I/O 스케줄러를 통한 SSD 성능향상 기법연구

강승엽*, 박현찬*, 유혁*
*고려대학교 컴퓨터전파통신공학과
e-mail : {sykang, hcpark, chuckyoo}@os.korea.ac.kr

A Study on I/O Scheduler for Improving SSD Performance

Seungyup Kang*, Hyunchan Park*, Chuck Yoo*
* Department of Computer Science and Engineering, Korea University

요 약

Solid State Device(SSD)는 플래시 메모리 모듈을 기반으로한 차세대 저장장치로서 디스크 회전방식의 Hard Disk Drive(HDD)를 대체하는 장치로 주목받고 있다. 하지만, 현재의 운영체제 I/O 스케줄러는 HDD 에 최적화되어 있기 때문에 플래시 모듈 기반의 SSD 의 성능을 최대한 끌어내지 못하는 한계를 갖고 있다. 따라서, 본 논문에서 우리는 SSD 의 성능을 끌어올리기 위해 SSD 의 특징을 분석하고 이를 바탕으로 SSD 에 최적화된 I/O 스케줄러를 고안하였다. 새로운 스케줄러(NSCHED)는 I/O Request 를 두 그룹으로 분류하고 각 요청에 타이머를 설정함으로써 I/O Request 를 빠르게 처리함과 동시에 기아방지를 위한 기법을 가진다. 우리는 Linux 2.6.30 에서 NSCHED 스케줄러를 구현하여 Postmark 벤치마크를 통해 성능평가를 수행했으며, NSCHED 스케줄러가 기본 Linux I/O 스케줄러보다 30% 가량 낮은 응답 시간이 소요됨을 확인했다.

1. 서 론

최근 SSD 는 모든 업계 및 학계에서 차세대 저장장치로 각광받고 있다[1][2]. SSD 는 전력소모가 낮을 뿐만 아니라 충격에 강하고 비교적 작은 크기로 휴대용 미디어 플레이어와 노트북 컴퓨터와 같은 모바일 제품 시장에서 주목받고 있다. 또한 SSD 는 균일하고 빠른 액세스 시간을 가지고 있기 때문에 빠른 접근속도를 요하는 엔터프라이즈 서버 시장에서도 각광 받고 있다. 이러한 이유로, SSD 는 차세대 저장 장치의 중추적인 기술로 간주되고 있다.

지난 수년 동안 연구자들은 SSD 의 성능을 끌어올리려는 지속적인 노력을 해왔다. 일부 연구자는 I/O Subsystem 전체 구조를 수정하는 시도를 해왔다[3]. 그러나, I/O Subsystem 전체 구조를 수정하면 이전 버전과의 호환성을 보장할 수 없는 문제가 있다. 이전 버전과의 호환성을 유지하면서 SSD 의 성능을 최대화하기 위해서 많은 연구자들이 I / O 스케줄러에 주목해 왔다[4][5]. Dunn 과 Reddy[4]는 SSD 의 성능에 영향을 미치는 매개 변수를 추출하기 위한 프레임 워크를 제시했지만, 기존 I/O 스케줄러의 성능을 뛰어넘지 못했다. [5]에서 I/O Request 를 논리 블록 할당(Logical Block Allocation, LBA) 크기로 묶어서 처리하는 기법을 고안했고 이는 블록간의 성능저해 요소를 피해감으로써 일정수준이상의 성능향상을 이끌었다. 하지만 이들은 LBA 크기로 요청을 묶는 과정에서 쓰기요청의 분류하지 않아 더 높은 성능향상의 기회를 놓쳤다.

따라서 우리는 SSD 의 성능향상을 이끌어내기 위해 I/O 요청을 그룹별로 나누고 기아방지를 위한 타이머를 설정한 새로운 I/O 스케줄러(NSCHED)를 제안한다.

우리는 Linux 2.6.30 에서 NSCHED 스케줄러를 구현했고, 그 결과 기존 I/O 스케줄러에 비해 30% 가량 낮은 응답 시간을 가짐을 확인했다.

