리눅스 I/O 스케줄러에 대한 SSD 성능 분석

박현찬 ○ 유 혁

고려대학교 정보통신대학

{hcpark, chuckyoo}@os.korea.ac.kr

A performance analysis of Solid State Disk for Linux I/O scheduler

Hyunchan, Park○ Chuck Yoo

Department of Computer Science and Engineering

요 약

SSD는 뛰어난 성능으로 인해 서버 시장에서 HDD를 빠르게 대체하며 각광받고 있다. 우리는 기존 SSD의 성능 분석이 단일한 I/O 패턴에 대해서만 이루어진 점을 주목하여, 다양한 패턴의 I/O가 동시에 수행될 경우, 성능에 어떠한 영향이 있는지 평가해보고자 한다. 이를 위해 4KB부터 64MB까지 다양한 블록 크기로 순차적/임의적 읽기/쓰기 연산을 수행함과 동시에 4KB 단위의 읽기/쓰기 I/O를 수행시켜 성능에 미치는 영향을 알아보았다. 이러한 평가를 네 가지 리눅스 I/O 스케줄러에 대해 각각 수행함으로써 스케줄러에 의한 영향 또한 평가하였다. 그 결과로 우리는 새로운 SSD의 성능 특성을 발견할 수 있었으며, 이는 새로운 I/O 스케줄러 및 SSD의 FTL 개발의 기반이 되리라 예상된다.

1. 서 론

SSD(Solid State Disk)는 플래시 메모리 모듈을 기반으로 한 차세대 저장장치로서 디스크 회전 방식의 HDD(Hard Disk Drive)를 대체하는 차세대 스토리지로 주목받고 있다. SSD는 플래시 메모리를 이용해 HDD의 물리적 특성으로 인한 성능, 전력 소모, 진동 및 소음 등의 한계를 극복함으로써 많은 컴퓨팅 분야에 적용되고 있다. 그러나 용량에 비해 높은 가격으로 인해 아직 개인용 컴퓨터 시장에서는 널리 쓰이지 못하고, 서버 시장에서는 서버 시스템의 I/O 성능을 극대화하기 위해 SSD를 활용하는 경우가 많이 늘어나고 있다.

그러나 서버 환경에서 많은 클라이언트의 다양한 I/O 패턴이 동시에 요청됨으로써 SSD의 성능을 최대로 활용할 수 없는 가능성이 있다. SSD는 다양한 I/O 패턴에 대해 HDD보다 크게 높은 성능을 보이지만, 내부적으로 여러 개의 NAND 플래시 칩을 장착하고 이에 대한 병렬적 I/O를 통해 성능을 극대화하는 구조[1]이기 때문에 병렬성을 활용할 수 없는 4KB 이하의 적은 크기의 블록에 대한 I/O 요청의 성능은 상대적으로 낮다. 따라서 이러한 4KB 이하 크기의 I/O 요청이 많은 경우, SSD의 전체적인 성능이 크게 저하될 수 있다.

본 논문에서는 이러한 성능 하락의 가능성을 구체적으로 분석해보기 위하여 서로 다른 형태의 I/O 연산이 동시에 수행되는 경우의 SSD 성능을 분석한다. 4KB 부터 64MB까지 다양한 블록 크기에 대해 임의적/순차적인 읽기/쓰기가 수행될 때, 동시에 4KB 크기의 임의적 읽기/쓰기를 수행하여 기본적인 SSD 성능과의 차이점을 파악한다. 또한 이러한 동작을 리눅스에 기본으로 탑재된 네 개의 스케줄러인 완전 공평 큐잉(이하 CFQ: Complete Fair Queuing), 예측(Anticipatory), 데드라인(Deadline), 무동작(No-operation) 스케줄러들에 대해 각각 수행하고 비교함으로써 스케줄러에 의한 영향도 측정하였다.

우리는 위와 같은 실험의 결과로 여러 I/O 패턴이 동시에 수행되는 경우, 다음과 같은 SSD 성능 특징을 새로이 발견하였다.

