운영체제 실습 Report

실습 제목: Assignment 5

실습일자: 2022년 11월 24일 (목)

제출일자: 2022년 12월 09일 (금)

학 과: 컴퓨터정보공학부

담당교수: 최상호 교수님

실습분반: 금요일 5,6교시

학 번: 2018202065

성 명: 박 철 준

● Introduction

■ 제목

Assignment 5

■ 과제 요구사항

- I/O Zone을 이용한 세 가지 Linux I/O scheduler 성능을 테스트한다.

1. Scheduler 별 성능 결과 분석한다.
2. 측정 하는 시점의 시스템 상황에 따라 성능이 달라질 수 있다.
3. 실험을 5회 이상 실시하여 얻은 평균값을 바탕으로 분석을 진행한다.
4. Scheduler 별 성능 결과 표 및 그래프 작성한다.
5. 결과에 대한 자신의 의견을 적는다.
6. IO Zone 수행 과정 캡처하여 보고서에 첨부한다.
7. 입력한 옵션 캡처 및 작성한다.

- Linux I/O scheduler의 종류는 다음과 같다.

1. noop
2. deadline
3. CFQ

- IO Zone은 다음과 같이 설정한다.

1. I/O 실험 결과에 대해 excel file로 작성한다.
2. File size : 1GB(변경 가능)
3. Buffer cache 거치지 않고 연산 수행한다.
4. record size(크기 변경하면서 실행)
5. 8KB / 16KB / 32KB / 64KB / 128KB / 256KB / 512KB / 8MB / 16MB를 전부 사용해야 한다.
6. thread or process : 1개
7. thread or process의 파일 경로 : ~/iozone_test

- 테스트 연산(4개 연산 수행)을 수행한다.

1. write/re-write
2. read/re-read
3. random-read/write
4. 1-3번 제외한 다른 연산 항목 선택해 사용한다. (선택한 이유에 대해 보고서에 작성할 것)

- 매 실험 전에 소스코드에 위치한 디렉토리에서 아래의 명령어를 수행한다. 캐시 및 버퍼를 비워서 실험에 영향을 주는 요소를 제거하기 위함이다.

1. rm –rf ~/iozone_test (Remove temporary directories and files)
2. sync (Linux command to flush file system buffer)
3. echo 3 | sudo tee /proc/sys/vm/drop_caches
   (Linux commands to free pagecache, dentries, and inodes)

● Conclusion & Analysis

■ 구현 및 분석방식

먼저 noop, deadlin, cfq 스케줄링 중 하나를 택하고 8KB / 16KB / 32KB / 64KB / 128KB / 256KB / 512KB / 8MB / 16MB의 경우 각각을 테스트 연산으로 각각 5번 수행하고 평균을 구하여 이를 noop, deadlin, cfq에 대한 표와 그래프를 구현하여 결과에 대해 비교하는 방식을 사용하였다. 또한, 4번째 연산은 fread를 선택하였고 이러한 이유는 fread는 intermediate buffer를 가지고 입출력을 하므로 일정한 크기의 데이터가 입력되거나 출력될 때까지, 물리적으로 입출력을 하지 않기 때문이다. 그래서, 데이터를 빈번하게 입출력하면 fread가 read보다 빠르므로 이러한 원리를 직접 테스트하기 위해 사용하였다.

