

졸업자격시험보고서

## Cgroups 성능 분석 및 평가

The evaluation for Cgroups

지도교수 강 동 현

동국대학교 과학기술대학 컴퓨터공학과

이 은 진

2 0 2 0

졸업자격실험보고서

Cgroups 성능 분석 및 평가

The evaluation for Cgroups

이 은 진

지도교수 강 동 현

본 보고서를 졸업자격 실험보고서로 제출함.  
2020년 06월 15일

이 은 진의 졸업자격 실험보고 통과를 인준함.  
2020년 06월 15일

주 심 도 제 수

부 심 박 기 석

부 심 강 동 현 (인)

동국대학교 과학기술대학 컴퓨터공학과

# 목 차

1. 서론 .....	1
2. 이론적 배경 .....	2
2.1. Cgroups .....	2
2.2. Rocks DB .....	3
2.3. Benchmark tool(FIO, YCSB) .....	4
3. 실험 환경 .....	5
3.1. 공통 실험 환경 .....	5
3.2. FIO 설치 .....	6
3.3. YCSB - Rocks DB 설치 .....	7
3.3.1 JDK 설치 .....	7
3.3.2 g++ 설치 .....	8
3.3.3 git 설치 .....	9
3.3.4 YCSB - Rocks DB 설치 .....	10
4. 실험 .....	11
4.1. Cgroups blkio weight 100, 200, 400, 800 성능 측정 .....	11
4.1.1. 디스크 추가 .....	11
4.1.1.1. 디스크 확인 .....	11
4.1.1.2. 파티션 생성 .....	12
4.1.1.3. 파티션 포맷 .....	13

4.1.1.4. 디스크 자동 마운트 .....	14
4.1.2. Cgroups Block I/O 컨트롤러 사용 .....	15
4.1.2.1. Blkio I/O 컨트롤러 실행 조건 .....	15
4.1.2.2. Cgroups 마운트 및 Blkio I/O 컨트롤러 사용 .....	16
4.1.2.3. Cgroups에 프로세스 할당 .....	17
4.1.2.4. weight 비율 할당 .....	18
4.1.3. FIO 툴을 사용해 IOPS 및 BW 측정 .....	19
4.1.3.1. FIO Job File 생성 .....	19
4.1.3.2. FIO Job File 명령어 .....	20
4.1.3.3. FIO 실행 .....	22
4.1.4. 실험 결과 .....	24
4.2. YCSB-Rocks DB를 이용한 FIO-Cgroups 성능 측정 .....	25
4.2.1. 디스크 추가 .....	25
4.2.1.1. 디스크 확인 .....	25
4.2.1.2. 파티션 설정 .....	26
4.2.1.3. 파티션 포맷 .....	27
4.2.1.4. 디스크 수동 마운트 .....	28
4.2.2. Cgroups Block I/O 컨트롤러 사용 .....	29
4.2.3. YCSB-Rocks DB를 이용해 FIO-Cgroups 성능 측정 .....	29
4.2.3.1. Read .....	29
4.2.3.2. Write .....	36
4.2.4. 실험 결과 .....	37
4.3. Cgroups를 활용한 I/O weight 개수에 따른 성능 평가 .....	38
4.3.1. 디스크 추가 .....	38
4.3.2. FIO 벤치마크 툴을 이용해 Cgroups 실행 .....	38
4.3.2.1. Cgroups Blkio I/O 컨트롤러 사용 .....	38
4.3.2.2. Cgroups 마운트 및 Blkio I/O 컨트롤러 사용 .....	39
4.3.2.3. Cgroups에 프로세스 할당 .....	40

4.3.2.4. weight 비율 할당 .....	41
4.3.3. FIO-Cgroups 성능 측정 .....	42
4.3.3.1. FIO Job File 생성 .....	42
4.3.3.2. FIO Job File 옵션 .....	42
4.3.3.3. FIO 실행 .....	44
4.3.4. 실험 결과 .....	46
 5. 결론 및 향후 실험 .....	 47
 참고문헌 .....	 48
 요약서 .....	 00

## 그 립 목 차

[그림 3-1] FIO 설치 .....	6
[그림 3-2] FIO 설치 완료 .....	6
[그림 3-3] JDK9 설치 .....	7
[그림 3-4] 환경 변수 설정 .....	7
[그림 3-5] Java 설치 완료 .....	7
[그림 3-6] g++ 설치여부 확인 .....	8
[그림 3-7] g++ 설치 .....	8
[그림 3-8] g++ 설치 완료 .....	8
[그림 3-9] git 설치여부 확인 .....	9
[그림 3-10] git 설치 .....	9
[그림 3-11] git 설치 완료 .....	9
[그림 3-12] git에서 YCSB 복제 .....	10
[그림 3-13] YCSB 설치 확인 .....	10
[그림 3-14] Maven 설치 .....	10
[그림 3-15] Rocks DB 설치 .....	10
[그림 4-1] 디스크 장착 확인 .....	11
[그림 4-2] 파티션 생성 과정 .....	12
[그림 4-3] 4개의 파티션 생성 .....	12
[그림 4-4] 각각의 파티션에 ext4 파일 시스템 포맷 .....	13
[그림 4-5] 디스크 자동 마운트 .....	14
[그림 4-6] Cgroups blkio 서브시스템 사용 준비 .....	15
[그림 4-7] Cgroups 마운트 및 task1-4 그룹 생성 .....	16
[그림 4-8] 프로세스 목록 확인 .....	17
[그림 4-9] 4개의 task 그룹에 임의의 TID 할당 .....	17
[그림 4-10] 임의로 지정한 TID가 정상적으로 할당 되었는지 확인 .....	17

[그림 4-11] wieght 설정 및 확인 .....	18
[그림 4-12] blkio 서브시스템 마운트 확인 .....	18
[그림 4-13] Job File 저장 .....	19
[그림 4-14] Job File 생성 .....	19
[그림 4-15] FIO 벤치마크 실행 .....	22
[그림 4-16] SSD를 마운트 한 디렉토리에 생성되어있는 파일 전체 삭제 ...	23
[그림 4-17] FIO 결과 그래프 (BW) .....	24
[그림 4-18] FIO 결과 그래프 (IOPS) .....	24
[그림 4-19] 디스크 장착 확인 .....	25
[그림 4-20] 파티션 설정 .....	26
[그림 4-21] 하나의 파티션 설정 과정 .....	26
[그림 4-22] 하나의 파티션에 ext4 파일 시스템 포맷 .....	27
[그림 4-23] 디스크를 마운트 할 디렉토리 생성 .....	28
[그림 4-24] 디스크 수동 마운트 .....	28
[그림 4-25] 디스크 마운트 확인 .....	28
[그림 4-26] YCSB 디렉토리로 이동 후 workloada를 여는 과정 .....	29
[그림 4-27] Read를 하기 위한 workloada 옵션 변경 .....	29
[그림 4-28] workloada를 load하는 옵션 및 과정 .....	30
[그림 4-29] FIO와 YCSB 동시 실행 .....	31
[그림 4-30] workloada를 run하는 옵션 및 과정 .....	32
[그림 4-31] loadtest.txt 파일과 runtest.txt 파일 생성 .....	33
[그림 4-32] runtest.txt 결과 파일 .....	33
[그림 4-33] FIO-Cgroupos test1 결과 .....	34
[그림 4-34] FIO-Cgroupos test2 결과 .....	34
[그림 4-35] FIO-Cgroupos test3 결과 .....	34
[그림 4-36] FIO-Cgroupos test4 결과 .....	34
[그림 4-37] load, run 파일 삭제 .....	35
[그림 4-38] SSD 마운트 디렉토리 안에 있는 파일 삭제 .....	35

[그림 4-39] Write를 하기 위한 workloada 옵션 변경 .....	36
[그림 4-40] FIO-Cgroup, YCSB-RocksDB & FIO-Cgroup (BW) .....	37
[그림 4-41] FIO-Cgroup, YCSB-RocksDB & FIO-Cgroup (IOPS) .....	37
[그림 4-42] blkio 서브시스템을 사용하기 위한 Cgroups 활성화 .....	38
[그림 4-43] Cgroups 지원 확인 .....	38
[그림 4-44] 스케줄러 확인 .....	38
[그림 4-45] Cgroups 루트 디렉토리와 blkio 하위 시스템 마운트 .....	39
[그림 4-46] task1, 2, 3, 4 ... 128 이름을 가진 그룹 생성 .....	39
[그림 4-47] task1, 2, 3, 4 ... 128 그룹에 임의의 TID 할당 .....	40
[그림 4-48] 임의로 지정한 TID 할당 확인 .....	40
[그림 4-49] task1, 2, 3, 4 ... 128 그룹에 weight 설정 .....	41
[그림 4-50] task1, 2, 3, 4 ... 128 그룹에 wieght 할당 확인 .....	41
[그림 4-51] task1, 2, 3, 4 ... 128 그룹 마운트 확인 .....	41
[그림 4-52] 4개의 task 그룹을 묶어 만든 Job File 실행 .....	44
[그림 4-53] 8개의 task 그룹을 묶어 만든 Job File 실행 .....	44
[그림 4-54] 16개의 task 그룹을 묶어 만든 Job File 실행 .....	45
[그림 4-55] FIO-Cgroups I/O weight 개수에 따른 결과 그래프 (IOPS) .....	46



## 표 목 차

[표 2-1] Cgroups 서브시스템 .....	2
[표 2-2] FIO 명령어 .....	4
[표 2-3] YCSB workload 종류 .....	4
[표 3-1] 공통 실험 환경 .....	5
[표 4-1] FIO Job File 옵션 .....	20
[표 4-2] 4개의 task 그룹 FIO Job File 옵션 .....	43
[표 4-3] 8개의 task 그룹 FIO Job File 옵션 .....	43

## 1. 서론

클라우드 플랫폼 기술 (예: Microsoft Azure, Amazon AWS, 등.)의 발전과 함께 최근 다양한 분야에서 클라우드 플랫폼을 활용하고 있다. 클라우드 플랫폼은 동일한 하드웨어 자원을 다수의 사용자 또는 어플리케이션이 실시간으로 공유하기 때문에 제한된 시스템의 자원을 효율적으로 분배하는 일은 중요한 이슈 중 하나이다. 이에, 산업계와 학계에서는 시스템의 자원을 적절하게 분배하기 위한 지원 및 연구가 이루어지고 있다[1]-[5].

리눅스 시스템의 경우, 다양한 시스템의 자원을 선택적으로 분배하기 위해 Cgroups를 지원하고 있으며, Cgroups는 호스트 시스템의 CPU, Memory, I/O 등의 자원에 대한 사용을 가중치(Weight) 기반으로 제한할 수 있다[1]. 또한, Cgroups는 다수의 프로세스를 하나의 태스크(Task) 그룹으로 지정하고 지정된 그룹을 대상으로 시스템의 자원을 제한할 수 있다.

이에, 본 논문에서는 Cgroups가 시스템의 자원을 가중치 기반으로 적절하게 분배하는지 확인한다. 특히 Cgroups로 지정하는 태스크 그룹의 개수가 증가하는 경우 자원이 제대로 분배되고 있는지에 대한 여부를 확인한다.

따라서 본 논문의 2장에서는 먼저 이론적 배경에 대해 설명하고 실험에 사용한 데이터베이스 및 벤치마크 툴에 대해 소개한다.

3장에서는 Cgroups의 성능을 분석하고 평가하기 위한 실험 환경을 소개하고, 실험에 필요한 벤치마크 툴과 데이터베이스 설치하는 방법을 설명한다.

4장에서는 본격적으로 본 논문에서 제시한 실험을 진행한다. 첫 번째 실험은 1개의 프로세스만을 가지는 4개의 태스크 그룹을 생성하고 100, 200, 400, 800의 가중치를 부여해 실험을 진행하였다. 두 번째 실험은 첫 번째 실험과 동일한 태스크 그룹 및 가중치를 부여하고 데이터베이스와 Cgroups를 동시에 측정하여 실험하였다. 세 번째 실험은 첫 번째 실험과 동일한 가중치를 할당하고, 태스크 그룹을 4에서 128까지 증가하며 실험을 진행하였다.

5장에서는 각 실험에 대한 결론을 설명하고 향후 실험에 대한 계획을 제시한다.

## 2. 이론적 배경

### 2.1. Cgroups

Cgroups란 구글이 개발한 시스템 소프트웨어로 현재 리눅스에서 시스템 리소스를 분배 및 제한하기 위해 사용되는 리눅스 커널 모듈이다. Cgroups는 호스트 시스템의 CPU, Memory, I/O 등의 하드웨어 자원 할당을 가중치(wieght) 기반으로 제한할 수 있으며, 다양한 서브시스템을 포함하고 있다[1]. 리눅스에서 Cgroups는 호스트 시스템의 특정 하드웨어 자원을 할당하고 이를 프로세스 그룹 단위로 제어 및 관리하는 기능만 수행하며, 특정 하드웨어에 필요한 자원을 분배하는 일은 서브시스템이 수행한다[1]-[3].

다양한 서브시스템 중 blkio 서브시스템은 I/O 호스트와 블록 디바이스 사이의 I/O에 대한 자원을 제어한다. 다양한 매개 변수를 통해 블록 디바이스의 I/O를 제어할 수 있으며, I/O 가중치에 따라 I/O 대역폭을 분배한다[6]. blkio.weight는 가장 대표적인 매개변수로 사용자는 blkio.weight 변수를 사용하여 Cgroups에서 지정한 그룹에 대해 상대적인 I/O 가중치를 부여할 수 있다. 설정 가능한 가중치는 100에서 1000까지이며, 가중치를 설정하지 않는 경우, 기본 가중치 값이 설정된다. 예를 들어 tak1 그룹에 기중치 100을 설정하고, task2 그룹에 가중치 200을 설정하면 task1 그룹과 task2 그룹은 서로 1:2 비율로 블록 디바이스의 I/O 액세스 자원을 분배하게 된다[7].