- 순차 읽기 연산에 대해, 적은 크기의 I/O에 의해 번들링(bundling)이 취소됨으로써 성능이 저하됨
- 임의 읽기와 순차 읽기가 동시에 수행되는 경우, 임의 읽기에 의해 순차 읽기의 성능이 하락함
- 읽기/쓰기 연산에 대해 1MB 이상 블록 크기의 I/O 요청은 성능이 하락됨 - 단일한 I/O 요청에 대해서는 블록 크기가 클수록 성능은 증가함

이러한 성능 관찰 내용은 다양한 I/O 패턴에 보다 좋은 성능을 보이는 새로운 I/O 스케줄러를 설계하거나 SSD의 FTL(Flash Translation Layer)를 발전시키는 토대가 될 것이다.

본 논문에서는 위와 같은 실험 내용과 그에 대한 성능 분석을 상세히 기술한다. 우선 2장에서는 SSD의 구조와 관련 연구에 대해 기술하고, 3장에서는 실험의 설계 내용을 기술한다. 4장에서는 실험 결과를 보이고 그에 대한 분석을 수행하고, 5장에서는 결론을 맺는다.

2. 배경 기술 및 관련 연구

2. 1. SSD의 구조

SSD는 그림 1과 같은 구조로 이루어져 있다. 핵심이 되는 모듈은 수 개의 NAND 플래시 모듈로서 이를 병렬로 연결함으로써 높은 성능을 보인다. 이와 동시에 NAND 플래시 모듈의 주요한 특징인 Wear-leveling, Out-place update등을 지원하기 위해 FTL이 필요하고, 이를 구동하기 위하여 CPU, SRAM, SDRAM등으로 구성된 임베디드 시스템이 탑재되어 있다. 이 시스템이 주로 수행하는 역할은 첫째, I/O 요청을 전달받아 이를 병렬적으로 구동하기 위해 플래시 칩을 제어하고, 둘째, 동시에 Wear-leveling 특성과 Out-place update 특성을 지원하기 위해 논리적 블록에 대해 행해지는 요청을 여러 물리적 블록으로 매핑하여 처리하는 FTL의 역할을 수행한다. 이러한 FTL의 동작에는 한 플래시 블록 내의 유효한 페이지와 유효하지 않은 페이지를 구분하여 정리하는 가비지 콜렉션 동작도 포함된다. 그리고 FTL은 매핑 사이즈로 사용되는 논리 블록(LB: Logical Block)을 정의한다. 이 블록은 주로 2KB/4KB 크기의 페이지와 64KB/128KB 크기를 갖는 플래시 블록을 모두 포함하기 위해 보통 1MB 이상의 크기로 정의된다.

SSD는 단순히 여러 개의 플래시 모듈이 장착된 구조가 아니라, 이렇게 복잡한 역할을 수행하는 S/W 레이어인 FTL을 탑재하고 있기 때문에 각 제조사의 FTL에 따라 다양한 성능 특징을 보인다. 이러한 FTL은 최근 대단히 많은 발전을 이루고 있으며 그로 인해 기존 SSD의 성능을 분석한 결과와 최근에 생산된 SSD의 성능은 차이가 있다. [2],[3] 논문에서는 삼성과 MTRON 등의 SSD를 분석하였지만 본 논문에서 실험한 인텔의 최신 SSD와는 그 특징에 차이가 있음을 알 수 있었다. 그러나 기본적으로 병렬성을 극대화하고 NAND 플래시의 특성을 지원한다는 점에서는 동일하며 우리는 공통적인 특성으로 인해 발생되는 I/O 성능의 특징을 분석하고자 노력하였다.

2. 2 리눅스 I/O 스케줄러

우리는 네 가지 리눅스 I/O 스케줄러에 대해 실험을 수행한다. 각 스케줄러의 특징에 대해 간략히 알아보면 다음과 같다.