2. 배경 지식

a. Linux 기본 스케줄러

Linux 2.6.30 은 기본적으로 Noop, Deadline, Anticipatory, CFQ 등 4 개의 I/O 스케줄러를 제공한다. 하지만, Noop 스케줄러를 제외한 나머지 스케줄러는 디스크기반의 HDD 에 특화돼있고, 디스크헤드의 움직임을 최소화하기 위해 엘리베이터 알고리즘을 기본으로 채택하고 있다. SSD 는 하드디스크와 달리 디스크헤드가 없기 때문에 엘리베이터 알고리즘은 SSD 를 위한 I/O 스케줄러에 불필요하다. 또한 SSD 는 플래시 메모리의 특성 및 SSD 고유의 특징들이 존재하기 때문에 SSD 의 특성을 살린 I/O 스케줄러가 필요하다.

b. 플래시메모리 및 SSD 의 특징

SSD 는 다수의 낸드 플래시 메모리 모듈로 구성되어 있다. Out-Place-Update 및 Wear Leveling 을 지원하기 위해 SSD 는 플래시 변환 계층(Flash Translation Layer, FTL)이라는 임베디드 시스템을 내장하고 있다. FTL 의 목표는 플래시 메모리를 병렬로 구성해서 성능을 극대화 하는 것이다[1]. 실제 물리 블록의 크기는 2KB 에서 128KB 로 다양하기 때문에 FTL 은 보통 1MB 이상으로 논리 블록을 정해 놓고 물리 블록과의 매핑관계를 정의한다. 또한 FTL 은 Garbage Collection 을 내부적으로 수행한다..

C. IRBW-RP 스케줄러

기존 연구자들은 SSD 를 위한 I/O 스케줄러로 쓰기 요청을 논리 블록 단위로 모아서 처리하는 스케줄러를 고안했다[5]. IRBW-RP 스케줄러의 주요 기능은 대역폭을 최대화하기 위해 쓰기 요청을 모아서 처리한다. 이 I/O 스케줄러는 Linux 기본 I/O 스케줄러에 비해 응답 시간의 증가 없이 대역폭을 향상시킬 수 있다.

이와 같은 IRBW-RP 스케줄러의 대역폭 개선에도 불구하고, 우리는 SSD 를 위한 I/O 스케줄러의 성능향상의 가능성을 찾았다. IRBW-RP 는 기본 Linux I/O 스케줄러에서 디스크 기반 저장장치를 위한 정렬 및 병합 프로세스를 제거하고 SSD 를 위한 기법을 추가함으로써 성능향상을 이끌어냈지만 응답 시간에 대한 개선이 관찰되지 않는다. 또한, 쓰기요청을 번들링할 때 쓰기 요청의 유형을 구분하지 않고 일괄적으로 처리함으로써 성능향상의 여지를 저해한다. 예를 들어, 파일 시스템 메타 데이터 및 O_DIRECT 표시가 있는 요청은 파일시스템 또는 응용프로그램 전체작업을 블록시킴으로써 전체 대기시간을 늘릴 수 있기 때문에 이와 같은 요청은 번들링하지 않고 빠르게 처리해줌으로써 전체 대기시간을 줄일 수 있다.

3. New I/O Scheduler; NSCHED

우리는 SSD 를 위한 새로운 I/O 스케줄러를 제안한다. 이 절에서 우리는 NSCHED 스케줄러의 특징을 살펴본다.

a. 기아방지 타이머

SSD 가 논리 블록과 물리 블록을 연결하기 위해 FTL 을 사용함으로써 논리블록의 사이즈는 SSD I/O 스케줄러의 성능향상을 위한 중요한 변수이다. IRBW-RP 스케줄러는 I/O 요청을 논리 블록 단위로 모아서 처리함으로써 성능향상을 이루어 내지만 특정 상황에서 이는 오히려 악영향을 줄 수 있다. 예를 들어 I/O 요청이 논리 블록 사이즈만큼 모이지 않았을 경우 논리 블록 사이즈 만큼 모일 때까지 요청의 처리가 무기한 연기 될 수 있다. 따라서 우리는 이러한 현상을 피하기 위해 각 요청에 타이머를 두고 일정 시간이 지나도 처리되지 않으면 바로 현재까지 모인 I/O 요청을 바로 처리될 수 있도록 한다.

b. 선택적 번들링

I/O 요청에는 파일시스템 메타데이터의 처리를 위한 요청 등 바로 처리되지 않으면 전체 시스템을 블록시킬 수 있는 I/O 요청이 존재한다. 따라서 우리는 I/O 요청을 두 가지 그룹으로 구분하여 다르게 처리한다. 전체 시스템을 블록시킬 수 있는 요청을 동기식 I/O 요청으로 정의하고 번들링하지 않고 요청이 들어오는 즉시 처리한다. 그 외의 I/O 요청을 비동기식 I/O 요청으로 정의하고 논리 블록 사이즈로 번들링하여 처리한다. 이러한 기법은 다양항 I/O 요청 타입에 대해서 대기시간을 일정수준으로 유지하며 전체 시스템이 블록되는 상황을 피할 수 있도록 한다.