[표 2-1] Cgroups 서브시스템

<b>CPU</b>	CPU의 사용량을 제한하고, CPU에 Cgroup 작업 액세스를 제공하기 위한 스케줄러 제공
<b>CPUacct</b>	CPU의 자원 사용에 대한 분석 및 통계
<b>CPUset</b>	CPU 및 메모리 배치 제어
<b>Memory</b>	Cgroup 작업에 사용되는 메모리 사용량 제한
<b>Devices</b>	디바이스에 대한 액세스를 허가 또는 제한
<b>Freezer</b>	그룹에 속한 프로세서를 일시적으로 정지 또는 재개
<b>Net_cls</b>	특정 Cgroup 작업에서 발생하는 패킷을 식별하기 위한 태그 지정
<b>Blkio</b>	특정 블록 디바이스에 대한 접근을 제한 또는 제어

## 2.2. Rocks DB

Rocks DB는 Facebook에서 개발하여 오픈 소스로 공개한 NoSQL 데이터베이스로, 서버 워크로드와 같은 대용량 데이터 처리에 적합하고, 고속 쓰기 및 읽기에 최적화된 데이터베이스이다[8]. SSD 저장 장치 기반 서버 환경에서 높은 성능을 보장하며, 고성능 I/O, 고성능 데이터 압축률을 제공한다. Rocks DB는 C++ 또는 Java 라이브러리 형태로 제공되고, key-value 쌍으로 데이터를 저장하는 key-value 저장방식을 취하며, Rocks DB를 구성하는 여러 요소들은 사용자가 수정 및 구현할 수 있다[8]-[9]. 또한 Rocks DB는 네트워크 연결을 통한 원격 접속을 지원하지 않으며, 로컬 컴퓨터에서만 접근 가능하다. Rocks DB의 저장소는 비휘발성으로 데이터를 영구적으로 저장하는데 적합한 데이터베이스이다. 또한 Rocks DB는 다양한 환경(예 : 하드디스크, 플래시 메모리, 등.)에서 사용 가능하며 다양한 디버깅 도구 및 압축 알고리즘을 사용할 수 있다[10].

### 2.3. Benchmark tool(FIO, YSCB)

FIO(Flexible I/O Tester)는 I/O의 읽기 및 쓰기, IOPS, BW를 측정하는 벤치마크 툴로 리눅스 I/O의 하위 시스템 및 스케줄러를 유연하게 측정할 수 있다[11].

[표 2-2] FIO 명령어

<b>bs</b>	테스트할 블록의 크기
<b>ioengine</b>	프로세스가 I/O를 발생하는 방법
<b>iodepth</b>	AIO를 사용할 때 동시에 발생할 I/O 수의 상한선
<b>direct</b>	Direct I/O 사용
<b>rw</b>	테스트 종류 (read, write, randread, randwrite)
<b>time_based</b>	runtime 시간 만큼 I/O를 실행
<b>runtime</b>	테스트 진행 시간 (단위 : 초)
<b>size</b>	Job이 완료하기까지 I/O를 수행할 데이터의 양

NoSQL 데이터베이스 벤치마크 YCSB(Yahoo Cloud Serving Benchmark)는 클라우드 서비스 성능을 측정하기 위한 벤치마크 툴이다. Java를 기반으로 하지만 JDK9, JDK10은 지원하지 않으며, YCSB의 빌드는 Maven을 통해서 수행된다[12]. 다음 [표 2-3]은 YCSB가 제공하는 workload이다.

[표 2-3] YCSB workload 종류

<b>Workload A</b>	데이터의 50%를 데이터베이스에서 읽고 50%를 데이터베이스에 쓴다.
<b>Workload B</b>	데이터의 95%를 데이터베이스에서 읽고 5%를 데이터베이스에 쓴다.
<b>Workload C</b>	데이터의 100%를 데이터베이스에서 읽는다.
<b>Workload D</b>	새로운 데이터를 데이터베이스에 저장하고, 가장 최근에 저장된 데이터들을 읽는다.
<b>Workload E</b>	각각의 데이터를 쿼리하는 대신, 어떤 짧은 구간 내에 속하는 데이터들을 쿼리한다.
<b>Workload F</b>	데이터를 읽고 수정한 후, 그 수정된 데이터를 다시 데이터베이스에 쓴다.

### 3. 실험 환경

#### 3.1. 공통 실험 환경

본 논문에서는 Cgroups의 성능을 분석 및 평가하기 위해 Cgroups의 blkio 서브시스템을 활용하였으며, AMD RYZEN 7 3700K 3.60GHz, 8GB DRAM, Samsung SSD 860 PRO 256GB 환경에서 실험을 진행하였다. 또한 실험을 위해 CFQ(Completely Fair Queuing)[13] I/O 스케줄러를 기본 스케줄러로 사용하였다.

[표 3-1] 공통 실험 환경

CPU	AMD RYZEN 7 3700K 3.60GHz
Memory	8GB (DDR4 4GB * 2)
Storage	Samsung SSD 860 PRO 256GB
Operation System	Ubuntu 18.04 LTS
Kernel Version	Linux Kernel 4.15.0

### 3.2. FIO 설치

Cgroups 실험에서 I/O 읽기 및 쓰기, IOPS, BW를 측정하기 위해 FIO 벤치마크 툴을 설치한다. 먼저 `sudo apt-get install fio` 명령어를 통해 FIO를 설치하고, `fio --version` 명령어를 통해 FIO 설치 여부를 확인한다.

```
eunjin@eunjin-PC:~$ sudo apt-get install fio
패키지 목록을 읽는 중입니다... 완료
의존성 트리를 만드는 중입니다
상태 정보를 읽는 중입니다... 완료
```

[그림 3-1] FIO 설치

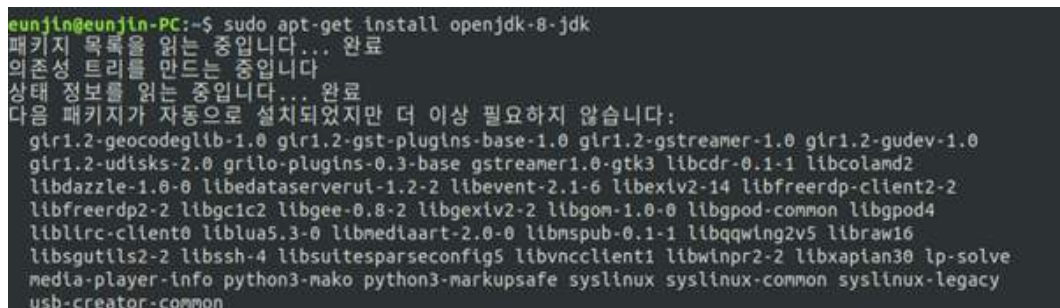
```
eunjin@eunjin-PC:~$ fio --version
fio-3.1
```

[그림 3-3] FIO 설치 완료

### 3.3. YCSB-Rocks DB 설치

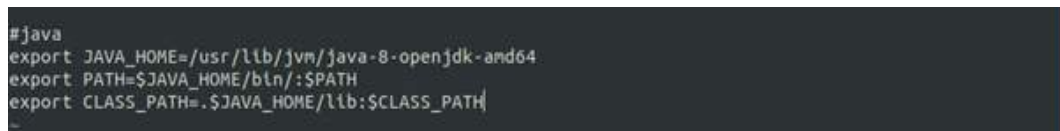
#### 3.3.1. JDK 설치

YCSB 벤치마크 툴을 통해 Rocks DB를 사용하기 위해 JDK를 설치한다. YCSB는 JDK9, JDK10 버전을 지원하지 않기 때문에 JDK8 버전을 설치한다. 먼저 `java -version` 명령어를 통해 Java 설치여부를 확인한 뒤 Java가 설치되어 있지 않으면 `sudo apt-get install openjdk-8-jdk` 명령어를 통해 Java를 설치한다. 그 다음 설치한 Java의 경로를 확인하고 `sudo vi /etc/profile` 명령어를 통해 환경 변수를 설정한다. 환경 변수를 설정할 때는 아래의 [그림3-4]와 같이 `export` 세줄을 입력하면 된다. 마지막으로 `source /etc/profile` 명령어를 통해 환경 변수를 적용하고 `echo $JAVA_HOME` 명령어를 통해 환경 변수 적용 테스트를 진행한다.



```
eunjin@eunjin-PC:~$ sudo apt-get install openjdk-8-jdk
패키지 목록을 읽는 중입니다... 완료
의존성 트리를 만드는 중입니다
상태 정보를 읽는 중입니다... 완료
다음 패키지가 자동으로 설치되었지만 더 이상 필요하지 않습니다:
gir1.2-geocodeglib-1.0 gir1.2-gst-plugins-base-1.0 gir1.2-gstreamer-1.0 gir1.2-gudev-1.0
gir1.2-udisks-2.0 grilo-plugins-0.3-base gstreamer1.0-gtk3 libcdr-0.1-1 libcolamd2
libdazzle-1.0-0 libdataserverui-1.2-2 libevent-2.1-6 libexiv2-14 libfreerdp-client2-2
libfreerdp2-2 libglibc2 libgee-0.8-2 libgexiv2-2 libgom-1.0-0 libgpod-common libgpod4
liblirc-client0 liblua5.3-0 libmediaart-2.0-0 libmtp-0.1-1 libqqwing2v5 libraw16
libsgutils2-2 libssh-4 libsuitesparseconfig5 libvncclient1 libwinpr2-2 libxapian30 lp-solve
media-player-info python3-mako python3-markupsafe syslinux syslinux-common syslinux-legacy
usb-creator-common
```

[그림 3-3] JDK9 설치



```
#java
export JAVA_HOME=/usr/lib/jvm/java-8-openjdk-amd64
export PATH=$JAVA_HOME/bin:$PATH
export CLASS_PATH=$JAVA_HOME/lib:$CLASS_PATH
```

[그림 3-4] 환경 변수 설정



```
eunjin@eunjin-PC:~$ java -version
openjdk version "1.8.0_242"
OpenJDK Runtime Environment (build 1.8.0_242-8u242-b08-0ubuntu3-18.04-b08)
OpenJDK 64-Bit Server VM (build 25.242-b08, mixed mode)
```

[그림 3-5] Java 설치 완료



### 3.3.2. g++ 설치

Rocks DB를 설치하기 위해 g++를 설치한다. 먼저 g++ --version 명령어를 통해 g++의 설치여부를 확인한 뒤 sudo apt-get install g++ 명령어를 통해 g++ 설치를 진행하고, 설치 여부를 확인한다.

```
eunjin@eunjin-PC:~$ g++ --version  
  
Command 'g++' not found, but can be installed with:  
  
sudo apt install g++
```

[그림 3-6] g++ 설치여부 확인

```
eunjin@eunjin-PC:~$ sudo apt-get install g++  
[sudo] eunjin의 암호:  
패키지 목록을 읽는 중입니다... 완료  
의존성 트리를 만드는 중입니다  
상태 정보를 읽는 중입니다... 완료
```

[그림 3-7] g++ 설치

```
eunjin@eunjin-PC:~$ g++ --version  
g++ (Ubuntu 7.5.0-3ubuntu1~18.04) 7.5.0  
Copyright (C) 2017 Free Software Foundation, Inc.  
This is free software; see the source for copying conditions. There is NO  
warranty; not even for MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.
```

[그림 3-8] g++ 설치 완료

### 3.3.3. git 설치

git에서 YCSB 벤치마크 툴을 복제하기 위해 git을 설치한다. 먼저 git --version 명령어를 통해 git의 설치여부를 확인한 뒤 sudo apt install git 명령어를 통해 git 설치를 진행하고, 설치 여부를 확인한다.

```
eunjin@eunjin-PC:~$ git --version  
Command 'git' not found, but can be installed with:  
sudo apt install git
```

[그림 3-9] git 설치여부 확인

```
eunjin@eunjin-PC:~$ sudo apt install git  
패키지 목록을 읽는 중입니다... 완료  
의존성 트리를 만드는 중입니다  
상태 정보를 읽는 중입니다... 완료
```

[그림 3-10] git 설치

```
eunjin@eunjin-PC:~$ git --version  
git version 2.17.1
```

[그림 3-11] git 설치 완료

### 3.3.4. YCSB-Rocks DB 설치

git에서 YCSB를 복제 후 YCSB가 정상적으로 설치되었는지 YCSB cd YCSB 명령어를 통해 설치 디렉토리로 이동하여 ./bin/ycsb 명령어를 통해 확인한다. 정상적으로 설치된 것을 확인한 후 Rocks DB를 설치하기 위해 mvn -version 명령어를 통해 Maven의 설치여부를 확인하고, sudo apt install maven 명령어를 통해 Maven 설치를 진행 및 확인한다. 마지막으로 Rocks DB를 설치하기 위해 YCSB 설치 디렉토리로 이동하여 mvn -pl site.ycsb:rocksdB-binding -am clean package 명령어를 통해 설치를 진행한다.

```
eunjin@eunjin-PC:~$ git clone https://github.com/brianfrankcooper/YCSB.git
'YCSB'에 복제합니다...
remote: Enumerating objects: 20365, done.
remote: Total 20365 (delta 0), reused 0 (delta 0), pack-reused 20365
오브젝트를 받는 중: 100% (20365/20365), 31.51 MiB | 3.64 MiB/s, 완료.
델타를 알아내는 중: 100% (7950/7950), 완료.
```

[그림 3-12] git에서 YCSB 복제

```
eunjin@eunjin-PC:~$ cd YCSB
eunjin@eunjin-PC:~/YCSB$ ./bin/ycsb
usage: ./bin/ycsb command database [options]

Commands:
  load          Execute the load phase
```

[그림 3-13] YCSB 설치 확인

```
eunjin@eunjin-PC:~$ sudo apt install maven
패키지 목록을 읽는 중입니다... 완료
의존성 트리를 만드는 중입니다
상태 정보를 읽는 중입니다... 완료
```

[그림 3-14] Maven 설치

```
eunjin@eunjin-PC:~$ cd YCSB
eunjin@eunjin-PC:~/YCSB$ mvn -pl site.ycsb:rocksdB-binding -am clean package
[INFO] Scanning for projects...
[INFO] -----
[INFO] Reactor Build Order:
[INFO]
[INFO] YCSB Root [pom]
```

[그림 3-15] Rocks DB 설치

## 4. 실험

### 4.1. Cgroups blkio weight 100, 200, 400, 800 성능 측정

#### 4.1.1. 디스크 추가

##### 4.1.1.1. 디스크 확인

먼저 실험을 위해 준비한 SSD를 마운트하기 위해 SSD를 컴퓨터에 연결시키고, Ubuntu에 연결돼 있는 SSD 목록 및 이름을 확인한다. SSD 목록 확인은 `fdisk -l` 명령어를 사용하면 현재 Ubuntu에 연결되어 있는 모든 디스크의 목록을 확인할 수 있다. 아래 [그림 4-1]을 보면 `sda`라는 238.5 GiB의 새로운 SSD가 장착되어 있는 것을 확인할 수 있다. SSD의 이름은 `sda`, `sdb`, `sdc` 등 장착된 메모리에 따라 다르게 표시될 수 있다.

```
root@eunjin-PC:~# fdisk -l
Disk /dev/nvme0n1: 477 GiB, 512110190592 bytes, 1000215216 sectors
Units: sectors of 1 * 512 = 512 bytes
Sector size (logical/physical): 512 bytes / 512 bytes
I/O size (minimum/optimal): 512 bytes / 512 bytes
Disklabel type: gpt
Disk identifier: 3F8672C5-4D2D-4F95-9339-C7DDF7F1110C

Device            Start      End  Sectors  Size Type
/dev/nvme0n1p1      2048    1085439    1083392    529M Windows recovery environmen
/dev/nvme0n1p2   1085440    1290239     204800    100M EFI System
/dev/nvme0n1p3   1290240    1323007      32768     16M Microsoft reserved
/dev/nvme0n1p4   1323008   874385407  873062400  416.3G Microsoft basic data
/dev/nvme0n1p5   874385408   998213631  123828224    59G Linux filesystem
/dev/nvme0n1p6  998213632 1000214527    2000896    977M Linux swap

Disk /dev/sda: 238.5 GiB, 256060514304 bytes, 500118192 sectors
Units: sectors of 1 * 512 = 512 bytes
Sector size (logical/physical): 512 bytes / 512 bytes
I/O size (minimum/optimal): 512 bytes / 512 bytes
Disklabel type: dos
Disk identifier: 0xc0723420
root@eunjin-PC:~#
```

