Zoned Namespace를 이용한 SSD 성능 Isolation 기법 연구

팀명: 레알

201824532 윤건우

201724625 황인욱

201824593 조준서

지도교수 : 안 성 용 교수님

부산대학교 전기컴퓨터공학부 정보컴퓨터공학전공

School of Electrical and Computer Engineering, Computer Engineering Major
Pusan National University

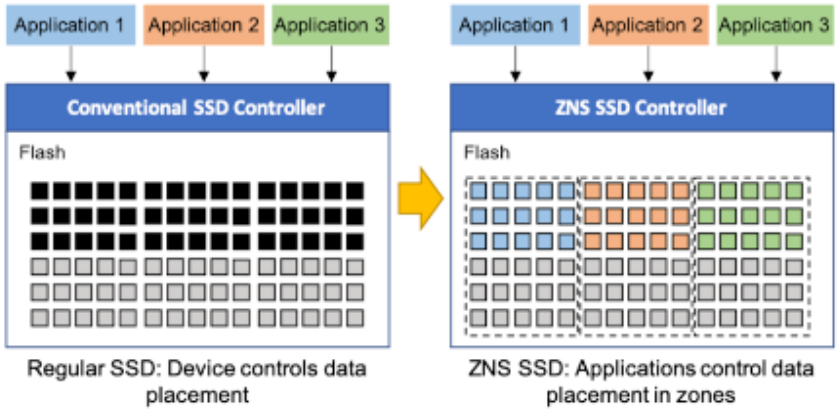
목차

1. 과제 배경

1.1 Zoned Namespace SSD

일반적인 SSD는 내부 저장 공간을 나누지 않고 여러 개의 프로세스에서 생성되는 데이터를 임의로 저장한다. 또한 덮어쓰기가 불가한 NAND Flash 특성 상 유효한 데이터와 불필요한 데이터가 혼재되어 저장 공간을 효율적으로 사용할 수 없다. 따라서 기존 SSD는 이를 해결하기 위해 유효한 데이터를 다른 빈 공간으로 옮겨 쓰고, 불필요한 데이터만 남은 영역을 지우는 Garbage Collection 작업이 필요하고 이 과정에서 추가적인 입출력이 발생한다.

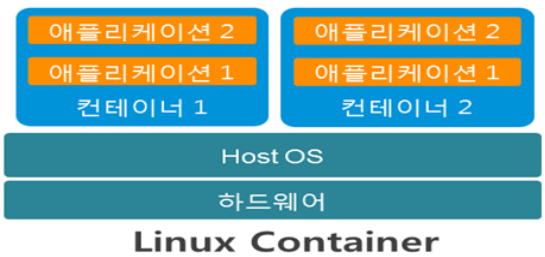
반면 Zoned Namespace SSD, 즉 ZNS SSD는 용도와 사용주기가 비슷한 데이터를 Zone 단위로 나누어 저장하고 지우기 때문에 기존 SSD의 Garbage Collection과 같은 작업으로 인한 추가적인 입출력이 발생하지 않는다.



[그림 1] 기존 SSD와 ZNS SSD의 데이터 배치

1.2 Linux Container

Linux Container란 리눅스 커널을 공유하면서 프로세스를 격리된 환경에서 실행하는 기술이다. 저장 공간을 Zone으로 나눠서 관리하는 ZNS SSD와 컨테이너의 프로세스 격리 환경을 결합한다면 컨테이너간의 간섭을 최소화하고 성능을 개선할 수 있다. 우리는 각 컨테이너의 입출력 작업을 ZNS SSD의 Zone에 분리 할당하는 정책을 개발하고자 한다.