- CFQ: 여러 프로세스에 대해 최대한 균등한 I/O 대역폭을 제공하는 것을 목표로 하는 스케줄러
- 예측: 여러 I/O 패턴이 동시에 수행될 때, 지역성으로 인한 HDD 헤드의 움직임을 최소화하기 위해 I/O 요청 처리 후 약간의 대기 시간을 두는 스케줄러
- 데드라인: 읽기/쓰기 연산에 대해 서로 다른 데드라인을 두고 데드라인 시간 내에서 요청을 최대한 효율적으로 정렬하여 처리하는 스케줄러

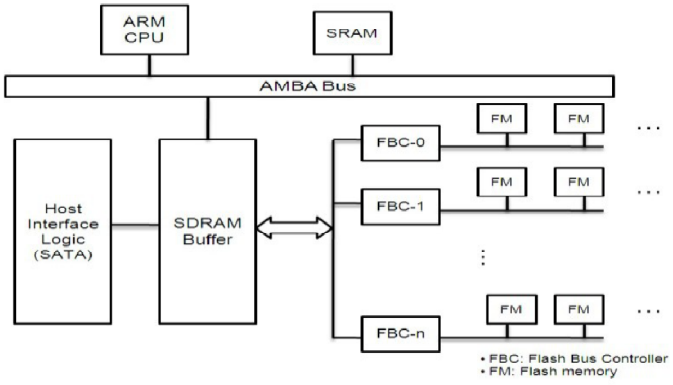


그림 1. SSD의 하드웨어적 구조

- 무동작: 단순한 선입선출 기반 큐만을 유지하며 I/O를 처리하는 스케줄러

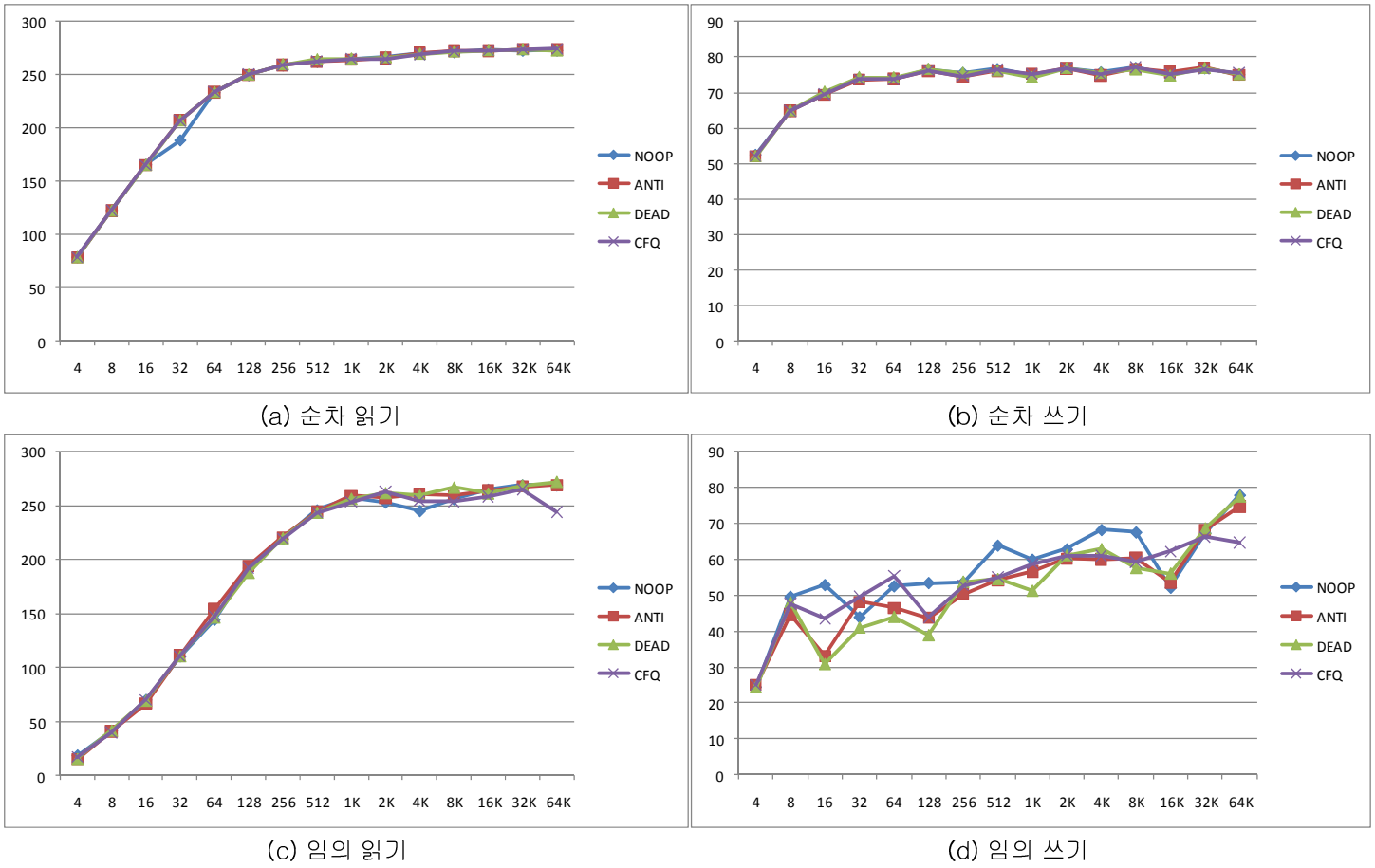
이러한 각각의 스케줄러의 특징에 따라 HDD에서도 전체적인 I/O 수행 성능은 많은 변화를 보인다. 우리는 HDD의 특징에 맞게 설계된 스케줄러들이 SSD에서는 어떠한 성능 특성을 보이는지 분석하고자 한다.

