2014211491 전재형

1lantine1@naver.com

요약

[매력적인 요약문으로 독자의 시선을 끌어 보세요. 일반적으로 요약은 문서의 내용을 간략하게 정리한 것입니다.   
내용을 추가하려면 여기를 클릭하고 입력하세요.]

XFS와 CASSANDRA db를 사용한 성능비교

[문서 부제]

내용

[기본 실험 환경 구성 2](#_Toc37851145)

[SSD 연결 확인 및 파티션 설정 2](#_Toc37851146)

[File System 포맷 및 Mount 하기 3](#_Toc37851147)

[Fio 벤치마크 설치 4](#_Toc37851148)

[XFS 파일 시스템 설치 4](#_Toc37851149)

[실험 5](#_Toc37851150)

[Fio 벤치마크를 이용한 ext4파일시스템에서 read, write, randread, randwrite에 대한 값의 차이보기 5](#_Toc37851151)

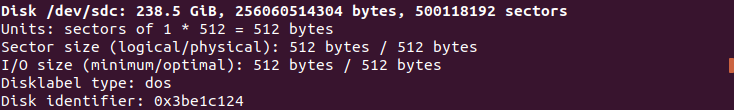
[실험결과 7](#_Toc37851152)

[Ext4 파일시스템에서 read, write, randread, randwrite에 대한 값의 차이 7](#_Toc37851153)

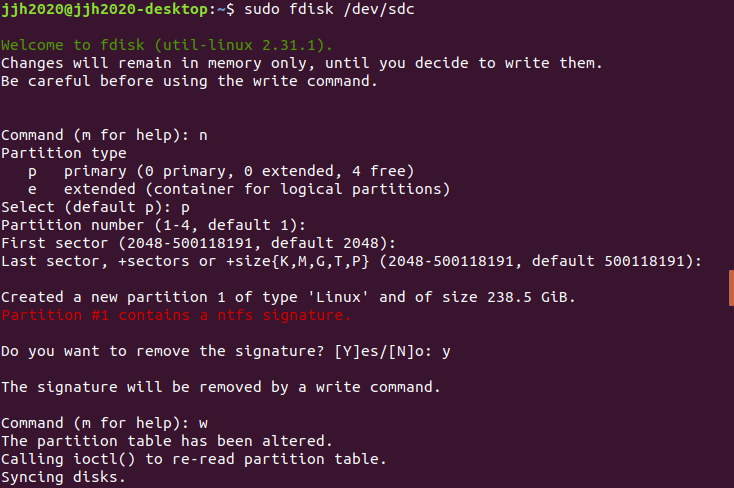
# 기본 실험 환경 구성

## SSD 연결 확인 및 파티션 설정

sudo fdisk –l로 연결된 ssd 확인



sudo fdisk /dev/sdc로 파티션 설정

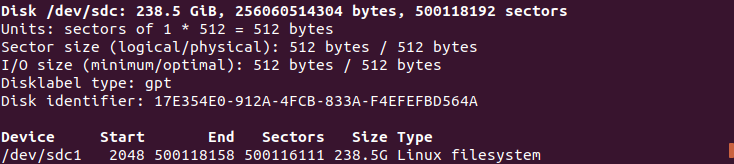


n을 입력하여 파티션 생성, p를 입력하여 기본 파티션 타입 설정

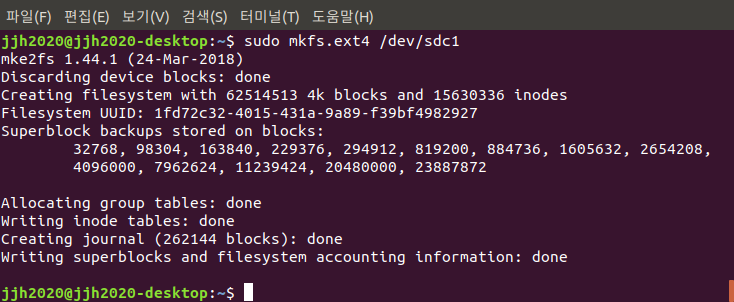
Enter 입력하여 default 1로 파티션 생성, 기존의 파일시스템을 지우겠냐고 물어보면 y,