[그림 4-1] 디스크 장착 확인

#### 4.1.1.2. 파티션 설정

SSD가 정상적으로 장착 되었으면, SSD에 4개의 파티션을 생성해준다. fdisk /dev/sda 명령어를 입력하면 파티션을 설정할 수 있다. 먼저 n을 눌러 새로운 파티션을 생성하고, p(primary)를 눌러 주파티션으로 설정한다. 그 다음 1번 파티션을 선택하여 2048부터 +50G를 해줌으로써 104859647까지 사용하게 된다. 마지막으로 p를 누르면 /dev/sda1이라는 새로운 파티션이 생성되었음을 확인할 수 있다. 위와 같은 방법으로 4번 반복해주면 4개의 새로운 파티션이 생성된다. 아래 [그림 4-3]을 보면 /dev/sda1부터 /dev/sda4까지 4개의 파티션이 생성되었음을 확인할 수 있다.

```
root@eunjin-PC:~# fdisk /dev/sda
Welcome to fdisk (util-linux 2.27.1).
Changes will remain in memory only, until you decide to write them.
Be careful before using the write command.

Command (m for help): n
Partition type
  p   primary (0 primary, 0 extended, 4 free)
  e   extended (container for logical partitions)
Select (default p): p
Partition number (1-4, default 1): 1
First sector (2048-500118191, default 2048): 2048
Last sector, +sectors or +size{K,M,G,T,P} (2048-500118191, default 500118191): +50G

Created a new partition 1 of type 'Linux' and of size 50 GiB.

Command (m for help): p
Disk /dev/sda: 238.5 GiB, 256060514304 bytes, 500118192 sectors
Units: sectors of 1 * 512 = 512 bytes
Sector size (logical/physical): 512 bytes / 512 bytes
I/O size (minimum/optimal): 512 bytes / 512 bytes
Disklabel type: dos
Disk identifier: 0xc0723420

Device     Boot Start      End  Sectors Size Id Type
/dev/sda1             2048 104859647 104857600  50G 83 Linux
```

[그림 4-2] 파티션 생성 과정

```
Disk /dev/sda: 238.5 GiB, 256060514304 bytes, 500118192 sectors
Units: sectors of 1 * 512 = 512 bytes
Sector size (logical/physical): 512 bytes / 512 bytes
I/O size (minimum/optimal): 512 bytes / 512 bytes
Disklabel type: dos
Disk identifier: 0xc0723420

Device     Boot      Start      End  Sectors Size Id Type
/dev/sda1             2048 104859647 104857600  50G 83 Linux
/dev/sda2    104859648 209717247 104857600  50G 83 Linux
/dev/sda3    209717248 314574847 104857600  50G 83 Linux
/dev/sda4    314574848 419432447 104857600  50G 83 Linux
```

[그림 4-3] 4개의 파티션 생성

#### 4.1.1.3. 파티션 포맷

새로 생성한 4개의 파티션에 각각 ext4 파일 시스템으로 모두 포맷해 준다. Ubuntu에서 지원하는 파일 시스템(예 : bfs, ext2, ext3, ext4, ntfs, fat, vfat, 등.) 이 있지만, 현재 리눅스에서 가장 보편적으로 사용되고 있는 ext4 파일 시스템을 사용하였다. 파티션 포맷을 진행할 때는 파티션 나누기가 완료된 상태이기 때문에 sda가 아닌 sda1을 입력하여 진행해야 한다. mkfs.ext4 /dev/sda1 명령어를 사용하면 sda1을 ext4로 포맷하게 된다. 이와 같은 과정을 sda4까지 반복한다.

```
root@eunjin-PC:~# mkfs.ext4 /dev/sda1
mke2fs 1.42.13 (17-May-2015)
/dev/sda1 contains a ext4 file system
last mounted on /ssd/1 on Tue Nov 19 11:18:28 2019
Proceed anyway? (y,n) y
/dev/sda1 is mounted; will not make a filesystem here!
root@eunjin-PC:~# mkfs.ext4 /dev/sda2
mke2fs 1.42.13 (17-May-2015)
/dev/sda2 contains a ext4 file system
last mounted on /ssd/2 on Tue Nov 19 11:18:28 2019
Proceed anyway? (y,n) y
/dev/sda2 is mounted; will not make a filesystem here!
root@eunjin-PC:~# mkfs.ext4 /dev/sda3
mke2fs 1.42.13 (17-May-2015)
/dev/sda3 contains a ext4 file system
last mounted on /ssd/3 on Tue Nov 19 11:18:28 2019
Proceed anyway? (y,n) y
/dev/sda3 is mounted; will not make a filesystem here!
root@eunjin-PC:~# mkfs.ext4 /dev/sda4
mke2fs 1.42.13 (17-May-2015)
/dev/sda4 contains a ext4 file system
last mounted on /ssd/4 on Tue Nov 19 11:18:28 2019
Proceed anyway? (y,n) y
/dev/sda4 is mounted; will not make a filesystem here!
root@eunjin-PC:~#
```

[그림 4-4] 각각의 파티션에 ext4 파일 시스템 포맷



#### 4.1.1.4. 디스크 자동 마운트

먼저 blkid 명령어를 사용해 마운트 할 디스크의 UUID 및 TYPE를 확인하고, mkdir 명령어를 사용해 4개의 파티션을 마운트 할 디렉토리를 생성한다. 그 다음 vi 명령어를 사용하여 fstab 파일에 4개의 파티션을 추가한다. 이때 UUID 및 TYPE을 형식(예 : UUID=00000000-0000-0000-000000000000 /ssd/1 ext4 defaults 0 0)에 맞춰 적어준다. 이후 mount -a 와 df -h 명령어를 입력하면 sda1부터 sda4까지 4개의 파티션이 새로 생성한 디렉토리에 마운트 되어 있는 것을 확인할 수 있다.

```
root@eunjin-PC:~# blkid
/dev/sda1: UUID="349e9448-5218-49cf-b8cd-87306fb23fd5" TYPE="ext4" PARTUUID="c0723420-01"
/dev/nvme0n1p1: LABEL="M-KM-3M-5M-JM-5M-" UUID="4C265FB4265F9E30" TYPE="ntfs" PARTLABEL="Basic data partition" PARTUUID="d72a8694-f414-4e89-b849-7db6602dcb76"
/dev/nvme0n1p2: UUID="9460-3154" TYPE="vfat" PARTLABEL="EFI system partition" PARTUUID="b5653ab5-9178-4ce1-b800-21750c140cd5"
/dev/nvme0n1p4: UUID="864C613A4C612661" TYPE="ntfs" PARTLABEL="Basic data partition" PARTUUID="aea7ed37-846a-44eb-82c9-7c0c5119be35"
/dev/nvme0n1p5: UUID="7e9a3b66-e5d7-4c61-80e9-a8e77c6b5b71" TYPE="ext4" PARTUUID="c608805d-82e3-43b6-b518-aa381f392a97"
/dev/nvme0n1p6: UUID="c7cb9349-ddd6-45b8-8709-0fa02f3a9169" TYPE="swap" PARTUUID="99383922-50cb-4ed1-8214-49c266b3f234"
/dev/sda2: UUID="5aa9d5c0-238b-4848-a6b3-c49e4fa440c5" TYPE="ext4" PARTUUID="c0723420-02"
/dev/sda3: UUID="ccd2732d-c163-46c3-857d-a2361f624810" TYPE="ext4" PARTUUID="c0723420-03"
/dev/sda4: UUID="9799aede-835d-4e77-b11a-204257413e9a" TYPE="ext4" PARTUUID="c0723420-04"
/dev/nvme0n1: PTUUID="3f8672c5-4d2d-4f95-9339-c7ddf7f1110c" PTTYPE="gpt"
/dev/nvme0n1p3: PARTLABEL="Microsoft reserved partition" PARTUUID="50538b6d-cba6-4ed5-a30e-1f46cd5405e8"
root@eunjin-PC:~# vi /etc/fstab
root@eunjin-PC:~# mount -a
root@eunjin-PC:~# df -h
```

Filesystem	Size	Used	Avail	Use%	Mounted on
udev	7.8G	0	7.8G	0%	/dev
tmpfs	1.6G	9.6M	1.6G	1%	/run
/dev/nvme0n1p5	58G	9.3G	46G	17%	/
tmpfs	7.9G	316K	7.9G	1%	/dev/shm
tmpfs	5.0M	4.0K	5.0M	1%	/run/lock
tmpfs	7.9G	0	7.9G	0%	/sys/fs/cgroup
/dev/nvme0n1p2	96M	31M	66M	33%	/boot/efi
tmpfs	1.6G	76K	1.6G	1%	/run/user/1000
/dev/sda2	50G	1.1G	46G	3%	/ssd/2
/dev/sda1	50G	52M	47G	1%	/ssd/1
/dev/sda3	50G	1.1G	46G	3%	/ssd/3
/dev/sda4	50G	1.1G	46G	3%	/ssd/4

```
root@eunjin-PC:~#
```

[그림 4-5] 디스크 자동 마운트

## 4.1.2. Cgroups Block I/O 컨트롤러 사용

### 4.1.2.1. Block I/O 컨트롤러 실행 조건

Cgroups 및 blkio 서브시스템을 사용하기 위해서는 먼저 블록 I/O 컨트롤러 활성화, 블록 레이어 조절 활성화, CFQ 그룹 스케줄링 사용을 활성화시킨 후 커널이 Cgroups를 지원하는지 확인한다. (blkio의 enabled가 1로 되어 있으면 완료) 마지막으로 스케줄러가 CFQ로 되어 있는지 확인한다. 만약 스케줄러가 noop 또는 deadline으로 되어 있을 경우 `echo cfq > /sys/block/sda/queue/scheduler` 명령어를 통해 CFQ로 바꿔준다.

```
root@eunjin-PC:/home/eunjin# CONFIG_BLK_CGROUP=Y
root@eunjin-PC:/home/eunjin# CONFIG_BLK_DEV_THROTTLING=Y
root@eunjin-PC:/home/eunjin# CONFIG_CFQ_GROUP_IOSCHED=Y
root@eunjin-PC:/home/eunjin# cat /proc/cgroups
#subsys_name hierarchy num_cgroups enabled
cpuset 11 1 1
cpu 9 1 1
cpuacct 9 1 1
blkio 8 1 1
memory 7 1 1
devices 5 78 1
freezer 6 1 1
net_cls 3 1 1
perf_event 12 1 1
net_prio 3 1 1
hugetlb 4 1 1
pids 2 5 1
rdma 10 1 1
root@eunjin-PC:/home/eunjin# cat /sys/block/sda/queue/scheduler
noop deadline [cfq]
root@eunjin-PC:/home/eunjin#
```

[그림 4-6] Cgroups blkio 서브시스템 사용 준비



#### 4.1.2.2. Cgroups 마운트 및 Blkio I/O 컨트롤러 사용

Cgroups의 blkio 서브시스템을 사용하기 위해 Cgroups 루트 디렉토리와 blkio 하위 시스템을 마운트 해준 후 blkio 컨트롤러에서 4개의 task 그룹을 생성해준다. Cgroups는 디렉토리 생성과 동시에 특수 파일을 배치하는데 ls [/blkio 컨트롤러에서 생성한 그룹 경로]를 지정해주면 확인할 수 있다. (예: ls /sys/fs/cgroup/blkio/task1)

```
root@eunjin-PC:/home/eunjin# mount -t tmpfs cgroup_root /sys/fs/cgroup
root@eunjin-PC:/home/eunjin# mkdir /sys/fs/cgroup/blkio
root@eunjin-PC:/home/eunjin# mount -t cgroup -o blkio none /sys/fs/cgroup/blkio
root@eunjin-PC:/home/eunjin# mkdir /sys/fs/cgroup/blkio/task1
root@eunjin-PC:/home/eunjin# mkdir /sys/fs/cgroup/blkio/task2
root@eunjin-PC:/home/eunjin# mkdir /sys/fs/cgroup/blkio/task3
root@eunjin-PC:/home/eunjin# mkdir /sys/fs/cgroup/blkio/task4
```

[그림 4-7] Cgroups 마운트 및 task1-4 그룹 생성

#### 4.1.2.3. Cgroups에 프로세스 할당

blkio 컨트롤러에서 생성한 4개의 task 그룹에 각각의 TID를 할당해 주어야 한다. 먼저 `cat /sys/fs/cgroups/blkio/tasks` 명령어를 통해 프로세스 목록을 확인한 후 `echo [TID] > /sys/fs/cgroup/blkio/task1/tasks` 명령어를 통해 임의의 TID를 4개의 task 그룹에 각각 할당해준다. 마지막으로 `cat` 명령어를 통해 4개의 task 그룹에 지정한 프로세스가 할당 되었는지 확인한다. 아래 [그림 4-10]를 보면 임의로 지정한 TID 4개가 정상적으로 할당 된 것을 확인할 수 있다.

```
root@eunjin-PC:/home/eunjin# cat /sys/fs/cgroup/blkio/tasks
1
2
4
6
7
8
9
10
11
12
13
14
15
16
18
19
20
21
```

[그림 4-8] 프로세스 목록 확인

```
7331
7335
7350
7351
7419
7423
7449
7508
root@eunjin-PC:/home/eunjin# echo 2801 > /sys/fs/cgroup/blkio/task1/tasks
root@eunjin-PC:/home/eunjin# echo 2802 > /sys/fs/cgroup/blkio/task2/tasks
root@eunjin-PC:/home/eunjin# echo 2803 > /sys/fs/cgroup/blkio/task3/tasks
root@eunjin-PC:/home/eunjin# echo 2804 > /sys/fs/cgroup/blkio/task4/tasks
```

[그림 4-9] 4개의 task 그룹에 임의의 TID 할당

```
root@eunjin-PC:/home/eunjin# cat /sys/fs/cgroup/blkio/task1/tasks
2801
root@eunjin-PC:/home/eunjin# cat /sys/fs/cgroup/blkio/task2/tasks
2802
root@eunjin-PC:/home/eunjin# cat /sys/fs/cgroup/blkio/task3/tasks
2803
root@eunjin-PC:/home/eunjin# cat /sys/fs/cgroup/blkio/task4/tasks
2804
root@eunjin-PC:/home/eunjin#
```