2. 3 기존 논문의 I/O 성능 분석

기존 SSD에 대한 다양한 논문에서 I/O 성능에 대한 분석이 이루어졌지만, 특히 우리는 다양한 I/O 스케줄러에 대해 평가가 이루어진 논문[2][3]을 주목하였다. 그 이유는 SSD의 FTL에 직접 I/O 요청을 수행하는 것이 I/O 스케줄러이기 때문이다. 서버 시스템에서는 파일 시스템도 여러 종류가 동시에 수행될 수 있고, 이와 함께 데이터베이스도 동작한다. 이러한 여러 시스템에서 요청하는 I/O들은 결국 단일한 I/O 스케줄러에서 모여 처리되기 때문에 I/O 스케줄러에 따른 성능을 분석하는 것이 옳다.

[2]에서는 FTL의 논리 블록 단위를 벗어나는 I/O 요청이 성능의 저하를 일으킬 것이라는 가정을 하고 SSD의 성능을 분석하였다. 여러 SSD에 대해 논리 블록 단위를 측정하고 블록의 경계에 일치하지 않는, 즉 여러 개의 블록에 걸쳐서 수행되는 I/O 연산의 성능이 저하된다는 점을 관찰하였다. 이러한 관찰을 토대로 논리 블록의 크기에 맞춘 I/O 요청을 수행하는 스케줄러를 디자인하였지만 전체적인 성능 향상이 뚜렷하게 관찰되지는 않았다.

[3]에서는 쓰기 연산을 번들링하고 읽기 연산에 대해 보다 높은 우선순위를 주는 IRBW-RP(Individual Read Bundled Write FIFO with Read Preference) I/O 스케줄러를 제안하였다. 이것은 논리 블록의 크기에 맞춘 쓰기 연산의 성능이 좋다는 실험 결과에서 도출된 정책으로, SSD에서 다양한 I/O 패턴에 대해 좋은 결과를 나타내었다. 그러나 단일한 I/O 프로세스에 대해 주로 평가가 이루어졌으며 다양한 I/O 패턴이 동시에 수행되는 경우에 대해서 자세한 분석이 이루어지지 않았다.



(a) 순차 읽기 (b) 순차 쓰기

(c) 임의 읽기 (d) 임의 쓰기

그림 2. 블록 크기에 따른 순차적, 임의적 읽기 쓰기 기본 성능 - X축: 블록 크기(KB), Y축: 입출력 성능(MB/s)

3. SSD 성능 분석 실험의 설계

우리는 SSD의 성능 분석 실험을 위해 인텔 X25-M G2 MainStream SATA SSD를 사용하였다. 이 SSD는 최근 발표된 SSD 중 가장 좋은 4K 읽기/쓰기 성능을 가지고 있는 것으로 인정받고 있다. 또한 최대 I/O 대역폭도 읽기 약 250 MB/s, 쓰기 약 70 MB/s로서 대단히 빠른 성능을 보여준다. 이러한 대역폭을 최대로 지원하기 위해 우리는 SATA2 방식의 I/O 인터페이스를 사용하였다. SATA1의 경우 150MB/s가 최대 지원 대역폭이기 때문에 최대의 읽기 속도를 발휘할 수 없다. 기타 CPU, 메모리 등의 하드웨어 구성요소들도 SSD의 성능을 최대로 발휘하는데 부족함이 없도록 구성하였다.