[그림 2] Linux Container

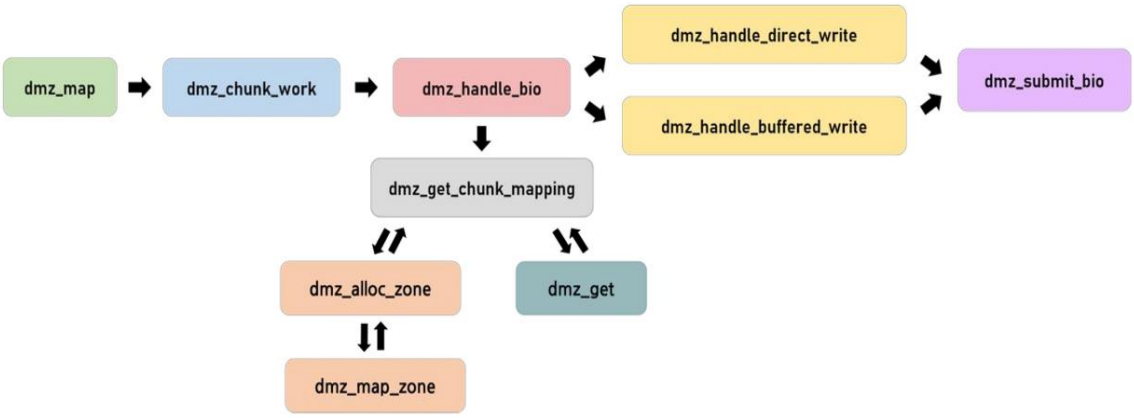
1.3 FIO

FIO란 Flexible I/O Tester의 약자로, 주로 디스크 및 파일 시스템의 성능을 테스트하고 측정하는데 사용되는 도구이다. FIO는 다양한 I/O 패턴과 설정을 사용하여 시스템의 입출력 작업을 시뮬레이션하고 성능을 평가한다.

FIO는 주로 명령줄 기반의 도구로 사용자가 성능 테스트에 필요한 다양한 파라미터와 옵션을 지정하여 테스트 작업을 정의하고 실행할 수 있다. 이렇게 정의된 작업은 "**jobfile**"이라고 불리는 설정 파일에 저장된다. Jobfile은 특정 작업의 세부 사항과 설정을 포함하며, FIO가 작업을 수행할 때 이 파일을 참조하여 작업을 수행한다.

2. dm-zoned 디바이스 매퍼

2.1 기존 코드 분석



[그림 3] Device Mapper Zone에서의 데이터 흐름

Block I/O(이하 bio)는 커널 내에서 블록 기반 장치와 상호 작용하기 위한 데이터 구조체로 블록 디바이스와 입출력 작업을 추상화하고 커널 내에서 블록 디바이스와 일관된 방식으로 작업할 수 있도록 도와준다. 이러한 bio는 Device Mapper Zone(이하 DMZ) target에서 다음과 같은 흐름을 보인다.

DMZ target에 bio(이하 입출력 작업)가 들어오면 입출력 작업을 처리하기 위해 dmz_map 함수를 호출한다. 입출력 작업을 적절한 크기로 나눈 후 dmz_queue_chunk_work 함수를 호출해 입출력 작업의 chunk 번호를 얻어 해당 번호의 작업을 radix_tree에서 찾는다. 작업이 존재하지 않는 경우 새로운 dm_chunk_work 구조체를 할당해 작업 큐에 추가하고, 작업 큐에 등록된 작업은 dmz_chunk_work 함수를 실행해 bio_list에서 bio를 하나씩 가져와 dmz_handle_bio 함수를 호출해 입출력 작업을 처리한다.

dmz_handle_bio 함수에서는 dmz_get_chunk_mapping 함수를 호출해 해당 입출력 작업에 대한 데이터 존 매핑을 가져오고, 입출력 작업의 연산 코드에 따라 읽기, 쓰기, 삭제, 블록 영역 초기화 등의 작업을 처리한다.

dmz_get_chunk_mapping 함수에서는 매핑된 데이터 영역의 dzone_id를 확인한다. 매핑되지 않은 상태일 때, 읽기 혹은 삭제 작업일 경우 매핑되지 않은 청크로 처리하고, 쓰기 작업인 경우에만 dmz_alloc_zone 함수를 수행해 새로운 데이터 영역을 할당하고 매핑을 수행한다.