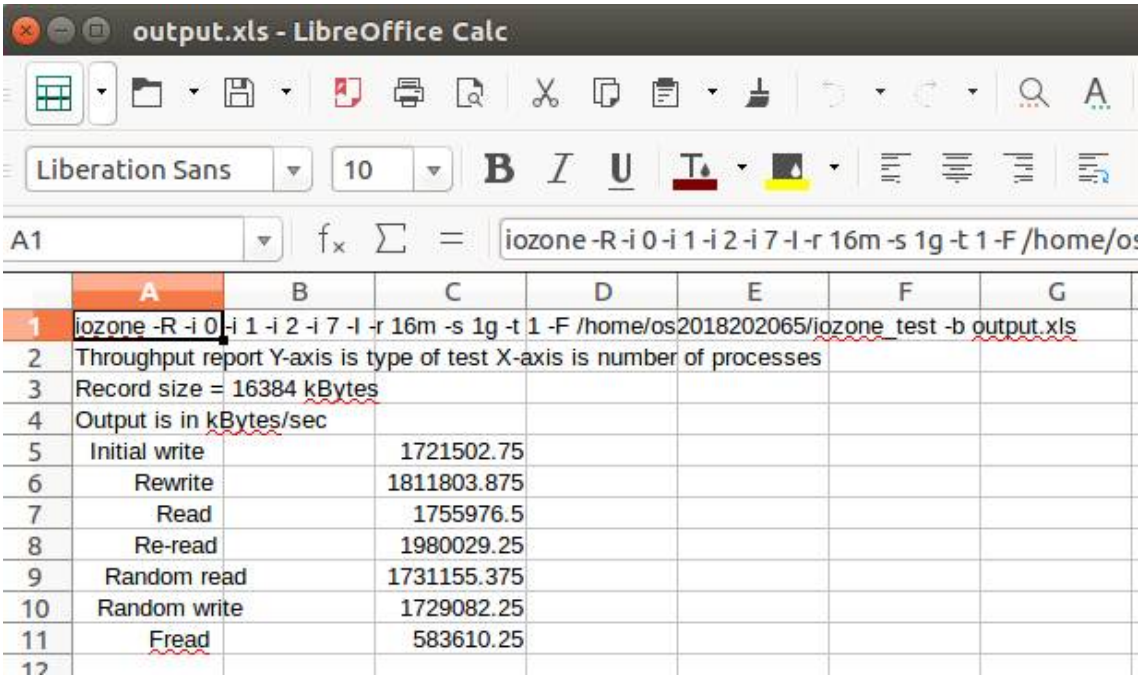
■ IOZone 수행 과정 캡처

(ex cfq 스케줄링으로 record size를 16MB로 하고 나머지 조건은 과제의 요구사항으로 설계)

```
os2018202065@ubuntu:~$ echo cfq | sudo tee /sys/block/sda/queue/scheduler
[sudo] password for os2018202065:
cfq
os2018202065@ubuntu:~$ cat /sys/block/sda/queue/scheduler
noop deadline [cfq]
os2018202065@ubuntu:~$ make
rm -rf ~/iozone_test
sync
echo 3 | sudo tee /proc/sys/vm/drop_caches
3
iozone -R -i 0 -i 1 -i 2 -i 7 -I -r 16m -s 1g -t 1 -F ~/iozone_test -b output.xls
        Iozone: Performance Test of File I/O
                Version $Revision: 3.429 $
                Compiled for 64 bit mode.
                Build: linux-AMD64

        Contributors:William Norcott, Don Capps, Isom Crawford, Kirby Collins
                     Al Slater, Scott Rhine, Mike Wisner, Ken Goss
                     Steve Landherr, Brad Smith, Mark Kelly, Dr. Alain CYR,
                     Randy Dunlap, Mark Montague, Dan Million, Gavin Brebner,
                     Jean-Marc Zucconi, Jeff Blomberg, Benny Halevy, Dave Boone,
                     Erik Habbinga, Kris Strecker, Walter Wong, Joshua Root,
                     Fabrice Bacchella, Zhenghua Xue, Qin Li, Darren Sawyer,
                     Vangel Bojaxhi, Ben England, Vikentsi Lapa.

        Run began: Sat Dec  3 06:39:56 2022

        Excel chart generation enabled
        O_DIRECT feature enabled
        Record Size 16384 kB
        File size set to 1048576 kB
        Command line used: iozone -R -i 0 -i 1 -i 2 -i 7 -I -r 16m -s 1g -t 1 -F /home/os2018202065/iozone_test -b output.xls
        Output is in kBytes/sec
        Time Resolution = 0.000001 seconds.
        Processor cache size set to 1024 kBytes.
        Processor cache line size set to 32 bytes.
        File stride size set to 17 * record size.
        Throughput test with 1 process
        Each process writes a 1048576 kByte file in 16384 kByte records

        Children see throughput for  1 initial writers  = 1721502.75 kB/sec
        Parent sees throughput for  1 initial writers   = 1693237.34 kB/sec
        Min throughput per process                      = 1721502.75 kB/sec
        Max throughput per process                      = 1721502.75 kB/sec
        Avg throughput per process                      = 1721502.75 kB/sec
        Min xfer                                        = 1048576.00 kB

        Children see throughput for  1 rewriters        = 1811803.88 kB/sec
        Parent sees throughput for  1 rewriters         = 1781762.31 kB/sec
        Min throughput per process                      = 1811803.88 kB/sec
        Max throughput per process                      = 1811803.88 kB/sec
        Avg throughput per process                      = 1811803.88 kB/sec
        Min xfer                                        = 1048576.00 kB
```

```
       Children see throughput for  1 readers          = 1755976.50 kB/sec
       Parent sees throughput for  1 readers          = 1725437.43 kB/sec
       Min throughput per process                     = 1755976.50 kB/sec
       Max throughput per process                     = 1755976.50 kB/sec
       Avg throughput per process                     = 1755976.50 kB/sec
       Min xfer                                       = 1048576.00 kB

       Children see throughput for 1 re-readers       = 1980029.25 kB/sec
       Parent sees throughput for 1 re-readers        = 1976301.22 kB/sec
       Min throughput per process                     = 1980029.25 kB/sec
       Max throughput per process                     = 1980029.25 kB/sec
       Avg throughput per process                     = 1980029.25 kB/sec
       Min xfer                                       = 1048576.00 kB

       Children see throughput for 1 random readers   = 1731155.38 kB/sec
       Parent sees throughput for 1 random readers    = 1728893.94 kB/sec
       Min throughput per process                     = 1731155.38 kB/sec
       Max throughput per process                     = 1731155.38 kB/sec
       Avg throughput per process                     = 1731155.38 kB/sec
       Min xfer                                       = 1048576.00 kB

       Children see throughput for 1 random writers   = 1729082.25 kB/sec
       Parent sees throughput for 1 random writers    = 1673130.80 kB/sec
       Min throughput per process                     = 1729082.25 kB/sec
       Max throughput per process                     = 1729082.25 kB/sec
       Avg throughput per process                     = 1729082.25 kB/sec
       Min xfer                                       = 1048576.00 kB

       Children see throughput for  1 freaders        =  583610.25 kB/sec
       Parent sees throughput for  1 freaders         =  582957.75 kB/sec
       Min throughput per process                     =  583610.25 kB/sec
       Max throughput per process                     =  583610.25 kB/sec
       Avg throughput per process                     =  583610.25 kB/sec
       Min xfer                                       = 1048576.00 kB
```

```
"Throughput report Y-axis is type of test X-axis is number of processes"
"Record size = 16384 kBytes "
"Output is in kBytes/sec"

"  Initial write " 1721502.75

"        Rewrite " 1811803.88

"           Read " 1755976.50

"        Re-read " 1980029.25

"    Random read " 1731155.38

"   Random write " 1729082.25

"          Fread "  583610.25


iozone test complete.
os2018202065@ubuntu:~$
```



올바른 실행결과가 나옴을 알 수 있다.

다음은 noop, deadline, cfq 스케줄링에 대한 record size를 변경하여 각각의 테스트 연산에 대한 5회 실행 결과에 대한 평균을 구하는 과정이다.

■ 입력한 옵션 캡처 (make 파일을 통해 진행)

▶ noop의 옵션

home > os2018202065 > M Makefile

```
noop_8k:
	echo noop | sudo tee /sys/block/sda/queue/scheduler
	cat /sys/block/sda/queue/scheduler
	rm -rf ~/iozone_test
	sync
	echo 3 | sudo tee /proc/sys/vm/drop_caches
	iozone -R -i 0 -i 1 -i 2 -i 7 -I -r 8k -s 1g -t 1 -F ~/iozone_test -b output.xls
noop_16k:
	echo noop | sudo tee /sys/block/sda/queue/scheduler
	cat /sys/block/sda/queue/scheduler
	rm -rf ~/iozone_test
	sync
	echo 3 | sudo tee /proc/sys/vm/drop_caches
	iozone -R -i 0 -i 1 -i 2 -i 7 -I -r 16k -s 1g -t 1 -F ~/iozone_test -b output.xls
noop_32k:
	echo noop | sudo tee /sys/block/sda/queue/scheduler
	cat /sys/block/sda/queue/scheduler
	rm -rf ~/iozone_test
	sync
	echo 3 | sudo tee /proc/sys/vm/drop_caches
	iozone -R -i 0 -i 1 -i 2 -i 7 -I -r 32k -s 1g -t 1 -F ~/iozone_test -b output.xls
noop_64k:
	echo noop | sudo tee /sys/block/sda/queue/scheduler
	cat /sys/block/sda/queue/scheduler
	rm -rf ~/iozone_test
	sync
	echo 3 | sudo tee /proc/sys/vm/drop_caches
	iozone -R -i 0 -i 1 -i 2 -i 7 -I -r 64k -s 1g -t 1 -F ~/iozone_test -b output.xls
noop_128k:
	echo noop | sudo tee /sys/block/sda/queue/scheduler
	cat /sys/block/sda/queue/scheduler
	rm -rf ~/iozone_test
	sync
	echo 3 | sudo tee /proc/sys/vm/drop_caches
	iozone -R -i 0 -i 1 -i 2 -i 7 -I -r 128k -s 1g -t 1 -F ~/iozone_test -b output.xls
noop_256k:
	echo noop | sudo tee /sys/block/sda/queue/scheduler
	cat /sys/block/sda/queue/scheduler
	rm -rf ~/iozone_test
	sync
	echo 3 | sudo tee /proc/sys/vm/drop_caches
	iozone -R -i 0 -i 1 -i 2 -i 7 -I -r 256k -s 1g -t 1 -F ~/iozone_test -b output.xls
noop_512k:
	echo noop | sudo tee /sys/block/sda/queue/scheduler
	cat /sys/block/sda/queue/scheduler
	rm -rf ~/iozone_test
	sync
	echo 3 | sudo tee /proc/sys/vm/drop_caches
	iozone -R -i 0 -i 1 -i 2 -i 7 -I -r 512k -s 1g -t 1 -F ~/iozone_test -b output.xls
noop_8m:
	echo noop | sudo tee /sys/block/sda/queue/scheduler
	cat /sys/block/sda/queue/scheduler
	rm -rf ~/iozone_test
	sync
	echo 3 | sudo tee /proc/sys/vm/drop_caches
	iozone -R -i 0 -i 1 -i 2 -i 7 -I -r 8m -s 1g -t 1 -F ~/iozone_test -b output.xls
noop_16m:
	echo noop | sudo tee /sys/block/sda/queue/scheduler
	cat /sys/block/sda/queue/scheduler
	rm -rf ~/iozone_test
	sync
	echo 3 | sudo tee /proc/sys/vm/drop_caches
	iozone -R -i 0 -i 1 -i 2 -i 7 -I -r 16m -s 1g -t 1 -F ~/iozone_test -b output.xls
```