우리는 실험에서 읽기/쓰기 요청에 대한 성능(bandwidth)을 측정하였고 이때 세 가지 변인을 설정하였다. 첫째로 4KB 부터 64MB까지 두 배씩 증가하는 파일 I/O 블록 크기이다. 실험에서 사용되는 유저 프로그램에서 우리는 표준 파일 I/O 입출력을 사용하였으며, 이때 한 번에 요청하는 I/O 크기를 다양한 크기로 설정하였다. 둘째로 네 가지 리눅스 I/O 스케줄러를 사용하였고, 셋째로 4KB의 임의적 읽기와 쓰기를 요청하는 두 개의 프로세스를 각각 동시에 수행하였다. 이러한 프로세스를 우리는 4kReader, 4kWriter라 지칭한다. 시스템에서는 총 두 개의 I/O 프로세스가 동작하게 되며, 성능 측정의 대상이 되는 기본 I/O 프로세스가 하나 동작하고 동시에 4kReader 혹은 4kWriter가 하나 수행된다.

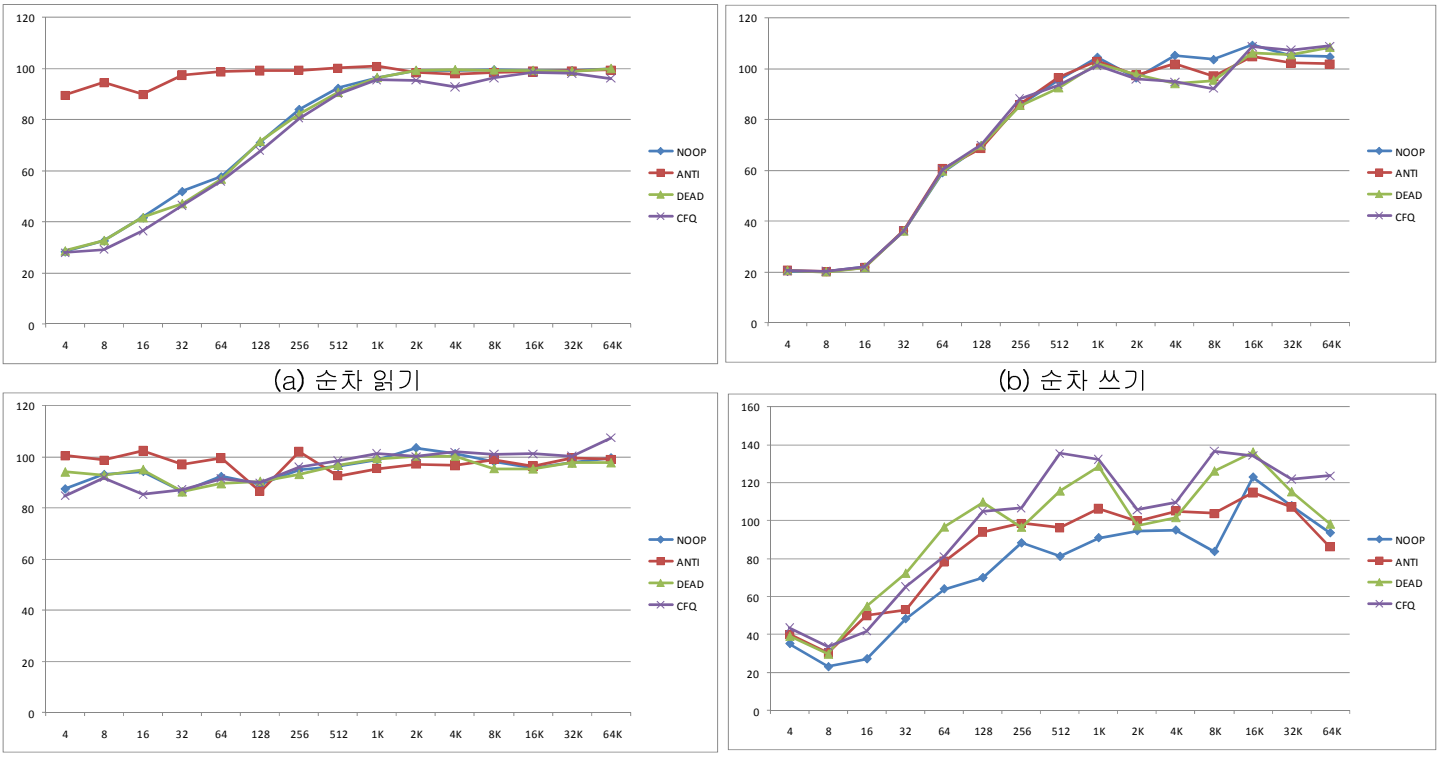
전체적인 공통사항으로는 임의적인 I/O에 있어 그 시작 주소나 크기를 결정하는 단위는 4KB 단위로 정렬하였다. 또한 모든 I/O 연산에 O_DIRECT 옵션을 사용하여 버퍼 캐쉬의 영향을 배제하였다.