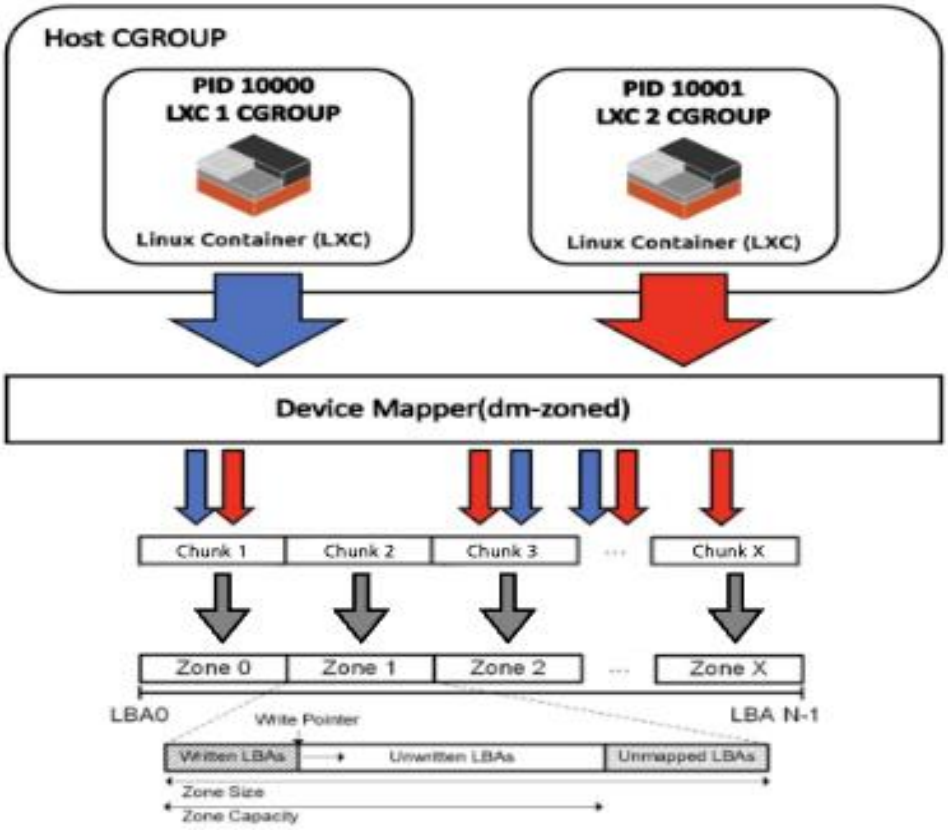
dmz_alloc_zone 함수는 데이터 영역을 할당하기 위해 flags 변수를 확인한다. 만약 DMZ_ALLOCK_RECLAIM 플래그가 없는 경우 회수 스케줄링을 수행하여 할당할 수 있는 데이터 영역이 보장되도록 하고 플래그가 설정되면 다른 장치에서 데이터 영역 할당을 시도한다. dmz_map_zone 함수는 zmd 매핑 테이블에서 해당 chunk에 대한 매핑 정보를 업데이트하고 dzone의 유형에 따라 매핑된 dzone을 적절한 리스트의 끝에 추가한다.

dmz_handle_write 함수는 입출력 작업이 쓰기 작업일 때 zone의 유형과 블록의 정렬 상태를 확인한다. 만약 랜덤 영역이거나 캐시 영역일 때 또는 순차 영역이면서 bio의 시작 블록이 해당 순차 영역의 쓰기 포인터와 정렬된 경우, 데이터를 직접 쓰는 작업을 수행한다. 그렇지 않으면 데이터를 버퍼에 기록하고 나중에 순차 영역의 쓰기 포인터에 맞춰 dmz_submit_bio 함수에서 주어진 bio를 복제한 후 실제 장치로 전송하는, 데이터를 쓰는 작업을 수행한다.

그 후 dmz_handle_bio에서 dmz_put_chunk_mapping을 통해 데이터 존과 버퍼 존의 유효성을 검사하고 필요한 경우 매핑을 해제, 회수하여 자원을 관리한다. 또한 dmz_bio_endio 함수를 통해 DMZ target의 입출력 작업의 완료를 처리하고, 데이터 존과 관련된 상태를 업데이트하여 정상적으로 작업이 처리되도록 한다.

2.2 기존의 매핑 방식

현재 ZNS SSD를 저장소로 사용하여 컨테이너를 생성한 후, 입출력을 요청하게 되면 device mapper인 dm-zoned를 거쳐서 chunk에 모이고 chunk들은 각 Zone에 매핑된다. 이때 현재 dm-zoned는 입출력 요청이 어떤 컨테이너로부터 생성되었는지 구분하지 않고 한 chunk가 다 찰 때까지 매핑한 후 Zone에 그대로 쓴다. 하나의 chunk는 하나의 Zone으로 매핑되기 때문에 결국 Zone에는 여러 컨테이너의 데이터들이 섞이게 된다. [그림2]는 이를 나타낸 것이다.



