical Evaluation of NVM Express SSD," IEEE ICCAC, 2015

[4] Y. J. Yu, D. I. Shin, H. Eom, and H. Y. Yeom, "NCQ vs. I/O scheduler: Preventing unexpected misbehaviors," *ACM Transactions on Storage, vol 6, no. 1*, 2010

[5] CFQ, https://www.kernel.org/doc/Documentation/block/cfq-iosched.txt

[6] S. Park, E. Seo, J-Y. Shin, S. Maeng, and J. Lee, "Exploiting internal parallelism of flash-based SSDs," *IEEE Computer Architecture Letters* , vol. 9, no. 1, 2010.

[7] BFQ Valente, Paolo, and Arianna Avanzini. "Evolution of the BFQ Storage-I/O Scheduler." *IEEE 2015 Mobile Systems Technologies Workshop*, 2015

[8] K. Shen and S. Park, "FlashFQ: A Fair Queueing I/O Scheduler for Flash-Based SSDs," *USENIX ATC*, 2013