[그림 4-10] 임의로 지정한 TID가 정상적으로 할당 되었는지 확인

#### 4.1.2.4. weight 비율 할당

4개의 task 그룹에 blkio.weight 변수를 통해 가중치(weight) 비율을 설정하고 cat 명령어를 통해 확인한다. 이때, 비율은 100, 200, 400, 800으로 설정한다. 즉, task1 그룹에 가중치 100, task2 그룹에 가중치 200, task3 그룹에 가중치 400, task4 그룹에 가중치 400을 설정한다. 이후 lscgroup 명령어를 통해 4개의 task 그룹과 blkio 서브시스템이 마운트 되었는지 확인한다.

```
root@eunjin-PC:/home/eunjin# echo 100 > /sys/fs/cgroup/blkio/task1/blkio.weight
root@eunjin-PC:/home/eunjin# echo 200 > /sys/fs/cgroup/blkio/task2/blkio.weight
root@eunjin-PC:/home/eunjin# echo 400 > /sys/fs/cgroup/blkio/task3/blkio.weight
root@eunjin-PC:/home/eunjin# echo 800 > /sys/fs/cgroup/blkio/task4/blkio.weight
root@eunjin-PC:/home/eunjin# cat /sys/fs/cgroup/blkio/task1/blkio.weight
100
root@eunjin-PC:/home/eunjin# cat /sys/fs/cgroup/blkio/task2/blkio.weight
200
root@eunjin-PC:/home/eunjin# cat /sys/fs/cgroup/blkio/task3/blkio.weight
400
root@eunjin-PC:/home/eunjin# cat /sys/fs/cgroup/blkio/task4/blkio.weight
800
root@eunjin-PC:/home/eunjin#
```

[그림 4-11] wieght 설정 및 확인

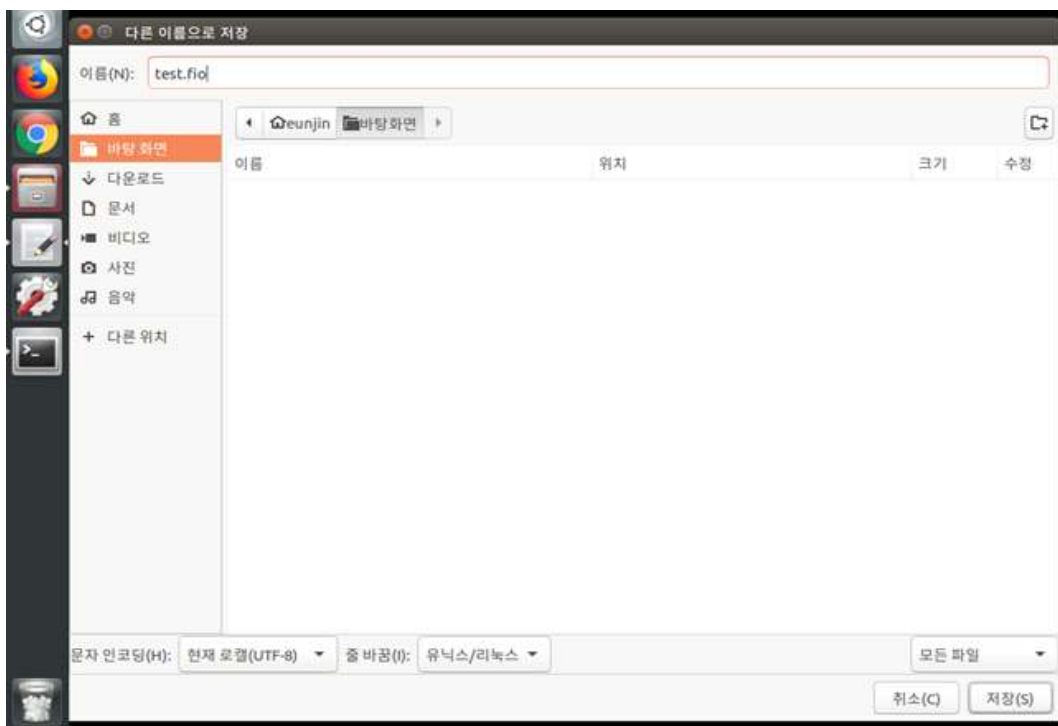
```
root@eunjin-PC:/home/eunjin# lscgroup
blkio:/
blkio:/task3
blkio:/task1
blkio:/task4
blkio:/task2
root@eunjin-PC:/home/eunjin#
```

[그림 4-12] blkio 서브시스템 마운트 확인

### 4.1.3. FIO 툴을 사용해 IOPS 및 BW 측정

#### 4.1.3.1. FIO Job File 생성

Job File을 생성하기 위해 먼저 텍스트 편집기를 열고 옵션을 정의한 후 원하는 위치에 저장한다. 본 실험에서는 바탕화면에 저장을 해주었다.



[그림 4-13] Job File 저장



[그림 4-14] Job File 생성

#### 4.1.3.2. FIO Job File 명령어

본 실험에서는 아래와 같은 FIO 옵션을 사용하였다.

[표 4-1] FIO Job File 옵션

<pre>[global] bs=4k ioengine=libaio iodepth=32 direct=1 rw=read time_based runtime=10800 cgroup_nodelte=1  [test1] directory=/ssd/1 size=1G cgroup=task1</pre>	<pre>[test2] directory=/ssd/2 size=1G cgroup=task2  [test3] directory=/ssd/3 size=1G cgroup=task3  [test4] directory=/ssd/4 size=1G cgroup=task4</pre>
--	--

[global] : 약속된 이름 (변경 불가능)

(global에 정의되어 있는 옵션들은 공통 옵션)

bs=4k : 테스트할 블록의 크기

ioengine=libaio : 프로세스가 I/O를 발생하는 방법 지정

(libaio : Linux AIO)

iodepth=32 : AIO를 사용할 때 동시에 발생할 I/O 수의 상한선

(대량의 I/O가 발생해 경합하는 것을 방지하기 위함)

direct=1 : Direct I/O 사용

rw=read : 테스트 종류

read : 순차읽기, write : 순차 쓰기

randread : 랜덤 읽기, randwrite : 랜덤 쓰기

time\_based : runtime 시간만큼 I/O를 실행 (파일 사이즈 필요 X)

runtime=10800 : 테스트 진행 시간 (단위 : 초)

cgroup\_nodelate=1 : Fio 작업 완료 후 cgroup을 남겨두기 위함  
(기본 값 false(cgroup 삭제))  
[test1] : 테스트 이름 (변경 가능)  
directory=/ssd/1 : 디렉토리 경로(SSD 마운트 한 디렉토리)  
size=1G : Job 당 총 파일 사이즈  
(Job이 완료하기까지 I/O를 수행할 데이터양)  
cgroup=task1 : cgroup과 task1 파일 연결

#### 4.1.3.3. FIO 실행

FIO 벤치마크를 실행하기 위해 먼저 cd 명령어를 통해 FIO Job File 을 저장한 위치로 이동한 뒤 fio [FIO job File 이름].fio 명령어를 통해 FIO 벤치마크를 실행한다. FIO 벤치마크를 실행하면 아래의 [그림 4-15]와 같은 실행결과 화면이 나타난다. 본 실험에서는 IOPS, BW를 기록하며 실행을 진행하였다. [그림 4-16]의 경우 test1 그룹의 IOPS는 6594(B), BW는 26377(KB/s)이고 test2 그룹의 IOPS 13113(B), BW 52452(KB/s)이다.

```
root@eunjin-PC:/home/eunjin# cd 바탕화면
root@eunjin-PC:/home/eunjin/바탕화면# fio test.fio
test1: (g=0): rw=read, bs=4K-4K/4K-4K/4K-4K, ioengine=libaio, iodepth=32
test2: (g=0): rw=read, bs=4K-4K/4K-4K/4K-4K, ioengine=libaio, iodepth=32
test3: (g=0): rw=read, bs=4K-4K/4K-4K/4K-4K, ioengine=libaio, iodepth=32
test4: (g=0): rw=read, bs=4K-4K/4K-4K/4K-4K, ioengine=libaio, iodepth=32
fio-2.2.6
Starting 4 processes
test1: Laying out IO file(s) (1 file(s) / 1024MB)
Jobs: 4 (f=4): [R(4)] [100.0% done] [387.8MB/0KB/0KB /s] [99.3K/0/0 iops] [eta 00m:00s]
test1: (groupid=0, jobs=1): err= 0: pld=3043: Tue Nov 19 13:44:25 2019
  read : io=4636.9MB, bw=26377KB/s, iops=6594, runt=180007msec
    slat (usec): min=0, max=207, avg= 1.92, stdev= 1.58
    clat (usec): min=88, max=400290, avg=4850.28, stdev=35825.57
    lat (usec): min=91, max=400291, avg=4852.26, stdev=35825.53
    clat percentiles (usec):
      | 1.00th=[ 225], 5.00th=[ 310], 10.00th=[ 314], 20.00th=[ 314],
      | 30.00th=[ 318], 40.00th=[ 318], 50.00th=[ 318], 60.00th=[ 318],
      | 70.00th=[ 322], 80.00th=[ 322], 90.00th=[ 326], 95.00th=[ 330],
      | 99.00th=[222208], 99.50th=[321536], 99.90th=[321536], 99.95th=[321536],
      | 99.99th=[391168]
    bw (KB /s): min=14307, max=35912, per=6.69%, avg=26434.51, stdev=2380.12
    lat (usec) : 100=0.01%, 250=1.34%, 500=95.65%, 750=1.25%, 1000=0.12%
    lat (msec) : 2=0.02%, 250=0.69%, 500=0.94%
    cpu       : usr=0.64%, sys=2.05%, ctx=314832, majf=0, minf=42
    IO depths : 1=0.1%, 2=0.1%, 4=0.1%, 8=0.1%, 16=0.1%, 32=100.0%, >=64=0.0%
    submit    : 0=0.0%, 4=100.0%, 8=0.0%, 16=0.0%, 32=0.0%, 64=0.0%, >=64=0.0%
    complete  : 0=0.0%, 4=100.0%, 8=0.0%, 16=0.0%, 32=0.1%, 64=0.0%, >=64=0.0%
    issued    : total=r=1187020/w=0/d=0, short=r=0/w=0/d=0, drop=r=0/w=0/d=0
    latency   : target=0, window=0, percentile=100.00%, depth=32
test2: (groupid=0, jobs=1): err= 0: pld=3044: Tue Nov 19 13:44:25 2019
  read : io=9220.8MB, bw=52452KB/s, iops=13113, runt=180013msec
    slat (usec): min=0, max=303116, avg= 2.74, stdev=478.54
    clat (usec): min=61, max=377693, avg=2435.71, stdev=23746.67
    lat (usec): min=63, max=378046, avg=2438.52, stdev=23751.47
    clat percentiles (usec):
      | 1.00th=[ 223], 5.00th=[ 310], 10.00th=[ 314], 20.00th=[ 314],
      | 30.00th=[ 318], 40.00th=[ 318], 50.00th=[ 318], 60.00th=[ 318],
      | 70.00th=[ 322], 80.00th=[ 322], 90.00th=[ 326], 95.00th=[ 326],
      | 99.00th=[ 322], 99.50th=[ 322], 99.90th=[ 326], 99.95th=[ 326],
      | 99.99th=[ 326]
```

[그림 4-15] FIO 벤치마크 실행

한 번의 실험이 완료되면 SSD를 마운트 한 위치로 이동하여 `df -h` 명령어를 통해 디스크의 용량을 확인하고, 실험을 통해 만들어진 파일을 `rm` 명령어를 통해 삭제한다.

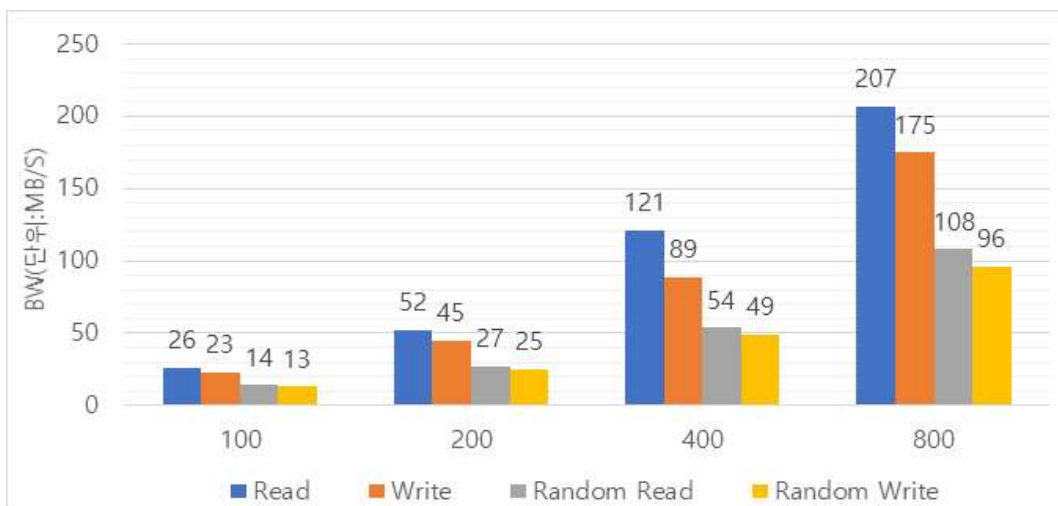
```
root@eunjin-PC:/home/eunjin/바탕화면# cd /ssd/1
root@eunjin-PC:/ssd/1# ls
lost+found  test1.0.0
root@eunjin-PC:/ssd/1# rm test*
root@eunjin-PC:/ssd/1# cd /ssd/2
root@eunjin-PC:/ssd/2# rm test*
root@eunjin-PC:/ssd/2# cd /ssd/3
root@eunjin-PC:/ssd/3# rm test*
root@eunjin-PC:/ssd/3# cd /ssd/4
root@eunjin-PC:/ssd/4# rm test*
```

[그림 4-16] SSD를 마운트 한 디렉토리에 생성되어 있는 파일 전체 삭제

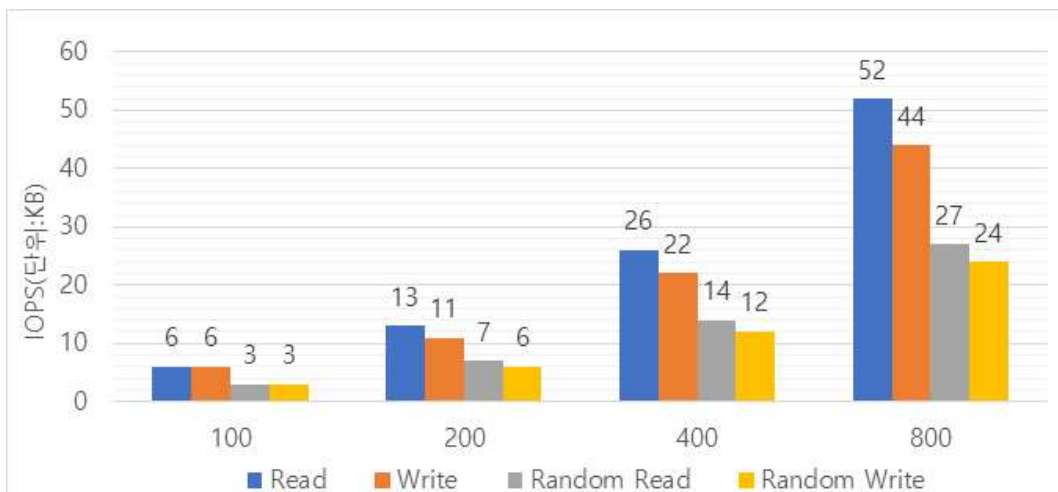


#### 4.1.4. 실험 결과

아래 [그림 4-17]과 [그림 4-18]은 4개의 task 그룹에 가중치 100, 200, 400, 800을 설정하여 실험을 진행한 FIO BW와 IOPS의 결과 그래프이다. 가중치가 100일 때, 200일 때, 400일 때, 800일 때 모두 시스템 자원에 대한 분배가 일정하게 이루어진다는 사실을 확인하였다.