▶ deadline의 옵션

```
deadline_8k:
    echo deadline | sudo tee /sys/block/sda/queue/scheduler
    cat /sys/block/sda/queue/scheduler
    rm -rf ~/iozone_test
    sync
    echo 3 | sudo tee /proc/sys/vm/drop_caches
    iozone -R -i 0 -i 1 -i 2 -i 7 -I -r 8k -s 1g -t 1 -F ~/iozone_test -b output.xls
deadline_16k:
    echo deadline | sudo tee /sys/block/sda/queue/scheduler
    cat /sys/block/sda/queue/scheduler
    rm -rf ~/iozone_test
    sync
    echo 3 | sudo tee /proc/sys/vm/drop_caches
    iozone -R -i 0 -i 1 -i 2 -i 7 -I -r 16k -s 1g -t 1 -F ~/iozone_test -b output.xls
deadline_32k:
    echo deadline | sudo tee /sys/block/sda/queue/scheduler
    cat /sys/block/sda/queue/scheduler
    rm -rf ~/iozone_test
    sync
    echo 3 | sudo tee /proc/sys/vm/drop_caches
    iozone -R -i 0 -i 1 -i 2 -i 7 -I -r 32k -s 1g -t 1 -F ~/iozone_test -b output.xls
deadline_64k:
    echo deadline | sudo tee /sys/block/sda/queue/scheduler
    cat /sys/block/sda/queue/scheduler
    rm -rf ~/iozone_test
    sync
    echo 3 | sudo tee /proc/sys/vm/drop_caches
    iozone -R -i 0 -i 1 -i 2 -i 7 -I -r 64k -s 1g -t 1 -F ~/iozone_test -b output.xls
deadline_128k:
    echo deadline | sudo tee /sys/block/sda/queue/scheduler
    cat /sys/block/sda/queue/scheduler
    rm -rf ~/iozone_test
    sync
    echo 3 | sudo tee /proc/sys/vm/drop_caches
    iozone -R -i 0 -i 1 -i 2 -i 7 -I -r 128k -s 1g -t 1 -F ~/iozone_test -b output.xls
deadline_256k:
    echo deadline | sudo tee /sys/block/sda/queue/scheduler
    cat /sys/block/sda/queue/scheduler
    rm -rf ~/iozone_test
    sync
    echo 3 | sudo tee /proc/sys/vm/drop_caches
    iozone -R -i 0 -i 1 -i 2 -i 7 -I -r 256k -s 1g -t 1 -F ~/iozone_test -b output.xls
deadline_512k:
    echo deadline | sudo tee /sys/block/sda/queue/scheduler
    cat /sys/block/sda/queue/scheduler
    rm -rf ~/iozone_test
    sync
    echo 3 | sudo tee /proc/sys/vm/drop_caches
    iozone -R -i 0 -i 1 -i 2 -i 7 -I -r 512k -s 1g -t 1 -F ~/iozone_test -b output.xls
deadline_8m:
    echo deadline | sudo tee /sys/block/sda/queue/scheduler
    cat /sys/block/sda/queue/scheduler
    rm -rf ~/iozone_test
    sync
    echo 3 | sudo tee /proc/sys/vm/drop_caches
    iozone -R -i 0 -i 1 -i 2 -i 7 -I -r 8m -s 1g -t 1 -F ~/iozone_test -b output.xls
deadline_16m:
    echo deadline | sudo tee /sys/block/sda/queue/scheduler
    cat /sys/block/sda/queue/scheduler
    rm -rf ~/iozone_test
    sync
    echo 3 | sudo tee /proc/sys/vm/drop_caches
    iozone -R -i 0 -i 1 -i 2 -i 7 -I -r 16m -s 1g -t 1 -F ~/iozone_test -b output.xls
```

▶ cfq의 옵션

```
cfq_8k:
    echo cfq | sudo tee /sys/block/sda/queue/scheduler
    cat /sys/block/sda/queue/scheduler
    rm -rf ~/iozone_test
    sync
    echo 3 | sudo tee /proc/sys/vm/drop_caches
    iozone -R -i 0 -i 1 -i 2 -i 7 -I -r 8k -s 1g -t 1 -F ~/iozone_test -b output.xls
cfq_16k:
    echo cfq | sudo tee /sys/block/sda/queue/scheduler
    cat /sys/block/sda/queue/scheduler
    rm -rf ~/iozone_test
    sync
    echo 3 | sudo tee /proc/sys/vm/drop_caches
    iozone -R -i 0 -i 1 -i 2 -i 7 -I -r 16k -s 1g -t 1 -F ~/iozone_test -b output.xls
cfq_32k:
    echo cfq | sudo tee /sys/block/sda/queue/scheduler
    cat /sys/block/sda/queue/scheduler
    rm -rf ~/iozone_test
    sync
    echo 3 | sudo tee /proc/sys/vm/drop_caches
    iozone -R -i 0 -i 1 -i 2 -i 7 -I -r 32k -s 1g -t 1 -F ~/iozone_test -b output.xls
cfq_64k:
    echo cfq | sudo tee /sys/block/sda/queue/scheduler
    cat /sys/block/sda/queue/scheduler
    rm -rf ~/iozone_test
    sync
    echo 3 | sudo tee /proc/sys/vm/drop_caches
    iozone -R -i 0 -i 1 -i 2 -i 7 -I -r 64k -s 1g -t 1 -F ~/iozone_test -b output.xls
cfq_128k:
    echo cfq | sudo tee /sys/block/sda/queue/scheduler
    cat /sys/block/sda/queue/scheduler
    rm -rf ~/iozone_test
    sync
    echo 3 | sudo tee /proc/sys/vm/drop_caches
    iozone -R -i 0 -i 1 -i 2 -i 7 -I -r 128k -s 1g -t 1 -F ~/iozone_test -b output.xls
cfq_256k:
    echo cfq | sudo tee /sys/block/sda/queue/scheduler
    cat /sys/block/sda/queue/scheduler
    rm -rf ~/iozone_test
    sync
    echo 3 | sudo tee /proc/sys/vm/drop_caches
    iozone -R -i 0 -i 1 -i 2 -i 7 -I -r 256k -s 1g -t 1 -F ~/iozone_test -b output.xls
cfq_512k:
    echo cfq | sudo tee /sys/block/sda/queue/scheduler
    cat /sys/block/sda/queue/scheduler
    rm -rf ~/iozone_test
    sync
    echo 3 | sudo tee /proc/sys/vm/drop_caches
    iozone -R -i 0 -i 1 -i 2 -i 7 -I -r 512k -s 1g -t 1 -F ~/iozone_test -b output.xls
cfq_8m:
    echo cfq | sudo tee /sys/block/sda/queue/scheduler
    cat /sys/block/sda/queue/scheduler
    rm -rf ~/iozone_test
    sync
    echo 3 | sudo tee /proc/sys/vm/drop_caches
    iozone -R -i 0 -i 1 -i 2 -i 7 -I -r 8m -s 1g -t 1 -F ~/iozone_test -b output.xls
cfq_16m:
    echo cfq | sudo tee /sys/block/sda/queue/scheduler
    cat /sys/block/sda/queue/scheduler
    rm -rf ~/iozone_test
    sync
    echo 3 | sudo tee /proc/sys/vm/drop_caches
    iozone -R -i 0 -i 1 -i 2 -i 7 -I -r 16m -s 1g -t 1 -F ~/iozone_test -b output.xls
```

■ 입력한 옵션에 대한 평균 계산

▶ noop

- noop (write) Output is in kBytes/sec

record size	1	2	3	4	5	평균
8KB	121860.60	140014.79	123249.04	124900.671	127586	127,522.22
16KB	178783.73	214576.15	222267.06	239830.55	240600.25	219,211.55
32KB	435464.78	395934.84	350540.28	354849.00	378687.56	383,095.29
64KB	679228.06	575546.44	564316.69	555590.75	704147.69	615,765.93
128KB	774501.00	791951.81	804711.44	795394.06	484285.56	730,168.77
256KB	1031216.50	1075482.50	1177914.62	982224.69	622392.25	977,846.11
512KB	1296617.38	1330165.38	1265544.88	1246127.00	1378780.25	1,303,446.98
8MB	1720277.25	1490326.38	1605790.00	1754716.12	1806553.38	1,675,532.63
16MB	1689619.25	605202.00	1611807.38	379626.03	1706741.25	1,198,599.18

- noop (Rewrite) Output is in kBytes/sec

record size	1	2	3	4	5	평균
8KB	121860.6	140014.79	119072.92	151497.34	125725.39	131,634.21
16KB	268921.12	219495.03	192663.9	253556.12	245721.98	236,071.63
32KB	418916.00	381796.25	406079.97	371025.38	399586.91	395,480.90
64KB	668200.31	589222.38	649105.44	601485.69	732025.00	648,007.76
128KB	756642.75	877887.44	854308.12	385352.72	439121.81	662,662.57
256KB	1118675.50	1325941.75	1158276.62	691324.81	713291.19	1,001,501.97
512KB	1378533.38	1182257.25	1413127.50	1552769.38	1486597.62	1,402,657.03
8MB	1822086.62	1650458.25	1721070.25	1776530.75	1778634.38	1,749,756.05
16MB	314893.06	596950.56	325856.62	914071.25	1214046.75	673,163.65

- noop (Read) Output is in kBytes/sec

record size	1	2	3	4	5	평균
8KB	137034.21	144685.85	131023.06	109967.24	69929.03	118,527.88
16KB	234531.18	244506.06	192663.9	163287.78	173360.30	201,669.84
32KB	188282.73	224159.17	248215.64	207339.39	465341.12	266,667.61
64KB	263894.59	298010.38	343979.44	329090.66	649308.06	376,856.63
128KB	466986.62	558715.19	564342.12	473187.50	459653.59	504,577.00
256KB	989194.69	1263900.25	1236028.62	691972.62	862857.12	1,008,790.66
512KB	1259899.00	1450631.62	1894730.75	1641056.12	1803812.62	1,610,026.02
8MB	2117501.75	1964718.12	1995792.88	2119701.00	2098726.50	2,059,288.05
16MB	1755294.00	2001011.38	1917844.50	2100428.25	1239592.12	1,802,834.05