w로 저장

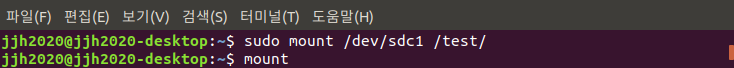
## File System 포맷 및 Mount 하기



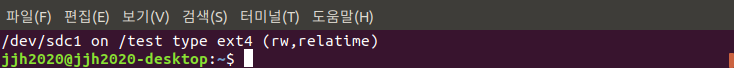
sudo fdisk –l로 정상적으로 설치된 것을 확인



sudo mkfs.ext4 /dev/sdc1으로 sdc1을 ext4 파일시스템으로 포맷

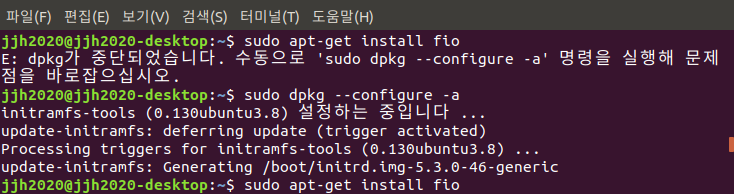


sudo mount /dev/sdc1 /test 를 입력하여 sdc1을 test폴더에 mount



mount 명령어로 mount가 된 것을 확인

## Fio 벤치마크 설치



sudo apt-get install fio 명령어로 fio 벤치마크 설치

만약 dpkg(데비안 패키지 관리시스템)에 문제가 있다면 수동으로 재설정

## XFS 파일 시스템 설치

C:\Users\user\AppData\Local\Microsoft\Windows\INetCache\Content.Word\스크린샷, 2020-04-15 12-13-57.png

# 실험

## Fio 벤치마크를 이용한 ext4파일시스템에서 read, write, randread, randwrite에 대한 값의 차이보기

Ex) 예시

sudo fio –directory=/test fio –name=read4k –rw=read –bs=4k –randrepeat=0 –size-1G –direct=0

–ioengine=sync –iodepth=16 –numjobs=16 –time\_based –runtime=180 –group\_reporting

--dirctory를 통해 경로를 지정

--name을 통해 이름을 지정

--rw를 통해 read/write를 지정

--bs를 통해 블록 사이즈 지정

--randrepeat

--size 작업당 i/o의 크기 지정

--direct direct i/o / buffered i/o 사용여부 지정

--ioengine 사용할 ioengine을 지정

--iodepth

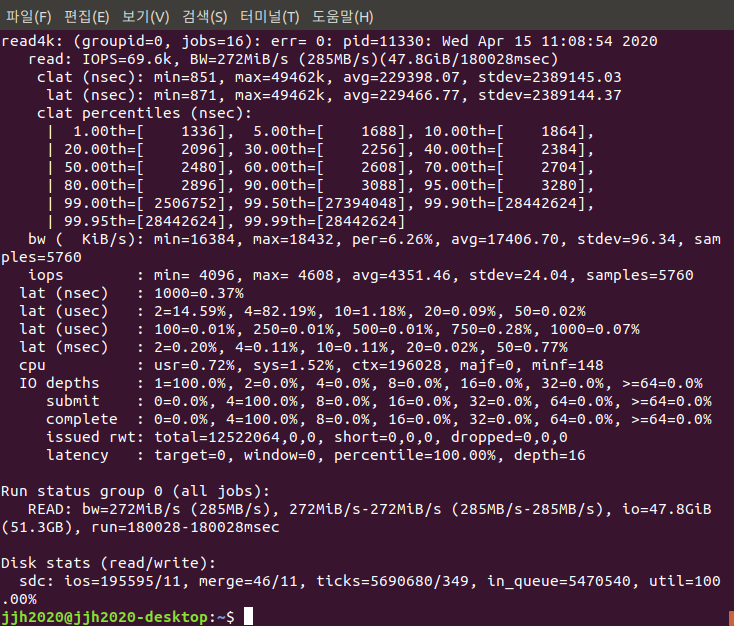
--numjobs

--time\_based

--runtime

--group\_reporting

rw = read , bs = 4k, randrepeat =0



# 실험결과

## Ext4 파일시스템에서 read, write, randread, randwrite에 대한 값의 차이

Disk Label Type이 실험에 영향을 줄수 있나? ex): mbr, gpt, dos

실험중 test option으로 값의 변화를 줄 수 있는게

blocksize, ioengine, iodepth, direct, fsync ?