[그림 4-17] FIO 결과 그래프 (BW)



[그림 4-18] FIO 결과 그래프 (IOPS)

## 4.2. YCSB-Rocks DB를 이용한 FIO-Cgroups 성능 측정

### 4.2.1. 디스크 추가

#### 4.2.1.1. 디스크 확인

먼저 실험을 위해 준비한 SSD를 마운트하기 위해 SSD를 컴퓨터에 연결시키고, Ubuntu에 연결돼 있는 SSD 목록 및 이름을 확인한다. SSD 목록 확인은 `fdisk -l` 명령어를 사용하면 현재 Ubuntu에 연결되어 있는 모든 디스크의 목록을 확인할 수 있다. 아래 [그림 4-19]를 보면 `sda`라는 238.5 GiB의 새로운 SSD가 장착되어 있는 것을 확인할 수 있다. SSD의 이름은 `sda`, `sdb`, `sdc` 등 장착된 메모리에 따라 다르게 표시될 수 있다.

```
eunjin@eunjin-PC:~$ sudo fdisk -l
[sudo] eunjin의 암호:

Disk /dev/nvme0n1: 477 GiB, 512110190592 bytes, 1000215216 sectors
Units: sectors of 1 * 512 = 512 bytes
Sector size (logical/physical): 512 bytes / 512 bytes
I/O size (minimum/optimal): 512 bytes / 512 bytes
Disklabel type: gpt
Disk identifier: AE6EE1CF-3970-481A-B2DD-07C5422A0108

Device                Start      End    Sectors   Size Type
/dev/nvme0n1p1         2048     1085439  1083392   529M Windows recovery environment
/dev/nvme0n1p2      1085440     1290239   204800   100M EFI System
/dev/nvme0n1p3      1290240     1323007    32768    16M Microsoft reserved
/dev/nvme0n1p4      1323008     874385407 873062400 416.3G Microsoft basic data
/dev/nvme0n1p5    998213632 1000214527  2000896   977M Linux swap
/dev/nvme0n1p6    874385408     998213631 123828224   59G Linux filesystem

Partition table entries are not in disk order.

Disk /dev/sda: 238.5 GiB, 256060514304 bytes, 500118192 sectors
Units: sectors of 1 * 512 = 512 bytes
Sector size (logical/physical): 512 bytes / 512 bytes
I/O size (minimum/optimal): 512 bytes / 512 bytes
Disklabel type: dos
Disk identifier: 0xc0723420
```

[그림 4-19] 디스크 장착 확인

#### 4.2.1.2. 파티션 설정

SSD가 정상적으로 장착된 것을 확인했다면, `sudo fdisk /dev/sda` 명령어를 통해 파티션을 설정한다. `n`을 눌러 하나의 새로운 파티션을 생성하고, Command가 나올 때까지 Enter를 눌러준다. Command가 나오면 `w`를 입력하여 저장 후 종료한다.

```
eunjin@eunjin-PC:~$ sudo fdisk /dev/sda

Welcome to fdisk (util-linux 2.31.1).
Changes will remain in memory only, until you decide to write them.
Be careful before using the write command.

Command (m for help): |
```

[그림 4-20] 파티션 설정

```
Command (m for help): n
Partition type
  p   primary (0 primary, 0 extended, 4 free)
  e   extended (container for logical partitions)
Select (default p):

Using default response p.
Partition number (1-4, default 1):
First sector (2048-500118191, default 2048):
Last sector, +sectors or +size{K,M,G,T,P} (2048-500118191, default 500118191):

Created a new partition 1 of type 'Linux' and of size 238.5 GiB.
Partition #1 contains a ext4 signature.
Do you want to remove the signature? [Y]es/[N]o: y

The signature will be removed by a write command.

Command (m for help): w
The partition table has been altered.
Calling ioctl() to re-read partition table.
Syncing disks.
```

[그림 4-21] 하나의 파티션 설정 과정

#### 4.2.1.3. 파티션 포맷

새로 생성한 하나의 파티션에 ext4 파일 시스템을 포맷해준다. Ubuntu에서 지원하는 파일 시스템(예 : bfs, ext2, ext3, ext4, ntfs, fat, vfat, 등.) 이 있지만, 현재 리눅스에서 가장 보편적으로 사용되고 있는 ext4 파일 시스템을 사용하였다. mkfs.ext4 /dev/sda1 명령어를 사용하면 sda1을 ext4로 포맷하게 된다. 파티션 포맷을 진행할 때는 파티션 나누기가 완료된 상태이기 때문에 sda가 아닌 sda1을 입력하여 진행해야 한다.

```
eunjin@eunjin-PC:~$ sudo mkfs.ext4 /dev/sda1
mke2fs 1.44.1 (24-Mar-2018)
Discarding device blocks: done
Creating filesystem with 62514518 4k blocks and 15630336 inodes
Filesystem UUID: 3227abf4-9aa9-4731-93f0-39010949b89d
Superblock backups stored on blocks:
    32768, 98304, 163840, 229376, 294912, 819200, 884736, 1605632, 2654208,
    4096000, 7962624, 11239424, 20480000, 23887872

Allocating group tables: done
Writing inode tables: done
Creating journal (262144 blocks): done
Writing superblocks and filesystem accounting information: done
```

[그림 4-22] 하나의 파티션에 ext4 파일 시스템 포맷

#### 4.2.1.4. 디스크 수동 마운트

먼저 `mkdir` 명령어를 통해 디스크를 마운트 할 하나의 디렉토리를 생성하고, `sudo mount /dev/sda /[디스크를 마운트 할 디렉토리 경로]` 명령어를 통해 디스크를 수동 마운트 한다. 마지막으로 정상적으로 `df -h` 명령어를 통해 SSD가 새로운 디렉토리에 마운트가 되었는지 확인한다.

```
eunjin@eunjin-PC:~$ sudo mkdir /mnt/test
```

[그림 4-23] 디스크를 마운트 할 디렉토리 생성

```
eunjin@eunjin-PC:~$ sudo mount /dev/sda1 /mnt/test
[sudo] eunjin의 암호:
eunjin@eunjin-PC:~$
```

[그림 4-24] 디스크 수동 마운트

```
eunjin@eunjin-PC:~$ df -h
Filesystem      Size  Used Avail Use% Mounted on
udev            7.8G   0    7.8G   0% /dev
tmpfs           1.6G  1.9M   1.6G   1% /run
/dev/nvme0n1p6  58G   7.7G   48G  14% /
tmpfs           7.9G   0    7.9G   0% /dev/shm
tmpfs           5.0M  4.0K   5.0M   1% /run/lock
tmpfs           7.9G   0    7.9G   0% /sys/fs/cgroup
/dev/loop0       43M   43M    0 100% /snap/gtk-common-themes/1313
/dev/loop1       89M   89M    0 100% /snap/core/7270
/dev/loop2       4.2M  4.2M    0 100% /snap/gnome-calculator/406
/dev/loop3       55M   55M    0 100% /snap/core18/1066
/dev/loop4       1.0M  1.0M    0 100% /snap/gnome-logs/61
/dev/loop5      150M  150M    0 100% /snap/gnome-3-28-1804/67
/dev/loop6       3.8M  3.8M    0 100% /snap/gnome-system-monitor/100
/dev/loop7       15M   15M    0 100% /snap/gnome-characters/296
/dev/nvme0n1p2   96M   31M   66M  33% /boot/efi
tmpfs           1.6G  24K   1.6G   1% /run/user/1000
/dev/sda1       234G  61M  222G   1% /mnt/test
```

[그림 4-25] 디스크 마운트 확인

## 4.2.2. Cgroups Block I/O 컨트롤러 사용

\* 4.1.2.과 동일한 방식과 옵션으로 실험을 진행하였다.

## 4.2.3. YCSB-Rocks DB를 이용해 FIO-Cgroups 성능 측정

### 4.2.3.1. Read

먼저 YCSB가 테스트에 사용할 데이터를 DB에 저장하기 위해 cd 명령어를 통해 YCSB의 workloads 디렉토리로 이동하여 vi 에디터를 사용해 workloada를 연다. a 키를 눌러 입력모드로 전환한 후 recordcount=100000000, operationcount= 100000000, readproportion=1로 변경하고 저장 후 종료한다.

```
eunjin@eunjin-PC:~$ cd YCSB/workloads
eunjin@eunjin-PC:~/YCSB/workloads$ vi workloada
```

[그림 4-26] YSCB 디렉토리로 이동 후 workloada를 여는 과정

```
recordcount=100000000
operationcount=100000000
workload=site.ycsb.workloads.CoreWorkload

readallfields=true

readproportion=1
updateproportion=0
scanproportion=0
insertproportion=0
```

[그림 4-27] Read를 하기 위한 workloada 옵션 변경



그 다음 `sudo ./bin/ycsb load rocksdb -s -P workloads/workloada > loadtest1.txt -p rocksdb.dir=/mnt/test -p maxexecutiontime=3600` 명령어를 사용하여 workload를 load한다.

```
eunjin@eunjin-PC:~/YCSB$ sudo ./bin/ycsb load rocksdb -s -P workloads/workloada
> loadtest1.txt -p rocksdb.dir=/mnt/test -p maxexecutiontime=3600
[WARN] Running against a source checkout. In order to get our runtime dependencies we'll have to invoke Maven. Depending on the state of your system, this may take ~30-45 seconds
[DEBUG] Running 'mvn -pl site.ycsb:rocksdb-binding -am package -DskipTests dependency:build-classpath -DincludeScope=compile -Dmdep.outputFilterFile=true'
java -cp /home/eunjin/YCSB/rocksdb/conf:/home/eunjin/YCSB/rocksdb/target/rocksdb-binding-0.18.0-SNAPSHOT.jar:/root/.m2/repository/org/apache/htrace/htrace-core4/4.1.0-incubating/htrace-core4-4.1.0-incubating.jar:/root/.m2/repository/org/slf4j/slf4j-api/1.7.25/slf4j-api-1.7.25.jar:/root/.m2/repository/net/jcip/jcip-annotations/1.0/jcip-annotations-1.0.jar:/root/.m2/repository/org/hdrhistogram/HdrHistogram/2.1.4/HdrHistogram-2.1.4.jar:/root/.m2/repository/org/codehaus/jackson/jackson-mapper-asl/1.9.4/jackson-mapper-asl-1.9.4.jar:/root/.m2/repository/org/rocksdb/rocksdbjni/6.2.2/rocksdbjni-6.2.2.jar:/root/.m2/repository/org/codehaus/jackson/jackson-core-asl/1.9.4/jackson-core-asl-1.9.4.jar:/home/eunjin/YCSB/core/target/core-0.18.0-SNAPSHOT.jar site.ycsb.Client -db site.ycsb.db.rocksdb.RocksDBClient -s -P workloads/workloada -p rocksdb.dir=/mnt/test -p maxexecutiontime=3600 -load
Command line: -db site.ycsb.db.rocksdb.RocksDBClient -s -P workloads/workloada -p rocksdb.dir=/mnt/test -p maxexecutiontime=3600 -load
YCSB Client 0.18.0-SNAPSHOT

Loading workload...
Starting test.
SLF4J: Failed to load class "org.slf4j.impl.StaticLoggerBinder".
```

[그림 4-28] workloada를 load하는 옵션 및 과정

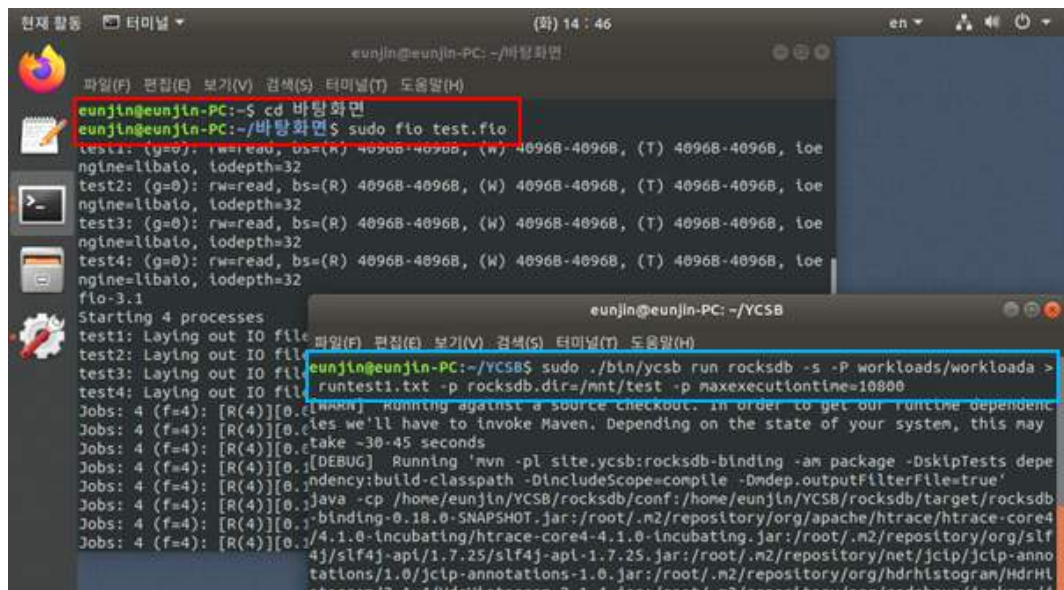
**빨간색 박스**는 rocksdb 데이터베이스를 사용하여 load하겠다는 의미이다. `-s` 옵션은 10초마다 operation이 얼마나 일어나고 있는지를 log로 보여준다.