- noop (Re-read) Output is in kBytes/sec

record size	1	2	3	4	5	평균
8KB	131488.56	144685.85	134960.2	114580.56	69929.03	119,128.84
16KB	50353	236971.09	121474.62	153600.88	186232.73	149,726.46
32KB	214094.98	321312.28	306600.81	303222.09	414096.00	311,865.23
64KB	280786.25	445233.84	482962.56	421827.72	674644.69	461,091.01
128KB	605579.06	565328.00	608340.62	572416.50	587601.12	587,853.06
256KB	798672.19	1259352.88	1167421.75	971559.62	703401.00	980,081.49
512KB	1303835.88	1067281.12	1690569.50	1617975.38	1820846.38	1,500,101.65
8MB	2153089.50	1960091.25	1911464.62	1957208.50	2045252.25	2,005,421.22
16MB	1967593.25	1966583.25	2078866.12	1994454.38	1844045.62	1,970,308.52

- noop (Random read) Output is in kBytes/sec

record size	1	2	3	4	5	평균
8KB	128677.67	127412.1	134960.2	105759.85	69929.03	113,347.77
16KB	150104.92	144138.2	78622.44	91490.58	157805.27	124,432.28
32KB	138942.02	153196.78	156250.55	171555.66	393838.16	202,756.63
64KB	231511.50	274924.75	269696.38	306876.19	746861.31	365,974.03
128KB	553929.31	637094.44	577296.81	821852.94	296272.59	577,289.22
256KB	866824.50	1016717.06	913283.94	1158638.50	1104399.25	1,011,972.65
512KB	1510144.00	1700547.12	1844385.88	1568522.25	1787030.75	1,682,126.00
8MB	2033787.38	1991620.12	2119439.50	2111812.00	1851859.12	2,021,703.62
16MB	2201124.50	1900145.00	1964246.88	1964972.62	1863270.50	1,978,751.90

- noop (Random write) Output is in kBytes/sec

record size	1	2	3	4	5	평균
8KB	119567.39	115007.04	122026.97	102439.9	69929.03	105,794.07
16KB	378881.03	209044.25	230752.12	201381.75	221384.81	248,288.79
32KB	437339.81	472591.25	423241.41	478950.09	454025.09	453,229.53
64KB	659247.94	624575.50	627414.94	591973.38	700875.75	640,817.50
128KB	808556.75	800960.56	848812.19	359962.81	719088.69	707,476.20
256KB	1104791.00	1197417.62	1171721.62	969307.88	653062.44	1,019,260.11
512KB	1182395.88	1431046.12	1643098.38	1166467.50	1486953.88	1,381,992.35
8MB	1596195.50	1503608.38	1547891.88	1746948.25	1782519.12	1,635,432.63
16MB	943317.75	1481106.62	1653023.88	1565723.50	1754489.38	1,479,532.23

- noop (Fread) Output is in kBytes/sec

record size	1	2	3	4	5	평균
8KB	550014.93	1153268.75	122026.976	102439.9	727057.875	530,961.69
16KB	790837.31	478026.28	264961.12	721235.88	837463.94	618,504.91
32KB	620690.56	1302640.62	1446367.62	1425718.62	1273485.00	1,213,780.48
64KB	706130.31	923506.50	1002786.56	1401890.38	1051609.00	1,017,184.55
128KB	912321.69	937055.88	944912.38	873472.62	1443611.75	1,022,274.86
256KB	1350192.88	1131848.38	1143712.12	627001.19	2150771.00	1,280,705.11
512KB	9156191.00	1431046.12	1722597.00	11468752.00	1722481.62	5,100,213.55
8MB	7604211.50	1833034.38	1946919.75	10801504.00	2033203.50	4,843,774.63
16MB	1364430.62	1298025.25	8435722.00	7884321.50	1993225.88	4,195,145.05

▶ deadline

- deadline (write) Output is in kBytes/sec

record size	1	2	3	4	5	평균
8KB	87649.67	87332.55	83776.07	102193.41	75671.23	87,324.59
16KB	179581.88	182217.84	183480.41	168269.41	233787.16	189,467.34
32KB	391457.59	364067.78	408018.56	290667.16	437924.81	378,427.18
64KB	681870.75	535437.12	708877.56	585411.69	723457.75	647,010.97
128KB	773799.75	867449.75	766139.44	760635.81	778872.56	789,379.46
256KB	1150042.00	1160737.12	1125412.50	1156559.25	1164150.62	1,151,380.30
512KB	1400705.00	1496859.88	1471041.62	1416020.62	1567898.12	1,470,505.05
8MB	1760265.12	1850972.25	1757763.38	1106010.00	1238148.75	1,542,631.90
16MB	1673889.25	1655281.00	1800878.88	1693009.88	1724833.25	1,709,578.45

- deadline (Rewrite) Output is in kBytes/sec

record size	1	2	3	4	5	평균
8KB	104221.46	108231.63	79329.68	75166.64	80095.98	89,409.08
16KB	209672.39	218221.22	167643.73	157962.30	160029.62	182,705.85
32KB	378550.97	419505.06	261361.22	284546.88	295586.75	327,910.18
64KB	573364.25	591642.00	648635.62	426441.94	527898.56	553,596.47
128KB	773493.81	827674.69	412716.16	537053.50	326290.19	575,445.67
256KB	1244655.38	1227096.62	1277125.38	1266247.75	335685.34	1,070,162.09
512KB	1508649.00	1527203.00	1583239.25	563041.25	1397145.62	1,315,855.62
8MB	1723911.00	1023319.62	1542500.00	537897.88	601244.25	1,085,774.55
16MB	1917532.62	1678634.25	1812056.88	651361.44	1795448.75	1,571,006.79

- deadline (Read) Output is in kBytes/sec

record size	1	2	3	4	5	평균
8KB	115621.57	115229.60	101086.96	113435.98	115467.77	112,168.38
16KB	235300.12	237563.77	167686.34	149533.34	197233.53	197,463.42
32KB	327461.50	377290.91	337299.00	275033.06	414557.88	346,328.47
64KB	637027.44	720296.06	404615.62	370049.75	524331.75	531,264.12
128KB	1105815.12	786336.00	530021.44	687552.50	416062.78	705,157.57
256KB	1308330.75	1480130.12	562642.25	1350454.12	803742.88	1,101,060.02
512KB	2045574.75	1791745.38	748592.94	1418927.12	1553480.75	1,511,664.19
8MB	1920225.88	1851230.88	644405.44	1885273.50	1556136.12	1,571,454.36
16MB	1462642.75	1987776.50	1953742.75	524791.50	1807331.38	1,547,256.98

- deadline (Re-read) Output is in kBytes/sec

record size	1	2	3	4	5	평균
8KB	100645.77	105224.64	109843.97	81351.39	124060.62	104,225.28
16KB	226920.06	233607.44	168614.39	166202.55	179340.11	194,936.91
32KB	367310.38	447857.53	303253.31	294923.16	355103.78	353,689.63
64KB	703827.31	748463.75	372310.09	506299.16	734109.75	613,002.01
128KB	970972.81	925006.19	605593.44	610756.81	854118.81	793,289.61
256KB	1417887.38	1411697.25	420126.12	485796.84	658148.50	878,731.22
512KB	1867385.25	1750123.75	578749.44	588421.81	1561712.12	1,269,278.47
8MB	1893707.62	1958593.38	1842004.38	1790625.62	593621.12	1,615,710.42
16MB	1684161.75	2026945.88	1800968.12	1255511.88	1942793.38	1,742,076.20

- deadline (Random read) Output is in kBytes/sec

record size	1	2	3	4	5	평균
8KB	104332.76	106135.22	84798.30	77866.16	106977.76	96,022.04
16KB	228061.22	227405.48	150482.62	176539.67	195477.88	195,593.37
32KB	441273.97	381798.62	317486.28	221921.61	273467.75	327,189.65
64KB	787414.25	798926.69	653528.81	390077.22	979418.06	721,873.01
128KB	945173.94	883157.12	332910.97	328981.81	760522.19	650,149.21
256KB	1452319.25	1425321.25	695166.56	747501.19	475643.69	959,190.39
512KB	1945670.00	1611049.88	811755.19	588735.06	991619.25	1,189,765.88
8MB	1970151.88	1918954.12	1815495.50	1908455.38	2016038.75	1,925,819.13
16MB	1897030.50	1969101.38	518485.31	569407.19	634473.50	1,117,699.58