<표 1> 각 명령별 평균 및 최대 응답시간 비교

평균	읽기	쓰기	열기	생성
Noop	211.324	12.28	33.61	850.892
Deadline	248.15	11.608	38.442	1344.042
Anticipatory	135.112	38.188	410.518	16136.47
CFQ	290.9	166	803.406	26815.02
NSCHED	**158.986**	11.716	42.364	**634.754**
최대	read	write	open	create
Noop	133703	1944140	66224	235195
Deadline	98966	345291	90511	1153734
Anticipatory	308560	1155450	442097	882439
CFQ	101193	3069883	1422038	1973431
NSCHED	**94774**	403423	80121	**165415**

* 나노 초.

우리는 I/O 요청의 처리방법을 크게 세 가지로 나눈다. 첫 째, I/O 요청이 논리 블록 크기에 도달하면 I/O 요청을 처리한다. 둘째, 동기식 I/O 요청인 경우 바로 처리한다. 마지막으로, 우리는 각 요청에 타이머를 설정하여 일정 시간이 지나면 바로 처리하도록 한다. 이 타이머는 각 I/O 요청이 논리 블록 크기만큼 모이는 동안 무기한 기다리는 현상을 방지한다.

4. 실험 결과

우리는 Linux 2.6.30 에 NSCHED 스케줄러를 구현했다. 기본 Linux I/O 스케줄러와의 성능비교를 위해 우리는 Postmark 1.5 벤치마크 프로그램을 통하여 성능을 측정했다.

<표 1> 은 각각의 다양한 작업 동안의 평균 및 최대 응답 시간을 보인다. 파일 읽기 및 쓰기, 열기, 생성 요청 동안의 응답시간을 측정한 것으로 파일 열기 및 쓰기 명령은 매우 긴 대기시간을 보인다. NSCHED 스케줄러에서는 읽기 요청이 기본 Linux I/O 스케줄러보다 빨리 처리되기 때문에 응답 시간이 다른 I/O 스케줄러에 비해 빠르다. 또한, 파일 생성의 경우 파일시스템 메타데이터에 관한 I/O 요청이기 때문에 전체 시스템을 블록시킬 수 있는 동기식 I/O 요청이다. NSCHED 스케줄러는 이러한 동기식 I/O 요청을 즉시 처리함으로써 더 나은 응답시간을 보인다.

5. 결 론

이 논문에서 우리는 기존의 다른 Linux I/O 스케줄러보다 나은 Bandwidth 와 더 낮은 대기시간을 갖는 SSD 용 I/O 스케줄러를 고안했다. 이를 위해 우리는 기아방지 타이머 및 선택적 번들링 기법을 고안했다. 기아방지 타이머는 각 I/O 요청들이 논리 블록 크기만큼 모이는 동안 무기한 블록될 수 있는 상황을 방지한다. 또한, 선택적 번들링 기법은 I/O 요청을 동기식 요청과 비동기식 요청으로 구분하고 동기식 요청의 경우 바로 처리해줌으로써 시스템 전체가 블록되는 상황을 방지한다. 실험 결과 우리는 NSCHED 스케줄러가 기본 Linux 스케줄러보다 약 30 나은 성능을 보여준다.

참고문헌

[1] Agrawal, N., V. Prabhakaran, et al. “Design tradeoffs for SSD performance,” USENIX Association, 2008.
[2] Chen, F., D. Koufaty, et al. "Understanding intrinsic characteristics and system implications of flash memory based solid state drives", ACM, 2009.
[3] Soundararajan, G., V. Prabhakaran, et al., "Extending SSD Lifetimes with Disk-Based Write Caches", FAST, 2010.
[4] Dunn, M. and A. Reddy. "A new I/O scheduler for solid state devices." Texas A&M University ECE Technical Report TAMU-ECE-2009-02, 2009.
[5] Kim, J., Y. Oh, et al. "Disk schedulers for solid state drivers", ACM, 2009.