위와 같은 실험을 통해 얻고자 하는 결과는 적은 크기의 I/O가 동시에 I/O 스케줄러에서 스케줄링 됨으로써 각 스케줄러의 정책과 블록 크기에 따라 어떤 성능 변화를 보이는지 알아보고, 이를 통해 새로운 I/O 스케줄러 디자인 정책을 도축하는 것이다. 본 논문에서는 우선 위 실험 설계에 따른 I/O의 성능 측정 결과를 보이고 이를 분석하고자 한다.

4. 실험 결과 및 분석

4.1 기본 성능

우리는 우선 SSD의 기본적인 성능을 파악하기 위해 4kReader, 4kWriter가 없는 경우의 성능을 실험해보았다. 그림 2는 그 결과를 나타내고 있으며 대체로 하드웨어



(a) 순차 읽기 (b) 순차 쓰기

(c) 임의 읽기 (d) 임의 쓰기

그림 3. 임의 4KB 읽기와 동시 수행 시 성능 – X축: 블록 크기(KB), Y축: 기본 성능에 대한 비교(%)

명세와 유사한 성능을 보이고 있다. 그리고 단일한 I/O를 수행하는 경우에 각 스케줄러 별로 거의 차이가 없는 성능을 보이고 있으며, 임의 쓰기에서만 성능이 불안정한 모습을 보이지만 각 스케줄러에 따른 특징을 파악할 수는 없었다. 임의 읽기의 성능을 보면 실험 대상인 인텔 SSD의 논리 블록 크기를 짐작할 수 있는데, 1MB부터 거의 동일한 최대 성능을 보이는 것으로 보아 논리 블록 크기가 1MB라는 점을 알 수 있다. 전체적인 성능면에서는 블록 크기가 적을수록 성능이 크게 저하된다는 SSD의 특징을 다시 한 번 파악할 수 있다.

우리는 이러한 기본 성능을 토대로 하여 4kReader, 4kWriter와 동시에 수행한 경우의 성능을 비교하였다. 차후의 실험 결과 그래프는 Y축을 위 기본 성능에 대한 퍼센티지로 나타낼 것이다.