- deadline (Random write) Output is in kBytes/sec

record size	1	2	3	4	5	평균
8KB	115336.62	105070.58	70493.55	108289.84	96428.51	99,123.82
16KB	195263.38	191502.02	120465.37	125654.58	172707.59	161,118.59
32KB	363640.50	364044.16	266726.97	250760.11	298437.25	308,721.80
64KB	694906.69	623378.69	403769.44	749866.12	651369.06	624,658.00
128KB	936415.62	754639.38	343847.41	355756.38	517994.34	581,730.63
256KB	1119073.12	1221315.25	415321.53	413109.31	512502.53	736,264.35
512KB	1389969.00	1396308.25	1413299.12	1349227.00	1438192.38	1,397,399.15
8MB	1295534.88	1131795.62	848793.00	560243.00	1463191.62	1,059,911.62
16MB	1812185.38	1871859.00	640119.19	639600.19	1770059.38	1,346,764.63

- deadline (Fread) Output is in kBytes/sec

record size	1	2	3	4	5	평균
8KB	1279730.00	1280910.00	1201751.00	1231781.25	1458937.38	1,290,621.93
16KB	1383522.62	1444917.00	1161217.62	862139.25	1403331.12	1,251,025.52
32KB	1322620.62	1408109.88	607550.00	1265619.50	805897.56	1,081,959.51
64KB	1220394.25	1324205.50	1267006.88	1289650.38	1392640.62	1,298,779.53
128KB	1434934.88	1322495.38	1046475.69	1087693.88	1355794.88	1,249,478.94
256KB	1452842.25	1575223.75	1258441.50	1354674.00	792284.06	1,286,693.11
512KB	1617606.38	1765202.12	1632157.25	1688917.25	640044.12	1,468,785.42
8MB	1751160.62	2030069.88	1047626.62	1675429.25	553804.31	1,411,618.14
16MB	1895483.62	1928370.00	821403.06	1824759.00	2038857.00	1,701,774.54

▶ CFQ

- CFQ (write) Output is in kBytes/sec

record size	1	2	3	4	5	평균
8KB	114984.41	81341.38	78462.84	44552.43	70142.52	77,896.72
16KB	196136.00	200217.38	235327.22	132598.80	158474.86	184,550.85
32KB	355312.06	400066.22	354735.44	376038.12	434738.78	384,178.12
64KB	669432.75	660103.31	574084.69	589882.31	699665.88	638,633.79
128KB	818389.94	776081.44	752457.62	729890.50	826266.50	780,617.20
256KB	1145605.38	1006077.25	1104255.88	1073999.88	1127241.50	1,091,435.98
512KB	1244763.12	1415789.50	1294306.50	1316286.25	1489225.75	1,352,074.22
8MB	1738047.25	1672349.00	1620278.00	1790041.88	1676007.62	1,699,344.75
16MB	1690586.38	1583186.75	1842506.75	1605443.62	1530454.88	1,650,435.68

- CFQ (Rewrite) Output is in kBytes/sec

record size	1	2	3	4	5	평균
8KB	93140.87	80550.37	37402.24	40072.57	74922.44	65,217.70
16KB	226796.72	206317.28	98522.29	82758.60	206874.81	164,253.94
32KB	389865.22	366883.06	416102.38	399591.62	184246.67	351,337.79
64KB	729527.88	708430.75	578924.25	315489.50	220522.06	510,578.89
128KB	990632.88	752192.06	343221.97	240169.41	375083.78	540,260.02
256KB	1014443.94	977965.88	1067942.25	1183506.62	1274137.75	1,103,599.29
512KB	1412406.38	1498068.25	1487772.38	1444377.38	464801.75	1,261,485.23
8MB	1962240.88	1801131.88	1820163.75	1551407.75	593583.19	1,545,705.49
16MB	1556332.75	1638971.38	1572282.50	1679678.00	721389.62	1,433,730.85

- CFQ (Read) Output is in kBytes/sec

record size	1	2	3	4	5	평균
8KB	59665.52	49168.29	40329.74	48875.15	54559.36	50,519.61
16KB	96849.61	97887.36	47834.49	46211.57	246288.22	107,014.25
32KB	372942.09	516777.41	459719.53	418948.12	197897.81	393,256.99
64KB	701468.88	907144.00	279462.34	256890.11	351665.31	499,326.13
128KB	840050.81	991761.88	345890.03	700162.56	774914.19	730,555.89
256KB	1116100.25	1225331.12	1379859.50	722890.62	1321532.12	1,153,142.72
512KB	1886297.88	1830141.88	440025.31	1459526.12	793978.81	1,281,994.00
8MB	2070921.50	1972387.38	1948562.12	420453.28	446083.69	1,371,681.59
16MB	1708065.00	1729923.50	1772251.88	1836298.50	1959944.62	1,801,296.70

- CFQ (Re-read) Output is in kBytes/sec

record size	1	2	3	4	5	평균
8KB	50791.51	58426.14	49105.90	50442.71	108617.70	63,476.79
16KB	98301.72	99529.20	74366.36	116722.01	256754.98	129,134.85
32KB	440766.28	418995.50	322567.31	417602.62	278713.81	375,729.10
64KB	759799.69	713853.62	305922.84	332763.78	345097.62	491,487.51
128KB	1015389.88	971442.50	506458.59	818135.75	918969.12	846,079.17
256KB	1191382.12	1275003.88	1238855.75	722890.62	775259.69	1,040,678.41
512KB	1886297.88	1727350.50	770752.12	880403.06	1559429.62	1,364,846.64
8MB	2205559.75	1843238.75	694365.50	580630.31	2140174.00	1,492,793.66
16MB	1699243.62	1853888.00	1790178.88	451906.19	658677.25	1,290,778.79

- CFQ (Random read) Output is in kBytes/sec

record size	1	2	3	4	5	평균
8KB	49311.13	45382.39	48160.49	45949.15	112007.56	60,162.14
16KB	89162.73	92464.85	91366.48	100644.49	160816.27	106,890.96
32KB	407811.22	359055.12	470581.47	441126.75	203615.66	376,438.04
64KB	758052.94	739777.81	218970.11	317715.06	527294.06	512,362.00
128KB	994065.38	936431.81	489722.62	512490.69	365886.44	659,719.39
256KB	1428289.75	1335759.12	529405.44	711013.12	396085.31	880,110.55
512KB	1959966.38	1690106.88	618374.25	615971.38	839359.00	1,144,755.58
8MB	1833966.12	1772243.38	2049081.00	580630.31	1867152.12	1,620,614.59
16MB	1939766.88	1853888.00	1930286.12	428068.47	1106579.75	1,451,717.84

- CFQ (Random write) Output is in kBytes/sec

record size	1	2	3	4	5	평균
8KB	94098.80	98627.13	99645.93	94523.46	115655.89	100,510.24
16KB	205017.67	210973.19	239472.31	201476.03	94044.83	190,196.81
32KB	388269.88	379400.03	364956.56	370512.00	229440.88	346,515.87
64KB	654033.94	662072.50	185391.05	226065.05	693953.75	484,303.26
128KB	918105.88	821550.94	304562.62	298474.19	437707.84	556,080.29
256KB	1126179.88	1078966.25	703993.69	423388.03	406889.88	747,883.55
512KB	1572669.88	1512358.88	709888.81	708945.69	402474.59	981,267.57
8MB	1673202.12	1510840.25	938500.19	1647303.00	1842872.62	1,522,543.64
16MB	1847275.50	1617983.12	585164.69	1819452.88	783179.25	1,330,611.09

- CFQ (Fread) Output is in kBytes/sec

record size	1	2	3	4	5	평균
8KB	606876.62	716770.19	592922.69	559920.19	1348765.62	765,051.06
16KB	669360.88	703198.12	697471.12	720465.31	1344890.25	827,077.14
32KB	1418441.62	1391250.75	1411221.62	1445609.88	531552.38	1,239,615.25
64KB	1331714.38	1353090.25	440700.91	749473.19	541635.50	883,322.85
128KB	1256602.75	1494426.12	1253067.62	1169464.75	1228407.62	1,280,393.77
256KB	1319066.38	1493347.38	733742.50	1421351.12	401058.69	1,073,713.21
512KB	1754795.25	1689627.12	1569409.88	1565356.88	1506176.38	1,617,073.10
8MB	1892787.50	1939350.25	506564.78	718283.44	2001225.38	1,411,642.27
16MB	1562417.88	1938304.12	758784.69	1960464.38	556877.31	1,355,369.68

■ 입력한 옵션에 대한 평균을 3개의 스케줄링 정책을 통해 성능 비교

▶ 성능 비교 방법

디스크 서비스 시간(sec)은 탐색시간(seek time) + 회전시간(rotation delay) + 전송시간(transfer time)으로 구하는데 다음과 같은 수행 결과를 해석하기 위해선 Output은 임의의 kBytes / 디스크 서비스 시간(sec)으로 결국 Output이 큰값이 나오는 것이 성능이 좋다. 또한 다음과 같은 스케줄링의 방식을 이해할 필요가 있다.