**초록색 박스**는 workloads 폴더에 있는 workloada라는 workload를 사용하여 load하고, 그 결과를 loadtest1.txt라는 파일을 만들어서 저장하라는 의미이다. `-P` 옵션은 파라미터들을 넣는데 사용된다.

**파란색 박스**는 mount한 SSD의 경로를 지정해준 것이다.

**보라색 박스**는 최대 실행시간을 3600초로 제한하라는 옵션이다.

load가 끝나면 터미널 창을 하나 더 띄워서 FIO-Cgroups를 실행시키고, 그와 동시에 `sudo ./bin/ycsb run rocksdb -s -P workloads/workloada > runtest1.txt -p rocksdb.dir=/mnt/test -p maxexecutiontime=86400` 명령어를 사용하여 workload를 run한다. run은 load 단계에서 저장한 데이터를 이용하여 workload를 실행하는 단계이다. 즉, 터미널 창 두 개로 FIO-Cgroups와 YCSB- Rocks DB를 동시에 실행시킨다. YCSB의 run을 실행하는 자세한 옵션과 과정은 아래 29페이지에서 다룬다.



[그림 4-29] FIO와 YCSB 동시 실행

**빨간색 박스**는 FIO - Cgroup을 실행하기 위한 것이다. cd 바탕화면은 5.FIOJobFile생성에서 만든 test.fio가 저장되어 있는 위치로 이동한다는 뜻입니다. sudo fio test.fio는 test.fio를 실행하라는 의미이다.

**파란색 박스**는 YCSB의 run 단계로 YCSB의 load 단계에서 저장한 데이터를 이용하여 workload를 실행하는 단계이다.

YCSB의 run 단계는 load 단계에서 저장한 데이터를 이용하여 workload를 실행하는 단계이다.



```
eunjin@eunjin-PC:~/YCSB$ sudo ./bin/ycsb run rocksdb -s -P workloads/workloada >
runtest1.txt -p rocksdb.dir=/mnt/test -p maxexecutiontime=10800
[WARN] Running against a source checkout. In order to get our runtime dependenc
ies we'll have to invoke Maven. Depending on the state of your system, this may
take ~30-45 seconds
[DEBUG] Running 'mvn -pl site.ycsb:rocksdb-binding -am package -DskipTests depe
ndency:build-classpath -DincludeScope=compile -Dmdep.outputFilterFile=true'
java -cp /home/eunjin/YCSB/rocksdb/conf:/home/eunjin/YCSB/rocksdb/target/rocksdb
-binding-0.18.0-SNAPSHOT.jar:/root/.m2/repository/org/apache/htrace/htrace-core4
/4.1.0-incubating/htrace-core4-4.1.0-incubating.jar:/root/.m2/repository/org/slf
4j/slf4j-api/1.7.25/slf4j-api-1.7.25.jar:/root/.m2/repository/net/jcip/jcip-anno
tations/1.0/jcip-annotations-1.0.jar:/root/.m2/repository/org/hdrhistogram/HdrHi
stogram/2.1.4/HdrHistogram-2.1.4.jar:/root/.m2/repository/org/codehaus/jackson/j
ackson-mapper-asl/1.9.4/jackson-mapper-asl-1.9.4.jar:/root/.m2/repository/org/ro
cksdb/rocksdbjni/6.2.2/rocksdbjni-6.2.2.jar:/root/.m2/repository/org/codehaus/ja
ckson/jackson-core-asl/1.9.4/jackson-core-asl-1.9.4.jar:/home/eunjin/YCSB/core/t
arget/core-0.18.0-SNAPSHOT.jar site.ycsb.Client -db site.ycsb.db.rocksdb.RocksDB
Client -s -P workloads/workloada -p rocksdb.dir=/mnt/test -p maxexecutiontime=10
800 -t
Command line: -db site.ycsb.db.rocksdb.RocksDBClient -s -P workloads/workloada -
p rocksdb.dir=/mnt/test -p maxexecutiontime=10800 -t
YCSB Client 0.18.0-SNAPSHOT

Loading workload...
Starting test.
SLF4J: Failed to load class "org.slf4j.impl.StaticLoggerBinder".
SLF4J: Defaulting to no-operation (NOP) logger implementation
SLF4J: See http://www.slf4j.org/codes.html#StaticLoggerBinder for further detail
s.
```

[그림 4-30] workloada를 run하는 옵션 및 과정

**빨간색 박스**는 rocksdb 데이터베이스를 사용하여 run하겠다는 의미이다. -s 옵션은 10초마다 operation이 얼마나 일어나고 있는지를 log로 보여준다.

**초록색 박스**는 workloads 폴더에 있는 workloada라는 workload를 사용하여 run하고, 그 결과를 runttest1.txt라는 파일을 만들어서 저장하라는 의미이다. -P 옵션은 파라미터들을 넣는데 사용된다.

**파란색 박스**는 mount한 SSD의 경로를 지정해준 것이다.

**보라색 박스**는 최대 실행시간을 10800초로 제한하라는 옵션이다.

실험이 완료되면 load와 run 과정에서 파일이 생성된다. 이는 YCSB 디렉토리에서 ls 명령어를 통해 확인할 수 있다.

아래 [그림 4-31]을 보면 loadtest1.txt, runtest1.txt 파일이 생성된 것을 확인할 수 있다. vi 에디터를 통해 run 파일을 열어 실험 결과를 확인하였다. [그림 4-32]의 RunTime은 전체 실행 시간을 뜻한다. Throughput은 평균 처리량을 뜻하고, latency는 지연 시간을 뜻한다. 95, 99thPercentileLatency(us)는 95번, 99번째 Percentile 지연시간을 뜻한다.

```
eunjin@eunjin-PC:~/YCSB$ ls
CONTRIBUTING.md  checkstyle.xml  googlebigtable  memcached       solr
LICENSE.txt       cloudspanner    googledatastore mongodb          solr6
NOTICE.txt        core            griddb          nosqlldb        solr7
README.md         couchbase       hbase1          orientdb         tablestore
accumulo1.9       couchbase2      hbase2          pom.xml         tarantool
aerospike         crail           hypertable      postgresql      target
arangodb          distribution    ignite          rados           voldemort
asynbase          doc             infinispans     redis           voltdb
azurecosmos       dynamodb       jdbc            rest            workloads
azuretablestorage elasticsearch   kudu            riak
bin               elasticsearch5 loadtest1.txt   rocksdb
binding-parent    foundationdb    maprdb          runtest1.txt
```

[그림 4-31] loadtest.txt 파일과 runtest.txt 파일 생성

```
[OVERALL], RunTime(ms), 10800218
[OVERALL], Throughput(ops/sec), 8412.832037279248
[TOTAL_GCS_PS_Scavenge], Count, 5407
[TOTAL_GC_TIME_PS_Scavenge], Time(ms), 14192
[TOTAL_GC_TIME_%_PS_Scavenge], Time(%), 0.13140475497809395
[TOTAL_GCS_PS_MarkSweep], Count, 4
[TOTAL_GC_TIME_PS_MarkSweep], Time(ms), 41
[TOTAL_GC_TIME_%_PS_MarkSweep], Time(%), 3.7962196688992757E-4
[TOTAL_GCs], Count, 5411
[TOTAL_GC_TIME], Time(ms), 14233
[TOTAL_GC_TIME_%], Time(%), 0.13178437694498388
[READ], Operations, 48963530
[READ], AverageLatency(us), 108.0747947502968
[READ], MinLatency(us), 2
[READ], MaxLatency(us), 2865151
[READ], 95thPercentileLatency(us), 236
[READ], 99thPercentileLatency(us), 271
[READ], Return=OK, 48963530
[READ], Return=NOT_FOUND, 41896890
```

[그림 4-32] runtest.txt 결과 파일

다음은 FIO-Cgroups의 결과 확인에 대한 내용이다. 앞의 4.1.3.에서 언급한 FIO 실험 방식과 옵션을 동일하게 사용하였다. 아래 [그림 4-33], [그림 4-34], [그림 4-35], [그림 4-36]은 test1-test4 실험 결과를 요약한 그림이다.

```
test1: (groupid=0, jobs=1): err= 0: pid=8830: Sat Apr 11 04:20:20 2020
read: IOPS=1748, BW=6993KiB/s (7161kB/s)(72.0GiB/10800017msec)
slat (nsec): min=912, max=896641, avg=2573.35, stdev=2134.58
```

[그림 4-33] FIO-Cgroups test1 결과

```
test2: (groupid=0, jobs=1): err= 0: pid=8831: Sat Apr 11 04:20:20 2020
read: IOPS=3545, BW=13.9MiB/s (14.5MB/s)(146GiB/10800026msec)
slat (nsec): min=902, max=1120.2k, avg=2257.62, stdev=1618.65
```

[그림 4-34] FIO-Cgroups test2 결과

```
test3: (groupid=0, jobs=1): err= 0: pid=8832: Sat Apr 11 04:20:20 2020
read: IOPS=7124, BW=27.8MiB/s (29.2MB/s)(294GiB/10800009msec)
slat (nsec): min=902, max=533140, avg=1992.48, stdev=1283.87
```

[그림 4-35] FIO-Cgroups test3 결과

```
test4: (groupid=0, jobs=1): err= 0: pid=8833: Sat Apr 11 04:20:20 2020
read: IOPS=14.2k, BW=55.6MiB/s (58.3MB/s)(586GiB/10800001msec)
slat (nsec): min=901, max=1528.2k, avg=1883.04, stdev=1193.03
```

[그림 4-36] FIO-Cgroups test4 결과

실험이 끝나면 rm 명령어를 사용하여 YCSB 디렉토리의 load, run 파일과 SSD 마운트 디렉토리에 있는 파일들을 지워준다.

```
eunjin@eunjin-PC:~/YCSB$ sudo rm load*
[sudo] eunjin의 암호:
eunjin@eunjin-PC:~/YCSB$ sudo rm run*
eunjin@eunjin-PC:~/YCSB$ ls
CONTRIBUTING.md  cassandra      foundationdb    maprdb          rocksdb
LICENSE.txt       checkstyle.xml geode           maprjsondb      s3
NOTICE.txt        cloudspanner   googlebigtable  memcached       solr
README.md         core           googledatastore mongodb          solr6
accumulo1.9       couchbase      griddb          nosqlldb        solr7
aerospike         couchbase2     hbase1          orientdb         tablestore
arangodb          crail          hbase2          pom.xml         tarantool
asynchbase        distribution   hypertable      postgrenoql     target
azurecosmos       doc            ignite          rados           voldemort
azuretablestorage dynamodb       infinispans     redis           voltdb
bin               elasticsearch jdbc            rest            workloads
binding-parent    elasticsearch5 kudu            riak
```

[그림 4-37] load, run 파일 삭제

```
eunjin@eunjin-PC:~/YCSB$ cd ../mnt/test
eunjin@eunjin-PC:/mnt/test$ ls
000011.sst 000017.log LOG OPTIONS-000008 test4.0.0
000014.log CURRENT LOG.old.1586842624705915 test1.0.0
000015.sst IDENTITY MANIFEST-000006 test2.0.0
000016.log LOCK OPTIONS-000005 test3.0.0
eunjin@eunjin-PC:/mnt/test$ sudo rm *
eunjin@eunjin-PC:/mnt/test$ ls
eunjin@eunjin-PC:/mnt/test$
```

[그림 4-38] SSD 마운트 디렉토리 안에 있는 파일 삭제

#### 4.2.3.2. Write

26페이지의 4.2.3.1.과 실험 방식은 동일하다. 다만, write를 사용함으로써 workloada 옵션이 조금 달라진다. 다음은 write의 옵션이다. recordcount=100000000, operationcount= 100000000, updateproportion=1로 변경하고 저장 후 종료한다. 이후 모든 실험 과정은 Read와 같다.

```
recordcount=100000000
operationcount=100000000
workload=site.ycsb.workloads.CoreWorkload

readallfields=true

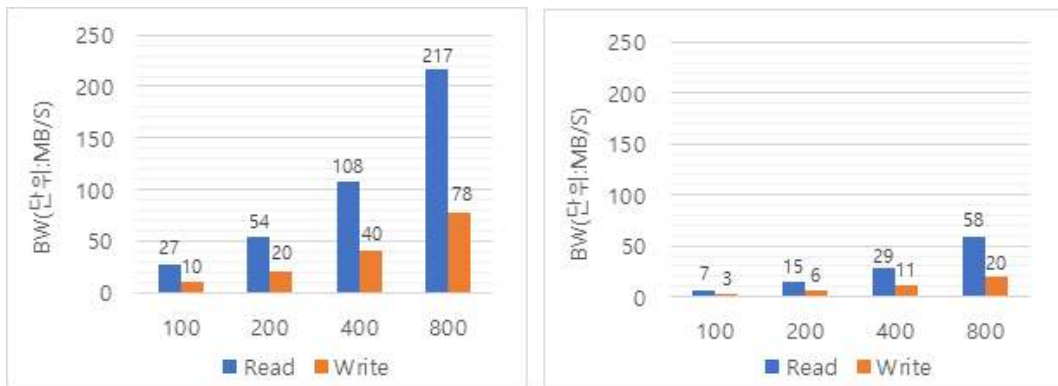
readproportion=0
updateproportion=1
scanproportion=0
insertproportion=0
```

[그림 4-39] Write를 하기 위한 workloada 옵션 변경

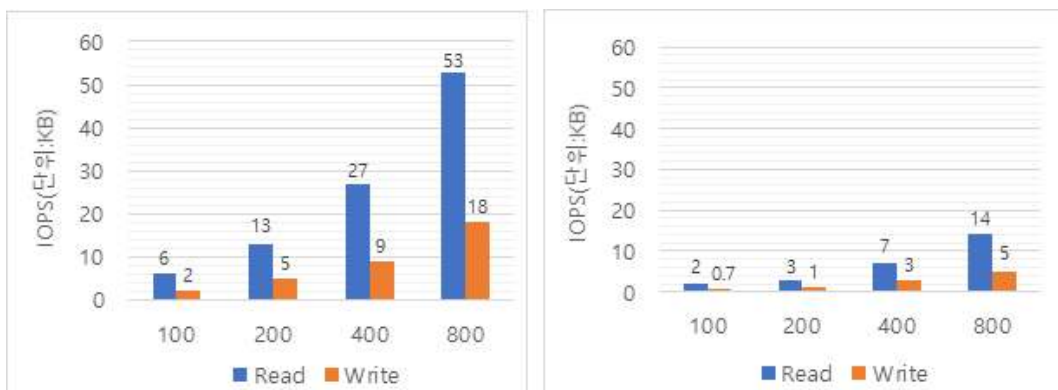


#### 4.2.4. 실험 결과

아래 [그림 4-40]과 [그림 4-41]은 4개의 task 그룹에 가중치 100, 200, 400, 800을 설정한 뒤 FIO-Cgroups와 YCSB-Rocks DB를 동시에 실행한 BW와 IOPS의 결과그래프이다. FIO와 YCSB를 동시에 실행함으로써 디스크 I/O의 성능 차이가 발생했지만, Cgroups의 가중치가 100일 때, 200일 때, 400일 때, 800일 때 모두 시스템 자원에 대한 분배가 일정하게 이루어진다는 사실을 확인하였다.