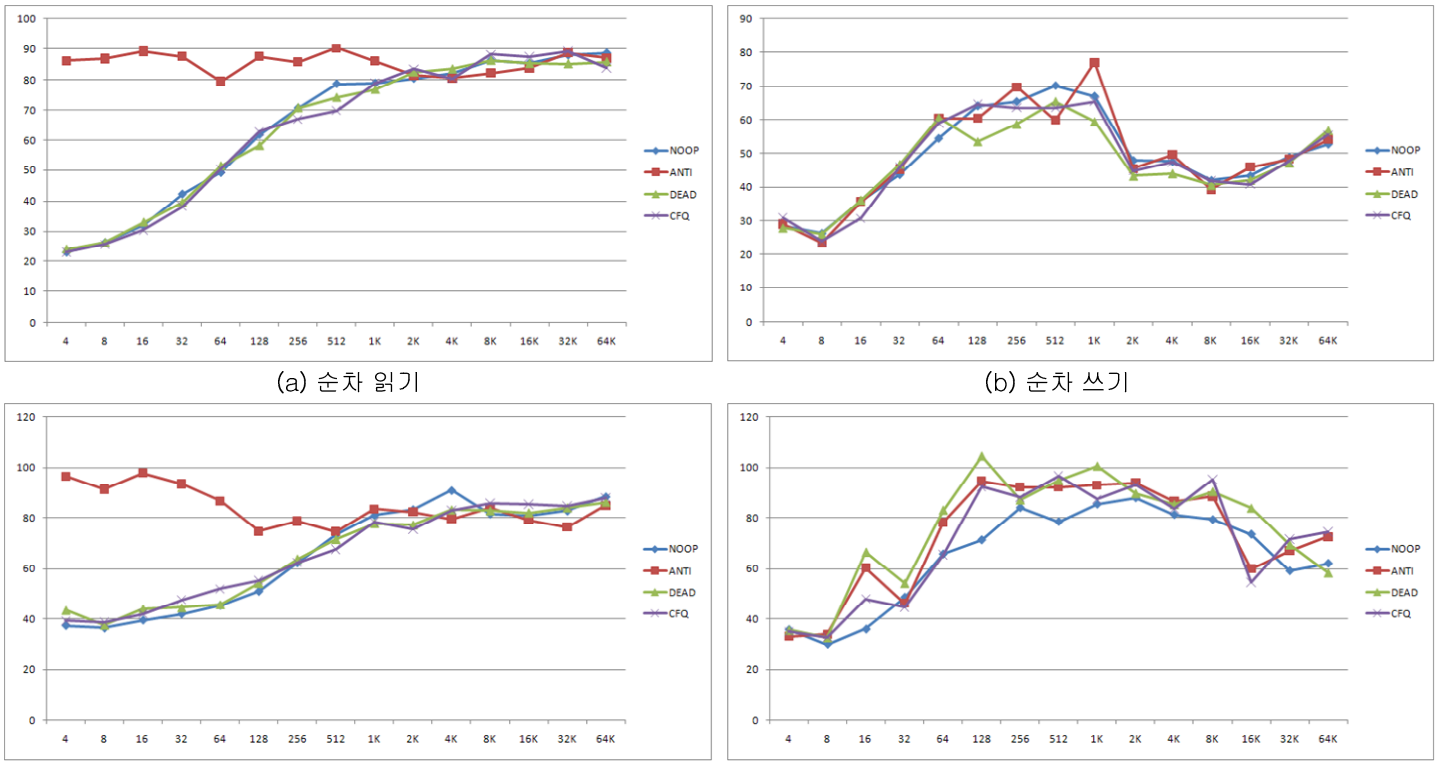
4.2 4K 임의 읽기와 동시 수행 시 성능

이 절에서는 4kReader와 함께 수행한 경우의 성능을 분석해본다. 그림 3이 그 결과를 나타낸 것이다. 순차 읽기의 경우, 예측 스케줄러만이 기존 성능을 거의 그대로 보여주고 있음을 알 수 있다. 다른 스케줄러들은 4KB 블록 크기인 경우 25%까지 저하된 성능을 보이는데, 4kReader와 동시에 수행중임을 감안하면 결국 두 개 이상의 I/O 요청 프로세스의 수행으로 인해 전체 I/O 성능이 절반 이하로 줄어들었음을 알 수 있다. 이러한 상황에서 예측 스케줄러는 순차 읽기 요청을 대기 시간 동안 번들링하여 처리함으로써 최대한의 I/O 성능을 보이고 있다고 판단할 수 있다. 순차 쓰기의 경우, 16KB 블록 까지 거의 20%의 성능을 보이고 있다. 순차 읽기와 쓰기 모두 1MB 이상의 블록 크기에 대해서만 모든 스케줄러가 거의 100%의 성능을 보이는데, 이로써 논리 블록의 크기에 맞게 I/O를 요청하면 다른 I/O 프로세스의 수행 여부에 관계없이 최대의 성능을 끌어낼 수 있다는 점을 알 수 있다.

임의 읽기의 경우는 4kReader 또한 임의 읽기이므로 단순히 대역폭을 분할해 사용한다는 면 외에 큰 성능 차이는 없다. 임의 쓰기의 경우에는 4kReader로 인해 성능의 저하가 발생하는데 오히려 특정 구간에서는 성능이 40% 가량 더 좋아지는 현상도 발생하였다. 또한 특이할 점으로는 대부분의 스케줄러에서 32MB이상의 구간에서 성능이 오히려 약간 저하된다는 점 또한 알 수 있다. 그러나 임의 쓰기와 관련한 이러한 현상들에 대해서는 그 원인을 분석하기가 어렵다.