Deadline I/O scheduler

(1) 4개의 큐 = 정렬된 읽기 큐, 정렬된 쓰기 큐, 입력 FIFO 큐, 출력 FIFO 큐를 사용한다.
(2) Deadline이 지난 요청이 없을 경우에 정렬된 큐에서 요청을 꺼내서 처리
(3) 모든 요청은 마감시간을 가짐
(4) 읽기 요청(500ms), 쓰기 요청(5s)
(5) 기아 현상(starvation) 방지
(6) 읽기 우선 정책

CFQ(Complete Fair Queueing) I/O scheduler

(1) 입출력을 요청하는 모든 프로세스들에 대해 디스크 I/O 대역폭을 공평하게 할당하는 것을 보장하는 기법
(2) I/O를 요청한 프로세스 별로 큐를 할당
(3) 각 프로세스에 관한 큐는 섹터 순으로 정렬
(4) 각 큐는 라운드 로빈으로 처리
(5) 4개씩 요청을 처리(설정 가능한 값)

NOOP(NO OPreation) I/O scheduler

(1) 인접한 요청 병합만 수행하고 그 외에 아무 작업을 하지 않음
(2) Requeset FIFO 큐만 유지
(3) Random access 하는 device를 위한 스케줄러
(4) 탐색에 대한 부담이 없음 → 정렬이 필요 없음
(5) 플래시 메모리에서 주로 사용

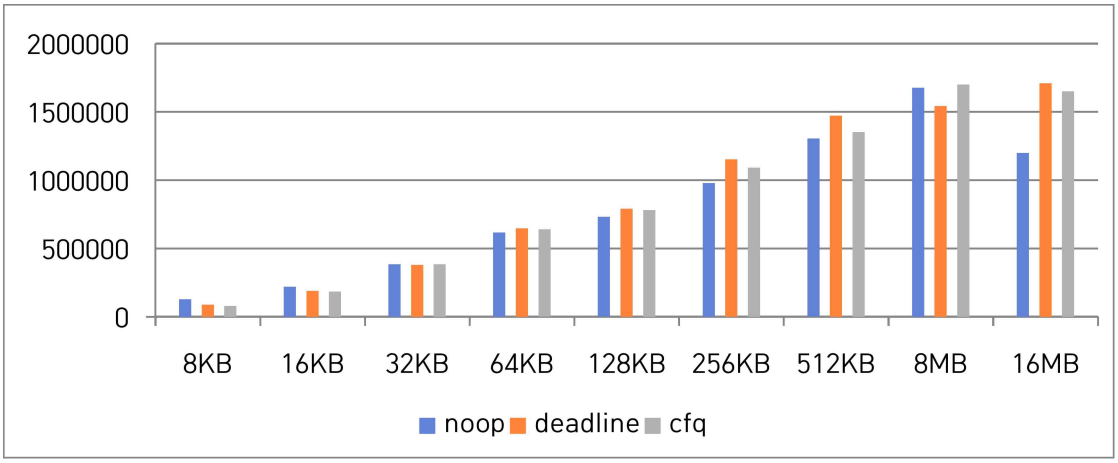
이를 이해하고 다음과 같은 결과를 비교한다.

▶ 성능 비교

(표와 그래프로 표현)

- Write Output is in kBytes/sec for three case

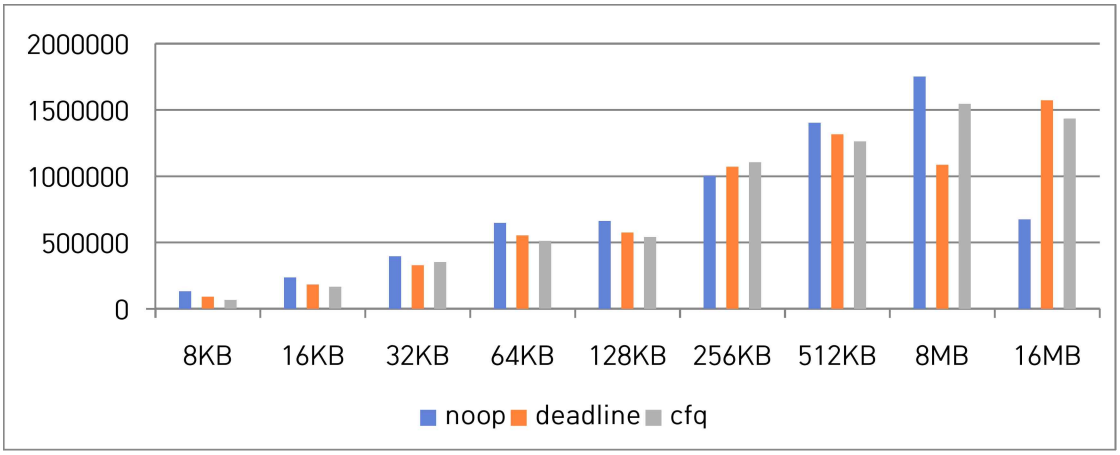
record size	noop	deadline	cfq
8KB	127,522.22	87,324.59	77,896.72
16KB	219,211.55	189,467.34	184,550.85
32KB	383,095.29	378,427.18	384,178.12
64KB	615,765.93	647,010.97	638,633.79
128KB	730,168.77	789,379.46	780,617.20
256KB	977,846.11	1,151,380.30	1,091,435.98
512KB	1,303,446.98	1,470,505.05	1,352,074.22
8MB	1,675,532.63	1,542,631.90	1,699,344.75
16MB	1,198,599.18	1,709,578.45	1,650,435.68



record size가 아주 작은 record size에서는 noop방식이 우수하다 커지면서 deadline과 cfq가 앞서나가다 deadline이 가장 좋아지는 양상을 보인다 cfq 또한 record size가 커짐에 따라 좋은 성능을 보인다. 하지만 오차를 제외하고 3개를 비교하여 보면 근소한 차이 정도만 있는 양상을 보인다. 이는 실험 결과에 따라 순위가 뒤집힐 수 있으며 어떠한 것이 좋은지 판단하기 어려워 보인다.

- Rewrite Output is in kBytes/sec for three case

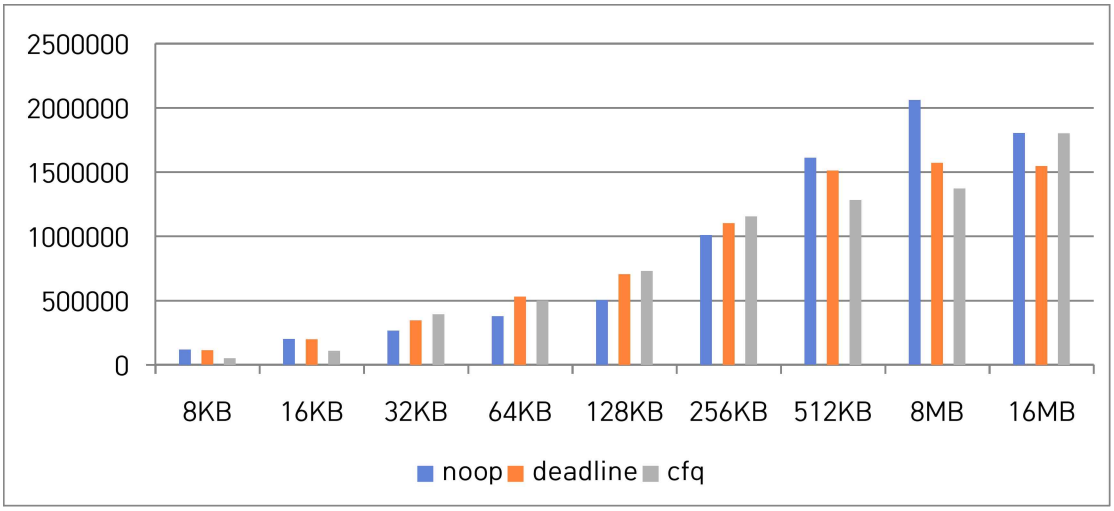
record size	noop	deadline	cfq
8KB	131,634.21	89,409.08	65,217.70
16KB	236,071.63	182,705.85	164,253.94
32KB	395,480.90	327,910.18	351,337.79
64KB	648,007.76	553,596.47	510,578.89
128KB	662,662.57	575,445.67	540,260.02
256KB	1,001,501.97	1,070,162.09	1,103,599.29
512KB	1,402,657.03	1,315,855.62	1,261,485.23
8MB	1,749,756.05	1,085,774.55	1,545,705.49
16MB	673,163.65	1,571,006.79	1,433,730.85