[그림 4] 현재 dm-zoned 매핑 방식

2.3 기존 방식 검증

2.3.1 실험 환경 구성

```
Disk /dev/nvme1n1: 8 GiB, 8589934592 bytes, 16777216 sectors
Disk model: FEMU BlackBox-SSD Controller
Units: sectors of 1 * 512 = 512 bytes
Sector size (logical/physical): 512 bytes / 512 bytes
I/O size (minimum/optimal): 512 bytes / 512 bytes


Disk /dev/nvme0n1: 8 GiB, 8589934592 bytes, 16777216 sectors
Disk model: FEMU ZNS-SSD Controller
Units: sectors of 1 * 512 = 512 bytes
Sector size (logical/physical): 512 bytes / 512 bytes
I/O size (minimum/optimal): 512 bytes / 512 bytes


Disk /dev/mapper/dmz-vSSD0: 13.77 GiB, 14763950080 bytes, 3604480 sectors
Units: sectors of 1 * 4096 = 4096 bytes
Sector size (logical/physical): 4096 bytes / 4096 bytes
I/O size (minimum/optimal): 4096 bytes / 4096 bytes
```

[그림 5] dm-zoned를 이용한 ZNS-SSD 가상 디바이스 생성

BlackBox-SSD와 ZNS-SSD를 dm-zoned를 사용해 가상의 ZNS 장치, dmz-vSSD0(이하 dm-0)을 생성한다. 해당 가상 장치에 ext4 파일 시스템을 사용해 기존 ZNS-SSD 동작 실험을 진행한다. 입출력에 사용할 파일에 대한 정보를 jobfile에 설정한다.

```
printk("%s: Cgroup ID %d Name %s's request is going to Chunk %u",__func__,
                css_id,
                css_name,
                chunk);
```

[그림 6] Cgroup과 Chunk 매핑 결과 출력 코드

dm-zoned-target.c의 dmz_handle_bio에 [그림 5]의 코드를 추가하여 파일 입출력이 일어날 때 어떤 Cgroup의 입력이 어느 Chunk로 가는지 출력하도록 하였다.

2.3.2 실험 결과

```
[  627.870718] dmz_handle_bio: Cgroup ID 24 Name cgroupA's request is going to Chunk 1
[  627.870817] dmz_handle_bio: Cgroup ID 24 Name cgroupA's request is going to Chunk 2
[  627.870887] dmz_handle_bio: Cgroup ID 24 Name cgroupA's request is going to Chunk 1
[  627.870950] dmz_handle_bio: Cgroup ID 32 Name cgroupB's request is going to Chunk 2
[  627.871034] dmz_handle_bio: Cgroup ID 32 Name cgroupB's request is going to Chunk 1
[  627.871117] dmz_handle_bio: Cgroup ID 32 Name cgroupB's request is going to Chunk 2
```