4.3 4K 임의 쓰기와 동시 수행 시 성능

이 절에서는 4kWriter와 함께 수행한 경우의 성능을 분석해본다. 그림 4가 그 결과를 나타낸 것이다. 우선 순차 읽기에서는 4kReader의 결과와 유사하게 예측 스케줄러만이 1MB 미만 구간에서 기본 성능과 비슷한 성능을 보이고 있다. 그러나 전체적으로 근소하게 낮은 성능이라는 점을 파악할 수 있는데, 이는 기본적으로 쓰기 성능이 읽기 성능에 비해 낮기 때문에 4kWriter를 함께 수행시켜 전체 성능이 저하된 것으로 파악할 수 있다. 순차 쓰기의 경우, 1MB 이상의 구간에서 갑자기



(a) 순차 읽기 (b) 순차 쓰기

(c) 임의 읽기 (d) 임의 쓰기

그림 4. 임의 4KB 쓰기와 동시 수행 시 성능 – X축: 블록 크기(KB), Y축: 기본 성능에 대한 비교(%)

크게 성능이 저하되는 현상이 나타났다. 이는 1MB 이상으로 이미 논리블록의 크기에 딱 맞게 요청되던 I/O들이 4kWriter로 인해 분할되는 효과를 얻었기 때문인 것으로 파악된다. 그 증거로 다시 8MB 이상으로 증가하면 블록이 분할되어 1MB 이하로 줄어들 확률이 감소하여 약간 성능이 증가한다는 점을 들 수 있다.

임의 읽기의 경우, 4kReader와의 수행 결과와 다르게 예측 스케줄러만이 적은 블록 크기에서 기본 성능과 유사한 성능을 보임을 알 수 있다. 그리고 전체적으로는 최대 80% 정도로 하락한 성능을 보이는데, 이 또한 4kWriter에 의해 전체적인 대역폭이 저하되었다고 파악할 수 있다. 임의 쓰기의 결과는 4kReader에서의 수행 결과와 유사한 형태를 보여준다. 역시 전체적으로 불안정하지만 작은 크기의 블록에서 큰 성능하락을 보여주며 16MB 이상의 블록에서도 성능하락이 있음을 알 수 있다.

5. 결론 및 향후 과제

우리는 SSD에서 다양한 형태의 I/O 요청이 동시에 수행될 경우의 성능을 알아보기 위해 다양한 블록 크기에 대해 임의적/순차적 읽기/쓰기 연산을 수행하며 동시에 4KB 단위의 임의 읽기/쓰기를 요청하는 프로세스를 수행하였다. 그 수행결과로 다음과 같은 SSD의 성능 특징을 새롭게 파악할 수 있었으며, 이것은 단일한 I/O 패턴에 대해서만 그 특징을 파악했던 결과와는 차이가 있다.

- 순차 읽기 연산에 대해, 적은 크기의 I/O에 의해 번들링(bundling)이 취소됨으로써 성능이 저하됨
- 임의 읽기와 순차 읽기가 동시에 수행되는 경우, 임의 읽기에 의해 순차 읽기의 성능이 하락함
- 읽기/쓰기 연산에 대해 1MB 이상 블록 크기의 I/O 요청은 성능이 하락됨 – 단일한 I/O 요청에 대해서는 블록 크기가 클수록 성능은 증가함

위와 같은 특징은 SSD를 위한 새로운 I/O 스케줄러 설계의 근거가 되고, 또한 SSD의 FTL을 작성하는데 있어 새로운 시각을 제시했다고 할 수 있다. 본 연구진은 이러한 결과를 토대로 현재 새로운 리눅스 I/O 스케줄러를 설계하고 있다.

6. 참고 문헌

[1] Agrawal, N., et al. “Design tradeoffs for SSD performance”. in USENIX Annual Technical Conference. 2008.

[2] Marcus Dunn and A. L. Narasimha Reddy, “A New I/O Scheduler for Solid State Devices”, Technical report TAMU-ECE-2009-02, 2009.

[3] Kim, J., et al. “Disk schedulers for solid state drivers”. EMSOFT, 2009.