record size가 아주 작은 record size에서는 noop방식이 우수하다 커지면서 deadline과 cfq가 앞서나가다 deadline이 가장 좋아지는 양상을 보인다 cfq 또한 record size가 커짐에 따라 좋은 성능을 보인다. 하지만 오차를 제외하고 3개를 비교하여 보면 근소한 차이 정도만 있는 양상을 보인다. 이는 실험 결과에 따라 순위가 뒤집힐 수 있으며 어떠한 것이 좋은지 판단하기 어려워 보인다. 하지만 Write와 같은 양상으로 그래프가 그려지는 것을 보았을 때 8MB에서 noop가 가장좋으며 이후 record size가 커짐에 따라 deadline에게 1등을 뺏기는 양상은 일관성있는 결과로 생각된다.

- Read Output is in kBytes/sec for three case

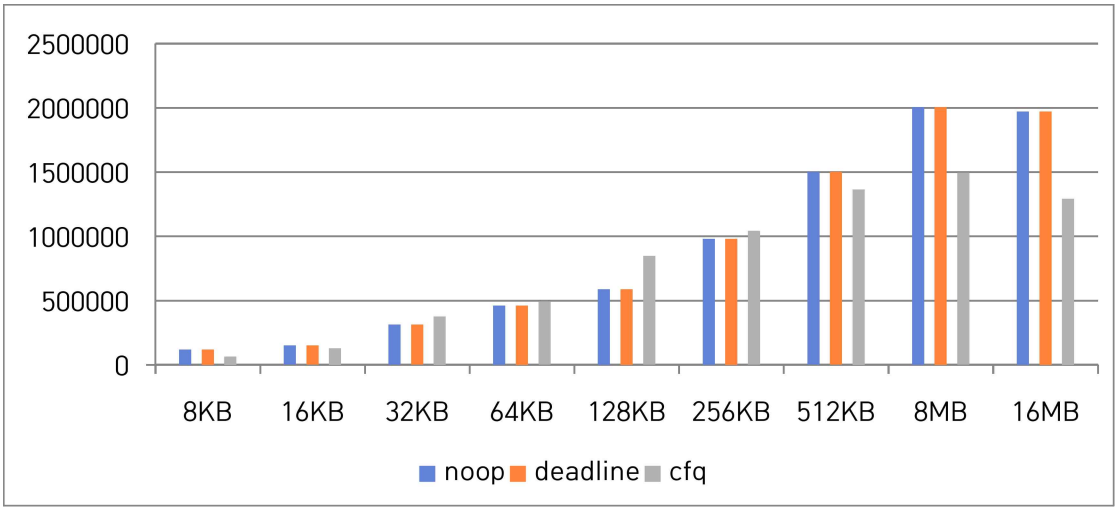
record size	noop	deadline	cfq
8KB	118,527.88	112,168.38	50,519.61
16KB	201,669.84	197,463.42	107,014.25
32KB	266,667.61	346,328.47	393,256.99
64KB	376,856.63	531,264.12	499,326.13
128KB	504,577.00	705,157.57	730,555.89
256KB	1,008,790.66	1,101,060.02	1,153,142.72
512KB	1,610,026.02	1,511,664.19	1,281,994.00
8MB	2,059,288.05	1,571,454.36	1,371,681.59
16MB	1,802,834.05	1,547,256.98	1,801,296.70



작은 레코드 사이즈에서 커질수록 다들 성능이 증가하는 양상을 보인다. 가장 좋은 성능은 noop가 8MB일때이고 이후 더 커진 레코드 사이즈에서는 noop는 감소하고 cfq는 성능이 증가하는 양상을 보인다.

- Re-read Output is in kBytes/sec for three case

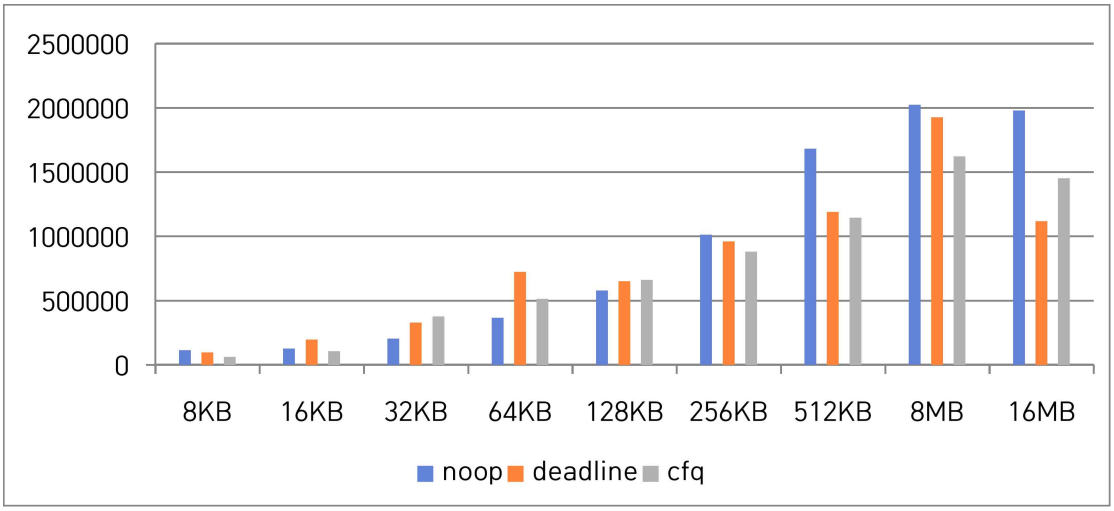
record size	noop	deadline	cfq
8KB	119,128.84	119,128.84	63,476.79
16KB	149,726.46	149,726.46	129,134.85
32KB	311,865.23	311,865.23	375,729.10
64KB	461,091.01	461,091.01	491,487.51
128KB	587,853.06	587,853.06	846,079.17
256KB	980,081.49	980,081.49	1,040,678.41
512KB	1,500,101.65	1,500,101.65	1,364,846.64
8MB	2,005,421.22	2,005,421.22	1,492,793.66
16MB	1,970,308.52	1,970,308.52	1,290,778.79



작은 레코드 사이즈에서 커질수록 다들 성능이 증가하는 양상을 보인다. 가장 좋은 성능은 noop와 deadline가 8MB일때이고 이후 더 커진 레코드 사이즈에서는 3개의 정책이 조금의 감소가 있음을 알 수 있다. 그 전 과정에 있어서 deadline이 특히 성능이 우수한 모습이 있는데 이는 읽기 우선 정책이 잘 반영됨을 알 수 있다.