[그림 4-40] FIO-Cgroup, YCSB-RocksDB & FIO-Cgroups 동시 측정(BW)



[그림 4-41] FIO-Cgroup, YCSB-RocksDB & FIO-Cgroup 동시 측정(IOPS)

### 4.3. Cgroups를 활용한 I/O wieght 개수에 따른 성능 평가

#### 4.3.1. 디스크 추가

\* 4.2.1.과 디스크 마운트 위치(/mnt/test)를 제외한 모든 방식과 옵션을 동일하게 진행하였다.

#### 4.3.2. FIO 벤치마크 툴을 이용해 Cgroups 실행

\* 4.1.2.와 동일한 방식으로 실험을 진행하였지만, task 그룹의 개수 및 가중치 할당은 다르게 지정하였습니다. 본 절에서는 Cgroups 실행 과정은 함축 시켰으므로 실행 과정에 대한 자세한 내용은 4.1.2. 절을 참고하면 된다.

##### 4.3.2.1. Cgroups Blkio I/O 컨트롤러 사용

Cgroups 및 blkio 서브시스템을 사용하기 위해서는 먼저 블록 I/O 컨트롤러 활성화, 블록 레이어 조절 활성화, CFQ 그룹 스케줄링 사용을 활성화시킨 후 커널이 Cgroups를 지원하는지 확인하고, 스케줄러가 CFQ로 되어 있는지 확인한다.

```
eunjin@eunjin-PC:~$ CONFIG_BLK_CGROUP=Y
eunjin@eunjin-PC:~$ CONFIG_BLK_DEV_THROTTLING=Y
eunjin@eunjin-PC:~$ CONFIG_CFQ_GROUP_IOSCHED=Y
```

[그림 4-42] blkio 서브시스템을 사용하기 위한 Cgroups 활성화

```
eunjin@eunjin-PC:~$ cat /proc/cgroups
#subsys_name hierarchy num_cgroups enabled
cpuset 11 1 1
cpu 2 1 1
cpuacct 2 1 1
blkio 9 1 1
```

[그림 4-43] Cgroups 지원 확인

```
eunjin@eunjin-PC:~$ cat /sys/block/sda/queue/scheduler
noop deadline [cfq]
```

[그림 4-44] 스케줄러 확인

#### 4.3.2.2. Cgroups 마운트 및 Blkio I/O 컨트롤러 사용

Cgroups의 blkio 서브시스템을 사용하기 위해 Cgroups 루트 디렉토리와 blkio 하위 시스템을 마운트 해준 후 blkio 컨트롤러에서 mkdir 명령어를 통해 128개의 task 그룹을 생성해준다. 아래의 [그림 4-46]과 동일한 방법으로 생성하면 된다.

```
eunjin@eunjin-PC:~$ sudo mount -t tmpfs cgroup_root /sys/fs/cgroup
[sudo] eunjin의 암호:
eunjin@eunjin-PC:~$ sudo mkdir /sys/fs/cgroup/blkio
eunjin@eunjin-PC:~$ sudo mount -t cgroup -o blkio none /sys/fs/cgroup/blkio
```

[그림 4-45] Cgroups 루트 디렉토리와 blkio 하위 시스템 마운트

```
eunjin@eunjin-PC:~$ sudo mkdir /sys/fs/cgroup/blkio/task1
eunjin@eunjin-PC:~$ sudo mkdir /sys/fs/cgroup/blkio/task2
eunjin@eunjin-PC:~$ sudo mkdir /sys/fs/cgroup/blkio/task3
eunjin@eunjin-PC:~$ sudo mkdir /sys/fs/cgroup/blkio/task4
eunjin@eunjin-PC:~$ sudo mkdir /sys/fs/cgroup/blkio/task5
eunjin@eunjin-PC:~$ sudo mkdir /sys/fs/cgroup/blkio/task6
eunjin@eunjin-PC:~$ sudo mkdir /sys/fs/cgroup/blkio/task7
eunjin@eunjin-PC:~$ sudo mkdir /sys/fs/cgroup/blkio/task8
eunjin@eunjin-PC:~$ sudo mkdir /sys/fs/cgroup/blkio/task9
eunjin@eunjin-PC:~$ sudo mkdir /sys/fs/cgroup/blkio/task10
eunjin@eunjin-PC:~$ sudo mkdir /sys/fs/cgroup/blkio/task11
eunjin@eunjin-PC:~$ sudo mkdir /sys/fs/cgroup/blkio/task12
eunjin@eunjin-PC:~$ sudo mkdir /sys/fs/cgroup/blkio/task13
eunjin@eunjin-PC:~$ sudo mkdir /sys/fs/cgroup/blkio/task14
eunjin@eunjin-PC:~$ sudo mkdir /sys/fs/cgroup/blkio/task15
eunjin@eunjin-PC:~$ sudo mkdir /sys/fs/cgroup/blkio/task16
```

[그림 4-46] task1, 2, 3, 4 ... 128 이름을 가진 그룹 생성



#### 4.3.2.3. Cgroups에 프로세스 할당

blkio 컨트롤러에서 생성한 128개의 task 그룹에 각각의 TID를 할당 해주어야 한다. 먼저 `cat /sys/fs/cgroups/blkio/tasks` 명령어를 통해 프로세스 목록을 확인한 후 `echo [TID] > /sys/fs/cgroup/blkio/task1/tasks` 명령어를 통해 임의의 TID를 4개의 task 그룹에 각각 할당해준다. 마지막으로 `cat` 명령어를 통해 128개의 task 그룹에 지정한 프로세스가 할당 되었는지 확인한다. 아래 [그림 4-48]를 보면 임의로 지정한 TID 7개가 정상적으로 할당 된 것을 확인할 수 있다.

```
root@eunjin-PC:/home/eunjin# echo 1127 > /sys/fs/cgroup/blkio/task1/tasks
root@eunjin-PC:/home/eunjin# echo 1128 > /sys/fs/cgroup/blkio/task2/tasks
root@eunjin-PC:/home/eunjin# echo 1134 > /sys/fs/cgroup/blkio/task3/tasks
root@eunjin-PC:/home/eunjin# echo 1135 > /sys/fs/cgroup/blkio/task4/tasks
root@eunjin-PC:/home/eunjin# echo 1136 > /sys/fs/cgroup/blkio/task5/tasks
root@eunjin-PC:/home/eunjin# echo 1138 > /sys/fs/cgroup/blkio/task6/tasks
root@eunjin-PC:/home/eunjin# echo 1139 > /sys/fs/cgroup/blkio/task7/tasks
root@eunjin-PC:/home/eunjin# echo 1162 > /sys/fs/cgroup/blkio/task8/tasks
root@eunjin-PC:/home/eunjin# echo 1164 > /sys/fs/cgroup/blkio/task9/tasks
root@eunjin-PC:/home/eunjin# echo 1165 > /sys/fs/cgroup/blkio/task10/tasks
root@eunjin-PC:/home/eunjin# echo 1166 > /sys/fs/cgroup/blkio/task11/tasks
root@eunjin-PC:/home/eunjin# echo 1167 > /sys/fs/cgroup/blkio/task12/tasks
root@eunjin-PC:/home/eunjin# echo 1168 > /sys/fs/cgroup/blkio/task13/tasks
root@eunjin-PC:/home/eunjin# echo 1169 > /sys/fs/cgroup/blkio/task14/tasks
```

[그림 4-47] task1, 2, 3, 4 ... 128 그룹에 임의의 TID 할당

```
root@eunjin-PC:/home/eunjin# cat /sys/fs/cgroup/blkio/task1/tasks
1127
root@eunjin-PC:/home/eunjin# cat /sys/fs/cgroup/blkio/task2/tasks
1128
root@eunjin-PC:/home/eunjin# cat /sys/fs/cgroup/blkio/task3/tasks
1134
root@eunjin-PC:/home/eunjin# cat /sys/fs/cgroup/blkio/task4/tasks
1135
root@eunjin-PC:/home/eunjin# cat /sys/fs/cgroup/blkio/task5/tasks
1136
root@eunjin-PC:/home/eunjin# cat /sys/fs/cgroup/blkio/task6/tasks
1138
root@eunjin-PC:/home/eunjin# cat /sys/fs/cgroup/blkio/task7/tasks
1139
root@eunjin-PC:/home/eunjin# cat /sys/fs/cgroup/blkio/task8/tasks
```

[그림 4-48] 임의로 지정한 TID 할당 확인

#### 4.3.2.4. weight 비율 할당

128개의 task 그룹에 blkio.weight 변수를 통해 가중치(weight) 비율을 설정하고 cat 명령어를 통해 확인한다. 이때, 비율은 차례대로 100, 200, 400, 800을 동일하게 설정한다. 즉, task1, 5, 9, 13 ... 125 그룹에 100, task2, 6, 10, 14 ... 126 그룹에 200, task3, 7, 11, 15 ... 127 그룹에 400, task4, 8, 12, 16 ... 128 그룹에 800의 가중치를 설정한다. 이후 lscgroup 명령어를 통해 128개의 task 그룹과 blkio 서브시스템이 마운트 되었는지 확인한다.

```
root@eunjin-PC:/home/eunjin# echo 100 > /sys/fs/cgroup/blkio/task1/blkio.weight
root@eunjin-PC:/home/eunjin# echo 200 > /sys/fs/cgroup/blkio/task2/blkio.weight
root@eunjin-PC:/home/eunjin# echo 400 > /sys/fs/cgroup/blkio/task3/blkio.weight
root@eunjin-PC:/home/eunjin# echo 800 > /sys/fs/cgroup/blkio/task4/blkio.weight
root@eunjin-PC:/home/eunjin# echo 100 > /sys/fs/cgroup/blkio/task5/blkio.weight
root@eunjin-PC:/home/eunjin# echo 200 > /sys/fs/cgroup/blkio/task6/blkio.weight
root@eunjin-PC:/home/eunjin# echo 400 > /sys/fs/cgroup/blkio/task7/blkio.weight
root@eunjin-PC:/home/eunjin# echo 800 > /sys/fs/cgroup/blkio/task8/blkio.weight
root@eunjin-PC:/home/eunjin# echo 100 > /sys/fs/cgroup/blkio/task9/blkio.weight
root@eunjin-PC:/home/eunjin# echo 200 > /sys/fs/cgroup/blkio/task10/blkio.weight
```

[그림 4-49] task1, 2, 3, 4 ... 128 그룹에 weight 설정

```
root@eunjin-PC:/home/eunjin# cat /sys/fs/cgroup/blkio/task1/blkio.weight
100
root@eunjin-PC:/home/eunjin# cat /sys/fs/cgroup/blkio/task2/blkio.weight
200
root@eunjin-PC:/home/eunjin# cat /sys/fs/cgroup/blkio/task3/blkio.weight
400
root@eunjin-PC:/home/eunjin# cat /sys/fs/cgroup/blkio/task4/blkio.weight
800
root@eunjin-PC:/home/eunjin# cat /sys/fs/cgroup/blkio/task5/blkio.weight
100
root@eunjin-PC:/home/eunjin# cat /sys/fs/cgroup/blkio/task6/blkio.weight
200
```

[그림 4-50] task1, 2, 3, 4 ... 128 그룹에 wieght 할당 확인

```
root@eunjin-PC:/home/eunjin# lscgroup
blkio:/
blkio:/task3
blkio:/task16
blkio:/task1
blkio:/task14
```

[그림 4-51] task1, 2, 3, 4 ... 128 그룹 마운트 확인

### 4.3.3. FIO-Cgroups 성능 측정

#### 4.3.3.1. FIO Job File 생성

\* 4.1.3.1. 과 동일한 방식으로 실험을 진행하였다.

#### 4.3.3.2. FIO Job File 옵션

\* 4.1.3.2. 와 동일한 FIO 옵션을 사용하였지만, task 그룹의 개수는 다르게 지정하였다. 본 절에서는 FIO 명령어 설명은 소개하지 않으므로 명령어에 대한 자세한 내용은 4.1.3.2. 절을 참고하면 된다.

아래 39페이지의 [표 4-2], [표 4-3]은 각각 read와 Write의 mix 옵션인 4, 8개의 task 그룹으로 Job File을 생성한 표이다. 16, 32, 64, 128개의 task 그룹도 마찬가지로 위의 [표 4-2], [표 4-3]과 같은 방법으로 옵션을 지정하여 FIO Job File을 생성하면 된다. 즉, 16개의 task 그룹을 묶어 하나의 Job File을 만들고, 32개의 task 그룹을 묶어 하나의 Job File을 만들고, 64개의 task 그룹을 묶어 하나의 Job File을 만들고 128개의 task 그룹을 묶어 하나의 Job File을 만든다.

[표 4-2] 4개의 task 그룹 FIO Job File 옵션

<pre>[global] bs=4k ioengine=libaio iodepth=32 direct=1 rw=rw time_based runtime=10800 cgroup_nodelete=1  [test1] directory=/mnt/test size=1G cgroup=task1</pre>	<pre>[test2] directory=/mnt/test size=1G cgroup=task2  [test3] directory=/mnt/test size=1G cgroup=task3  [test4] directory=/mnt/test size=1G cgroup=task4</pre>
--	---

[표 4-3] 8개의 task 그룹 FIO Job File 옵션

<pre>[global] bs=4k ioengine=libaio iodepth=32 direct=1 rw=rw time_based runtime=10800 cgroup_nodelete=1  [test1] directory=/mnt/test size=1G cgroup=task1  [test2] directory=/mnt/test size=1G cgroup=task2  [test3] directory=/mnt/test size=1G cgroup=task3</pre>	<pre>[test4] directory=/mnt/test size=1G cgroup=task4  [test5] directory=/mnt/test size=1G cgroup=task5  [test6] directory=/mnt/test size=1G cgroup=task6  [test7] directory=/mnt/test size=1G cgroup=task7  [test8] directory=/mnt/test size=1G cgroup=task8</pre>
--	---

#### 4.3.3.3. FIO 실행

\* 4.1.3.3.과 동일한 방식으로 실험을 진행하였다. 본 실험에서는 IOPS만 기록 하여 측정하였다.