block size : default value size가 4k, 읽기 쓰기에 쓰일 수 있음

random workloads에서 128k가 종종 쓰임

이 값을 변경하여 가능한한 실제 현장에서 쓰인 것처럼 사용해볼 수 있음

만약 서버가 4k block size로 99퍼의 시간을 다룬다면(차지한다면), 1mb block size로 왜 test를 해보나 <-?

ioengine

by default, fio는 sync io engine을 사용하여 테스트를 돌림

엔진은 변경할 수 있음. 다른 사용할 수 있는 많은 옵션들이 있지만,

커널에서 지원을 한다면 리눅스에서 가장 일반적인 옵션은 sync와 libaio

iodepth

이 옵션은 테스트동안 파일에 요청과 함께 두들기는걸(연속적인 접근?) 계속하면서 io unit의 총량을 규정한다.

default sync ioengine을 사용한다면, io depth의 값이 1을 넘어서는건 효과를 가지지 못한다.

libaro 같은 engine으로 변경해도 OS가 iodepth의 Maximum을 제한하고 specified value를 무시한다.

이것 때문에, io depth의 값을 1로 시작하고 값을 올려서 , 예를 들면 16까지, 실험하는걸 추천.

만약 퍼포먼스에서 어떠한 차이도 볼 수 없다면, 이 옵션을 건드리는걸 원하지 않았거나, direct io의 값이 1일 때 차이가 없음

모든 서버와 OS가 다르기 때문에 결과들을 저장하기 전에 여러 조합을 실험해봐라

direct

direct io를 쓰거나 buffered io 쓰는걸 말한다.

default value = 0 은 fio가 buffered io를 사용하는걸 뜻함

만약 값을 1로 정하고 buffered io를 사용하는걸 피한다면, 주 O\_DIRECT와 유사하다. ( 캐쉬를 거치지않는다?)

buffered io값을 사용하는게 non\_buffered io 사용하는 것보다 더 나은 퍼포먼스를 제공한다, 특히 read에서.

만약 ram이 용량이 엄청 큰 서버에서 테스트한다면 non\_buffered io를 쓰는게 부풀어진 결과값을 피하는데 도움이 된다. ssd가 600000 iops값을 계속 수행했다면, 이상한게 아니고, ram을 읽어낸 것, 명확히 엄청빠름

fsync

이 옵션은 얼마나 종종 fsync를 사용하여 disk의 "dirty data"를 씻어낼껀지를 말한다.

by default, fsync=0 은 sync를 하지않는다는걸 뜻한다.

많은 어플리케이션들이 이 옵션으로 수행하고 리눅스에서, 메모리에서 디스크로 데이터를 flush 하는걸 계산할 때 떠난다. 어플리케이션이나 서버가 디스크에 write할 때 항상 flush 한다면 이 옵션을 포함시키고 값을 1로 줘야한다.

어플리케이션이 각각의 write 후에 디스크에서 data를 flush하지 못하거나 잠재적인 데이터손실에 걱정이 없다면 이 값을 alone하게 둬라 (leave this value alone)

fsync값을 1로 하면 완벽하게 읽기의 버퍼링을 피할 수 있다.

만약 block device 에서 io performance의 최악의 경우를 보고 싶다면 fsync를 1로 하고 random write test를 돌리면 결과는 without fsync 보다 낮을 것이지만, single write operation이 디스크로 flush되고, 디스크는 스트레스를 받을 것이다.