- Random read Output is in kBytes/sec for three case

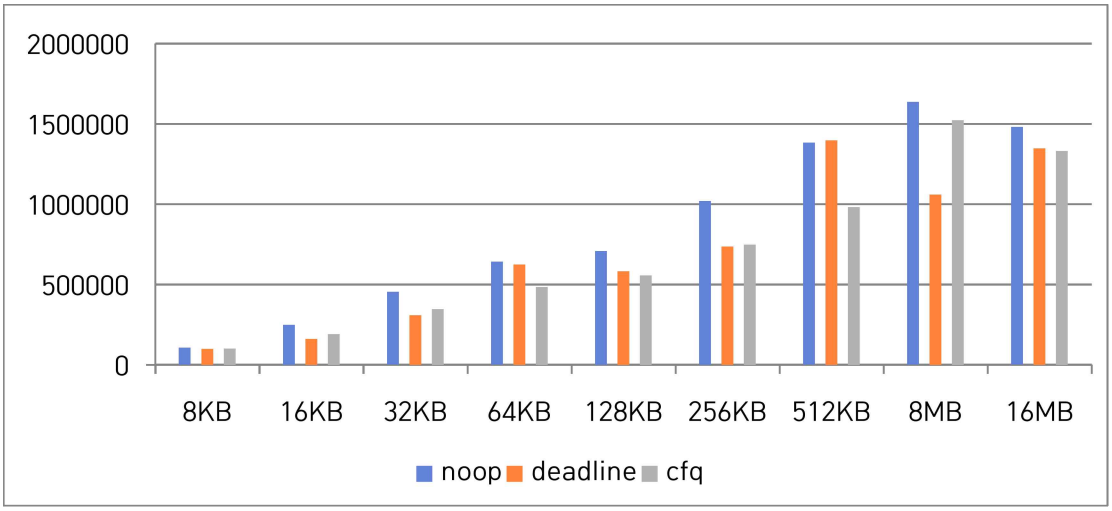
record size	noop	deadline	cfq
8KB	113,347.77	96,022.04	60,162.14
16KB	124,432.28	195,593.37	106,890.96
32KB	202,756.63	327,189.65	376,438.04
64KB	365,974.03	721,873.01	512,362.00
128KB	577,289.22	650,149.21	659,719.39
256KB	1,011,972.65	959,190.39	880,110.55
512KB	1,682,126.00	1,189,765.88	1,144,755.58
8MB	2,021,703.62	1,925,819.13	1,620,614.59
16MB	1,978,751.90	1,117,699.58	1,451,717.84



작은 레코드 사이즈에서 커질수록 다들 성능이 증가하는 양상을 보인다. 가장 좋은 성능은 noop가 8MB일때이고 이는 Random access 하는 device를 위한 스케줄러의 특성을 잘 반영하였다. 이후 더 커진 레코드 사이즈에서는 3개의 정책이 감소가 있음을 알 수 있다. 특히 deadline의 경우 감소 폭이 나머지 둘 보다 크다.

- Random write Output is in kBytes/sec for three case

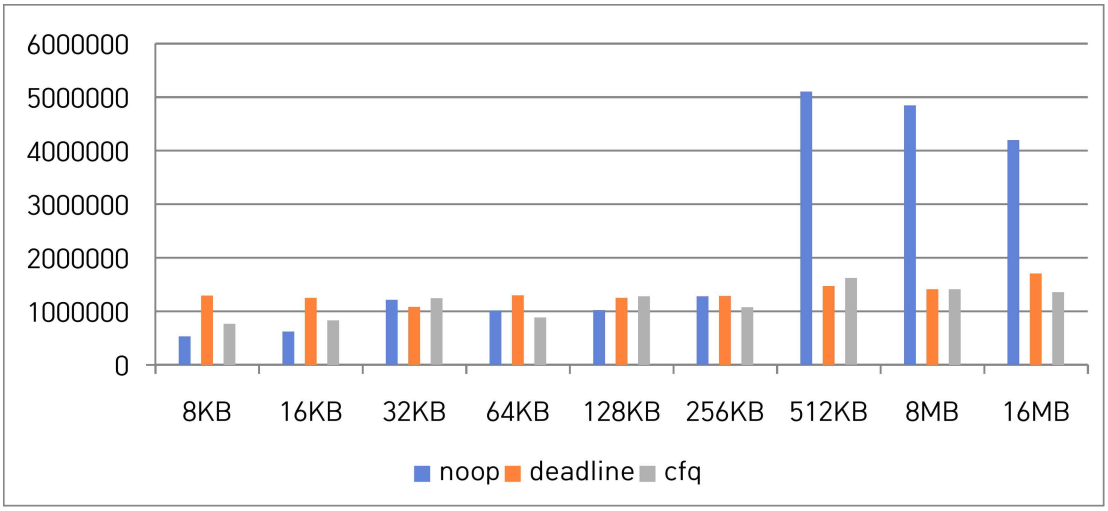
record size	noop	deadline	cfq
8KB	105,794.07	99,123.82	100,510.24
16KB	248,288.79	161,118.59	190,196.81
32KB	453,229.53	308,721.80	346,515.87
64KB	640,817.50	624,658.00	484,303.26
128KB	707,476.20	581,730.63	556,080.29
256KB	1,019,260.11	736,264.35	747,883.55
512KB	1,381,992.35	1,397,399.15	981,267.57
8MB	1,635,432.63	1,059,911.62	1,522,543.64
16MB	1,479,532.23	1,346,764.63	1,330,611.09



작은 레코드 사이즈에서 커질수록 다들 성능이 증가하는 양상을 보인다. 가장 좋은 성능은 noop가 8MB일때이고 이는 Random access 하는 device를 위한 스케줄러의 특성을 잘 반영하였다. 이후 더 커진 레코드 사이즈에서는 noop와 cfq가 감소가 있음을 알 수 있다. 하지만 deadline의 경우 대폭 증가의 모습을 볼 수 있다. cfq같은 경우 512KB보다 8MB에서 대폭 증가의 모습을 보인다. noop의 16MB 전까지 꾸준한 증가를 볼 수 있다.

- Fread Output is in kBytes/sec for three case

record size	noop	deadline	cfq
8KB	530,961.69	1,290,621.93	765,051.06
16KB	618,504.91	1,251,025.52	827,077.14
32KB	1,213,780.48	1,081,959.51	1,239,615.25
64KB	1,017,184.55	1,298,779.53	883,322.85
128KB	1,022,274.86	1,249,478.94	1,280,393.77
256KB	1,280,705.11	1,286,693.11	1,073,713.21
512KB	5,100,213.55	1,468,785.42	1,617,073.10
8MB	4,843,774.63	1,411,618.14	1,411,642.27
16MB	4,195,145.05	1,701,774.54	1,355,369.68



먼저 일반 read와 비교하였을 시 월등히 빠른 성능을 확인할 수 있다. 이는 위에서 설명했다시피 intermediate buffer를 가지고 입출력하기 때문이다. 또한, 512KB부터는 noop의 월등한 성능을 볼 수 있는데 플래시 메모리에서 주로 사용하는 특성이 반영된 것으로 생각한다. 나머지 두 개의 정책은 크기가 커짐에 따라 증가하는 폭도 매우 적을뿐더러 오히려 감소하는 때도 다수 존재한다.

● 고찰

실험 결과가 예상한 것보다 비교 분석이 복잡하게 나왔는데 이를 해결하기 위해 평균을 구하기 위한 과정을 5회가 아닌 그 이상으로 해야 한다고 생각한다. 또한, 실행횟수를 보게 되면 엄청나게 많은 실행을 한다는 것을 알 수 있다. 이를 조금이나마 시간을 단축하기 위해 make 파일의 조건부 컴파일을 활용하여 구현하게 되었다.

또한, 해당 과제에서 필요한 스케줄링 정책은 아니지만 새롭게 알게된 스케줄링 정책은 아래와 같다.

Anticipatory I/O scheduler

1. 데드라인 I/O 스케줄러에 기반한다.
2. 인접한 위치에서 짧은 시간 간격으로 읽기 요청하는 경우가 빈번하다
3. 읽기 요청을 처리한 후, 헤드를 이동하지 않고 잠시동안 새로운 요청 기다린다.
   이때 시간은 6ms(설정 가능한 값) 으로 설정하고 예측 실패 시 오버헤드(Overhead)가 발생한다.

Elevator scheduler

1. Linux Kernel 2.4에서 사용한다
2. 디스크 헤드의 이동 방향을 고려한다.
3. Mmerge - Front, Back
4. Insertion point
   - 기존 요청과 인접한 요청이 들어오면 병합한다
   - 물리적으로 적당한 위치에 삽입한다.
   - 큐에 오랫동안 기다린 요청이 있을 경우, 큐의 맨 끝에 삽입한다.

또한, 4번째 연산은 fread를 선택하였고 이러한 이유는 fread는 intermediate buffer를 가지고 입출력을 하므로 일정한 크기의 데이터가 입력되거나 출력될 때까지, 물리적으로 입출력을 하지 않기 때문이다. 그래서, 데이터를 빈번하게 입출력하면 fread가 read보다 빠르므로 이러한 원리를 직접 테스트하기 위해 사용하였다.