4.1.3.2.에서 task 그룹 개수에 따라 생성한 6개의 FIO Job File에 대한 모든 실험을 진행하였다. 즉, 4개의 task 그룹을 묶어 만든 Job File 한 번 진행, 8개의 task 그룹을 묶어 만든 Job File 한 번 진행, 16개의 task 그룹을 묶어 만든 Job File 한 번 진행, 32개의 task 그룹을 묶어 만든 Job File 한 번 진행, 64개의 task 그룹을 묶어 만든 Job File 한 번 진행, 128개의 task 그룹을 묶어 만든 Job File 한 번 진행, 총 6개의 Job File을 실행하였다. 아래의 [그림 4-52], [그림 4-53], [그림 4-54]는 각각 4개, 8개, 16개의 task 그룹을 묶어 만든 Job File을 실행하는 화면이다.

```
eunjin@eunjin-PC:~/바탕화면$ sudo fio test.fio
test1: (g=0): rw=rw, bs=(R) 4096B-4096B, (W) 4096B-4096B, (T) 4096B-4096B, ioengine=libaio, iodepth=32
test2: (g=0): rw=rw, bs=(R) 4096B-4096B, (W) 4096B-4096B, (T) 4096B-4096B, ioengine=libaio, iodepth=32
test3: (g=0): rw=rw, bs=(R) 4096B-4096B, (W) 4096B-4096B, (T) 4096B-4096B, ioengine=libaio, iodepth=32
test4: (g=0): rw=rw, bs=(R) 4096B-4096B, (W) 4096B-4096B, (T) 4096B-4096B, ioengine=libaio, iodepth=32
fio-3.1
Starting 4 processes
test1: Laying out IO file (1 file / 1024MiB)
test2: Laying out IO file (1 file / 1024MiB)
test3: Laying out IO file (1 file / 1024MiB)
```

[그림 4-52] 4개의 task 그룹을 묶어 만든 Job File 실행

```
eunjin@eunjin-PC:~/바탕화면$ sudo fio test.fio
test1: (g=0): rw=rw, bs=(R) 4096B-4096B, (W) 4096B-4096B, (T) 4096B-4096B, ioengine=libaio, iodepth=32
test2: (g=0): rw=rw, bs=(R) 4096B-4096B, (W) 4096B-4096B, (T) 4096B-4096B, ioengine=libaio, iodepth=32
test3: (g=0): rw=rw, bs=(R) 4096B-4096B, (W) 4096B-4096B, (T) 4096B-4096B, ioengine=libaio, iodepth=32
test4: (g=0): rw=rw, bs=(R) 4096B-4096B, (W) 4096B-4096B, (T) 4096B-4096B, ioengine=libaio, iodepth=32
test5: (g=0): rw=rw, bs=(R) 4096B-4096B, (W) 4096B-4096B, (T) 4096B-4096B, ioengine=libaio, iodepth=32
test6: (g=0): rw=rw, bs=(R) 4096B-4096B, (W) 4096B-4096B, (T) 4096B-4096B, ioengine=libaio, iodepth=32
test7: (g=0): rw=rw, bs=(R) 4096B-4096B, (W) 4096B-4096B, (T) 4096B-4096B, ioengine=libaio, iodepth=32
test8: (g=0): rw=rw, bs=(R) 4096B-4096B, (W) 4096B-4096B, (T) 4096B-4096B, ioengine=libaio, iodepth=32
fio-3.1
Starting 8 processes
test1: Laying out IO file (1 file / 1024MiB)
test2: Laying out IO file (1 file / 1024MiB)
test3: Laying out IO file (1 file / 1024MiB)
test4: Laying out IO file (1 file / 1024MiB)
test5: Laying out IO file (1 file / 1024MiB)
test6: Laying out IO file (1 file / 1024MiB)
test7: Laying out IO file (1 file / 1024MiB)
test8: Laying out IO file (1 file / 1024MiB)
```

[그림 4-53] 8개의 task 그룹을 묶어 만든 Job File 실행



```

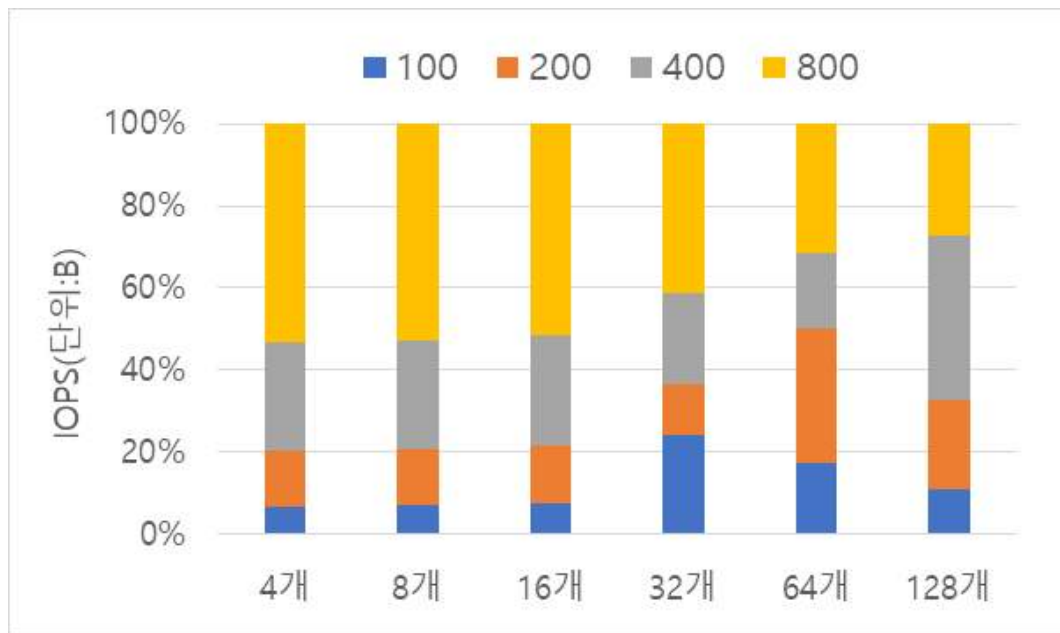
eunjin@eunjin-PC:~/바탕화면$ sudo fio test.fio
test1: (g=0): rw=RW, bs=(R) 4096B-4096B, (W) 4096B-4096B, (T) 4096B-4096B, ioengine=libaio, iodepth=32
test2: (g=0): rw=RW, bs=(R) 4096B-4096B, (W) 4096B-4096B, (T) 4096B-4096B, ioengine=libaio, iodepth=32
test3: (g=0): rw=RW, bs=(R) 4096B-4096B, (W) 4096B-4096B, (T) 4096B-4096B, ioengine=libaio, iodepth=32
test4: (g=0): rw=RW, bs=(R) 4096B-4096B, (W) 4096B-4096B, (T) 4096B-4096B, ioengine=libaio, iodepth=32
test5: (g=0): rw=RW, bs=(R) 4096B-4096B, (W) 4096B-4096B, (T) 4096B-4096B, ioengine=libaio, iodepth=32
test6: (g=0): rw=RW, bs=(R) 4096B-4096B, (W) 4096B-4096B, (T) 4096B-4096B, ioengine=libaio, iodepth=32
test7: (g=0): rw=RW, bs=(R) 4096B-4096B, (W) 4096B-4096B, (T) 4096B-4096B, ioengine=libaio, iodepth=32
test8: (g=0): rw=RW, bs=(R) 4096B-4096B, (W) 4096B-4096B, (T) 4096B-4096B, ioengine=libaio, iodepth=32
test9: (g=0): rw=RW, bs=(R) 4096B-4096B, (W) 4096B-4096B, (T) 4096B-4096B, ioengine=libaio, iodepth=32
test10: (g=0): rw=RW, bs=(R) 4096B-4096B, (W) 4096B-4096B, (T) 4096B-4096B, ioengine=libaio, iodepth=32
test11: (g=0): rw=RW, bs=(R) 4096B-4096B, (W) 4096B-4096B, (T) 4096B-4096B, ioengine=libaio, iodepth=32
test12: (g=0): rw=RW, bs=(R) 4096B-4096B, (W) 4096B-4096B, (T) 4096B-4096B, ioengine=libaio, iodepth=32
test13: (g=0): rw=RW, bs=(R) 4096B-4096B, (W) 4096B-4096B, (T) 4096B-4096B, ioengine=libaio, iodepth=32
test14: (g=0): rw=RW, bs=(R) 4096B-4096B, (W) 4096B-4096B, (T) 4096B-4096B, ioengine=libaio, iodepth=32
test15: (g=0): rw=RW, bs=(R) 4096B-4096B, (W) 4096B-4096B, (T) 4096B-4096B, ioengine=libaio, iodepth=32
test16: (g=0): rw=RW, bs=(R) 4096B-4096B, (W) 4096B-4096B, (T) 4096B-4096B, ioengine=libaio, iodepth=32
fio-3.1
Starting 16 processes
test1: Laying out IO file (1 file / 1024MiB)
test2: Laying out IO file (1 file / 1024MiB)
test3: Laying out IO file (1 file / 1024MiB)
test4: Laying out IO file (1 file / 1024MiB)
test5: Laying out IO file (1 file / 1024MiB)
test6: Laying out IO file (1 file / 1024MiB)
test7: Laying out IO file (1 file / 1024MiB)
test8: Laying out IO file (1 file / 1024MiB)
test9: Laying out IO file (1 file / 1024MiB)
test10: Laying out IO file (1 file / 1024MiB)
test11: Laying out IO file (1 file / 3072MiB)
test12: Laying out IO file (1 file / 1024MiB)
test13: Laying out IO file (1 file / 1024MiB)
test14: Laying out IO file (1 file / 1024MiB)
test15: Laying out IO file (1 file / 1024MiB)
test16: Laying out IO file (1 file / 1024MiB)

```

[그림 4-54] 16개의 task 그룹을 묶어 만든 Job File 실행

#### 4.3.4. 실험 결과

아래 [그림 4-55]는 task 그룹의 개수가 증가하는 경우, Cgroup의 자원이 제대로 분배되고 있는지에 대한 실험을 진행한 결과 그래프이다. task 그룹이 4개, 8개, 16개인 경우 I/O 가중치의 비율이 Cgroups를 통해 지정한 100, 200, 400, 800의 비율을 잘 반영하고 있다. 그러나 32개의 태스크 그룹부터는 예상과 다르게 각 task 그룹이 지정한 가중치 비율이 잘 반영되지 않는 것을 확인하였다. 이는 FIO의 프로세스가 증가함과 동시에 스토리지의 I/O의 양이 증가하고 그 결과 Cgroups의 가중치에 영향을 주는 것으로 보인다.



[그림 4-55] FIO-Cgroups I/O weight 개수에 따른 결과 그래프 (IOPS)

## 5. 결론 및 향후 실험

본 논문에서는 리눅스의 Cgroups를 활용한 성능 평가를 수행하였다. 4.1. 절과 4.2. 절에서 진행한 실험에서는 가중치가 100일 때, 200일 때, 400일 때, 800일 때 모두 I/O 가중치의 비율이 Cgroups를 통해 지정한 100, 200, 400, 800의 비율을 잘 반영하고 있는 것을 확인했지만, 4.3. 절에서 진행한 실험에서는 task 그룹이 4개, 8개, 16개인 경우 I/O 가중치의 비율이 Cgroups를 통해 지정한 100, 200, 400, 800의 비율을 잘 반영하고 있는 것을 확인했지만, 32개의 태스크 그룹부터는 예상과 다르게 각 task 그룹이 지정한 가중치 비율이 잘 반영되지 않는 것을 확인할 수 있었다. 이는 FIO의 프로세스가 증가함과 동시에 스토리지의 I/O의 양이 증가하고 그 결과 Cgroups의 가중치에 영향을 주는 것으로 보인다.

이에, 추후 연구에서는 4.3. 실험에서 자원 분배가 지켜지지 않은 이유에 대해서 분석하고, 고성능 NVMe SSD를 활용하여 추가적인 실험을 진행하고자 한다.



## 참 고 문 헌

- [1] introduction to Control Groups[Online]. Available: [https://access.redhat.com/documentation/ko-kr/red\\_hat\\_enterprise\\_linux/6/html/resource\\_management\\_guide/ch01](https://access.redhat.com/documentation/ko-kr/red_hat_enterprise_linux/6/html/resource_management_guide/ch01) (downloaded 2020, Apr. 27)
- [2] H. Kamezawa, "Cgroup and Memory Resource Controller", Japan Linux Symposium, pp. 9, 2008.
- [3] 이성우, 박종규, 엄영익, "리눅스 I/O 최적화기법이 Cgroups 를 활용한 I/O weight 에 미치는 영향 분석", 한국정보과학회 학술발표논문집, pp. 2225, 2018.
- [4] 김명선, 임진택, 박대동, "안드로이드 단말의 태스크 그룹에 따른 CPU 점유율 분석", 한국컴퓨터정보학회 학술발표논문집, pp. 9, 2013.
- [5] K. Ryoji, M. Hideaki, I. Yohei, S. Akira, "Effectiveness of Priority Control Method by Using Cgroups in KVM", Proc. ITC-CSCC International Technical Conference on Circuits Systems, pp. 2, 2016.
- [6] 손수호, 박기태, 권민재, 안성용, "멀티-큐 SSD 를 위한 경량 가중치 기반 I/O 스케줄러", 한국정보과학회 학술발표논문집, pp. 1761, 2019.
- [7] 오권제, 박종규, 엄영익, "다중 레벨 Cgroup 에서 BFQ 스케줄러의 I/O 성능 비율 저하 현상 분석", 한국정보과학회 학술발표논문집, pp. 1078, 2019
- [8] 안미진, 오기환, 강운학, 이상원, "RocksDB의 동기화 성능 비교", 한국정보과학회 학술발표논문집, pp. 1516, 2014

- [9] Building and open-sourcing Rocksdb[Online]. Available: <https://ko-kr.facebook.com/notes/facebook-engineering/under-the-hood-building-and-open-sourcing-rocksdb/10151822347683920/> (downloaded 2020, May, 24)
- [10] 박연수, 오기환, 이종백, 강운학, 이상원, “동기화 옵션에 따른 RocksDB 성능 평가”, 한국정보과학회 학술발표논문집, pp. 1731, 2014
- [11] fio-Flexible I/O tester rev.3.19[Online]. Available: [https://fio.readthedocs.io/en/latest/fio\\_doc.html](https://fio.readthedocs.io/en/latest/fio_doc.html) (downloaded 2020, May, 24)
- [12] YCSB[Online]. Available: <https://github.com/brianfrankcooper/YCSB> (downloaded 2020, May, 24)
- [13] 박현찬, 유혁, “CFQ 스케줄러의 튜닝 변수 설정에 따른 SSD 성능 분석”, 한국정보과학회 학술발표논문집, pp. 424, 2011.