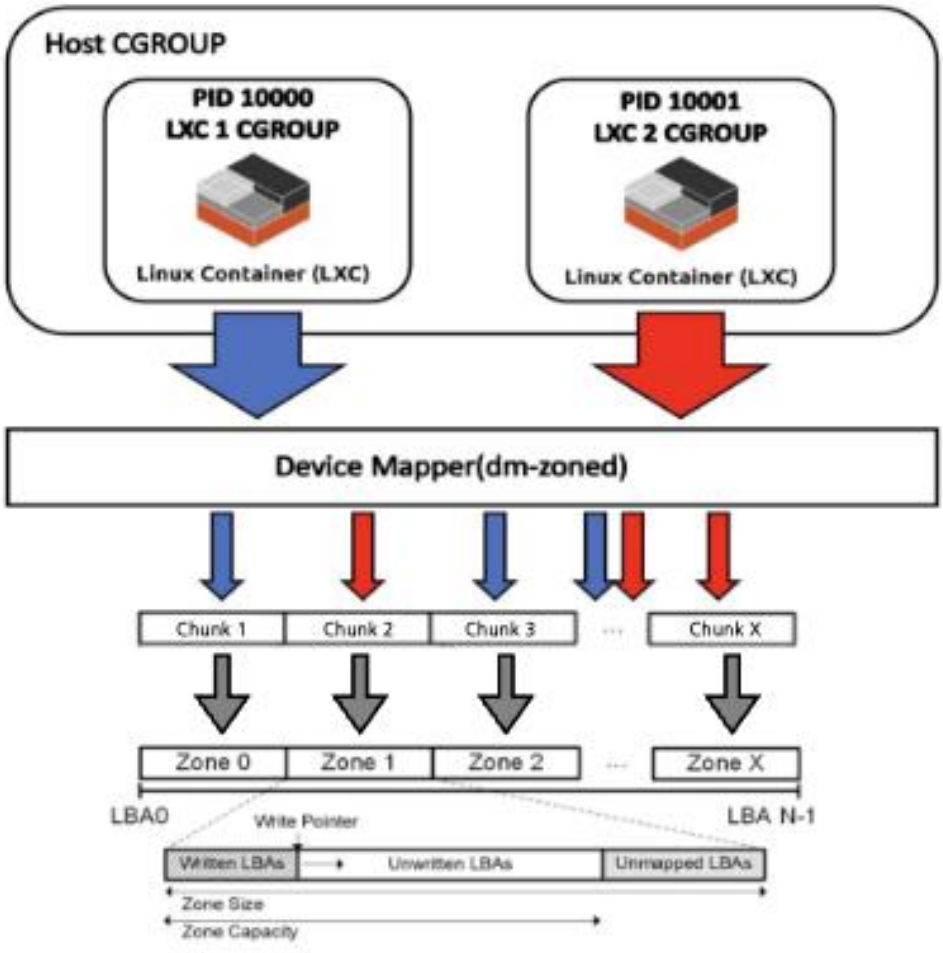
[그림 7] Cgroup과 Chunk 매핑 결과

[그림 6]을 보면 Cgroup ID가 24, 32인 두 Cgroup의 입력이 각각 Chunk 1, Chunk 2 모두에게 요청된 것을 확인할 수 있다. 이를 통해 현재 ZNS-SSD의 동작에서 Cgroup 별 분리가 제대로 이루어지지 않는 것을 확인할 수 있다.

3. 과제 목표

3.1 개선 방법

컨테이너간의 I/O Isolation을 위해서는 각 Zone에는 같은 컨테이너들의 데이터만 쓰여져야한다. 이를 위해서는 각 chunk에 같은 컨테이너의 BIO만 들어가야하고 dm-zoned의 동작과정을 수정해야한다. [그림 3]은 수정된 동작과정을 나타내고 있다.



[그림 8] 개선된 dm-zoned 매핑 방식

bio에는 해당 bio를 요청한 cgroup의 정보가 들어있다. 우리는 dm-zoned의 매핑 방식을 개선하기 위해 bio가 chunk에 매핑되기 전에 cgroup의 정보를 얻어, 같은 컨테이너에서 요청한 bio인지를 판단, 분리하여 매핑하는 것을 구현하려고 한다.

4. 과제 진행 내용

4.1 현재 과제 계획

<table>
<tr><th colspan="5">6월</th><th colspan="5">7월</th></tr>
<tr><td>1주</td><td>2주</td><td>3주</td><td>4주</td><td>5주</td><td>1주</td><td>2주</td><td>3주</td><td>4주</td><td>5주</td></tr>
<tr><td colspan="3">ZNS 스터디</td><td></td><td></td><td></td><td></td><td></td><td></td><td></td></tr>
<tr><td></td><td colspan="3">개발 환경 구축</td><td></td><td></td><td></td><td></td><td></td><td></td></tr>
<tr><td></td><td></td><td colspan="4">리눅스 커널 분석</td><td></td><td></td><td></td></tr>
<tr><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td colspan="2">중간보고서 작성</td></tr>
<tr><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td colspan="2">데이터 분리 기준 설정</td></tr>
</table>

4.2 향후 과제 계획

<table>
<tr><th colspan="5">8월</th><th colspan="5">9월</th></tr>
<tr><td>1주</td><td>2주</td><td>3주</td><td>4주</td><td>5주</td><td>1주</td><td>2주</td><td>3주</td><td>4주</td><td>5주</td></tr>
<tr><td colspan="5">zone 할당 알고리즘 개발</td><td></td><td></td><td></td><td></td><td></td></tr>
<tr><td></td><td></td><td></td><td colspan="2">테스트 및 디버깅</td><td></td><td></td><td></td><td></td><td></td></tr>
<tr><td></td><td></td><td></td><td></td><td colspan="2">1차 성능 분석</td><td></td><td></td><td></td><td></td></tr>
<tr><td></td><td></td><td></td><td></td><td></td><td colspan="2">알고리즘 개선</td><td></td><td></td><td></td></tr>
<tr><td></td><td></td><td></td><td></td><td></td><td></td><td colspan="2">2차 성능 분석</td><td></td><td></td></tr>
<tr><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td colspan="3">최종 발표 및 보고서 준비</td></tr>
</table>

4.3. 구성원별 진척도

윤건우	리눅스 커널 분석
황인욱	리눅스 커널 분석
조준서	리눅스 커널 분석