ZNS를 이용한 SSD 성능 Isolation 기법 연구

팀명 : 레알

201724625 황인욱

201824532 윤건우

201824593 조준서

지도교수 : 안성용

목차

1. 과제 배경 및 목표 3
1.1 과제 배경 3
1.2 기존 문제점 3
1.3 과제 목표 4
2. 연구 5
2.1 연구 배경 5
2.1.1 Block Interface SSD 5
2.1.2 ZNS SSD 5
2.1.3 Device Mapper (dm-zoned) 6
2.1.4 Linux Cgroup 7
3. 실험 환경구성 8
4. 개발 일정 및 역할 분담 9
4.1 개발 일정 9
4.2 구성원 역할 10

1. 과제 배경 및 목표

1.1 과제 배경

리눅스 컨테이너는 가장 널리 사용되는 클라우드 플랫폼 기술로 하나의 서버에 여러 개의 독립된 컴퓨팅 환경을 구축해 사용자들에게 제공한다. 또한, 운영체제 수준의 가상화 기술로서 애플리케이션의 실행 환경을 격리하고 시스템 자원을 할당함으로써 별도의 게스트 운영체제가 필요 없고 성능 오버헤드가 적다. Zoned Namespace SSD는 저장장치 내부를 zone이라는 일정한 크기의 영역으로 나누고 이를 zone 기반 인터페이스를 통해 호스트에서 zone에 대한 정보를 확인하고 관리할 수 있는 저장장치이다. 그러나 현재의 ZNS에서는 읽기/쓰기에 대한 영역 분리만 있을 뿐, 실제 SSD에 저장되어 있는 데이터들은 격리되어 있지 않다.

1.2 기존 문제점

현재 리눅스 컨테이너가 사용하는 block interface SSD는 페이지 단위로 쓰기를 실행한다. SSD에 데이터를 저장할 때는 접근하려는 페이지가 비어있는 상태일 때만 쓰기가 가능하다. 이미 쓰여진 데이터가 변경되려면 기존 페이지의 내용은 내부 레지스터로 복사 후 변경되어 비어있는 새로운 페이지에 기록되는 out-place update, 외부 업데이트 과정을 거친다. 또한, 페이지 크기보다 작은 쓰기가 요청되어도 무조건 하나의 페이지를 통째로 사용하는 write amplification, 부가적 쓰기가 발생한다. 이러한 외부 업데이트와 부가적 쓰기 제약으로 인해 유효한 데이터와 가비지(garbage) 영역의 분리가 일어나지 않아 효율적인 저장 공간의 사용이 어렵다.

ZNS SSD는 Zoned Namespace Command Set에 따라 동작하는 SSD로 SSD와 호스트가 데이터 배열을 구역화 된 스토리지 장치 인터페이스로 제공한다. 전체 저장공간을 작고 일정한 용량의 영역(zone)으로 나누어 별도의 영역에 순차적으로 쓰기 하는 것이 가능하다. 이를 통해 데이터를 SSD의 물리적 미디어에 정렬해 전체 성능을 개선하고 호스트에 제공될 수 있는 용량이 증가된다. 하나의 영역에는 유사한 데이터들이 모여 있으며 영역 단위로 삭제가 이루어지기 때문에 가비지가 발생하지 않는다. 그러나 현재의 ZNS SSD를 저장소 풀로 지정해 컨테이너를 생성하게 되면 device mapper를 거쳐 여러 개의 영역에 컨테이너의 입출력이 발생한다. 컨테이너를 추가로 생성하는 경우 비어 있는 영역을 할당하지 않고 기존에 생성된 컨테이너가 사용하고 있는 영역에 추가하는 형식으로 입출력이 발생한다. 결국 컨테이너마다 독립된 영역을 할당 받지 못하고 이미 다른 컨테이너가 사용하고 있는 영역을 공유하므로 ZNS SSD의 장점이 제대로 활용되지 않는다

1.3 과제 목표

ZNS SSD를 씀으로써 입출력 스트림을 서로 다른 영역에 기록해 스트림 간의 성능 간섭을 제거하거나 각 컨테이너들에게 서로 다른 영역 할당해 입출력 요청에 대한 환경 및 성능에 대한 격리를 기대한다. 하지만 현재 리눅스 컨테이너는 ZNS SSD를 지원하지 않기 때문에 단순히 저장 장치의 종류를 변경하는 것만으로는 ZNS SSD의 이점을 활용할 수 없고 현재의 dm-zoned, 영역 기반 저장장치를 위한 디바이스 매퍼는 논리적인 블록 SSD를 에뮬레이트 할 수 있으나 단순히 데이터를 영역 단위로 관리하는 것 외에는 부가적인 기능이 없다. 또한, 순차 쓰기제약(Sequential write constraint)이 있기 때문에 실질적인 컨테이너별 영역 분리 효과는 기대하기 어렵다. 때문에 이번 과제에서는 기존의 영역 관리 방식에 컨테이너별 데이터 분리 기준을 추가하여 각각의 컨테이너가 서로 다른 영역을 사용하도록 할당한다면 입출력 스트림 간의 간섭이 제거되는 효과를 기대할 수 있다. 따라서 본 팀은 각 컨테이너에서 입출력이 발생할 시 사용되는 데이터들을 영역 단위로 격리시키는 알고리즘을 개발, 적용하여 리눅스 컨테이너가 ZNS SSD의 특성을 제대로 활용할 수 있도록 하는 것을 목표로 한다.

2. 연구

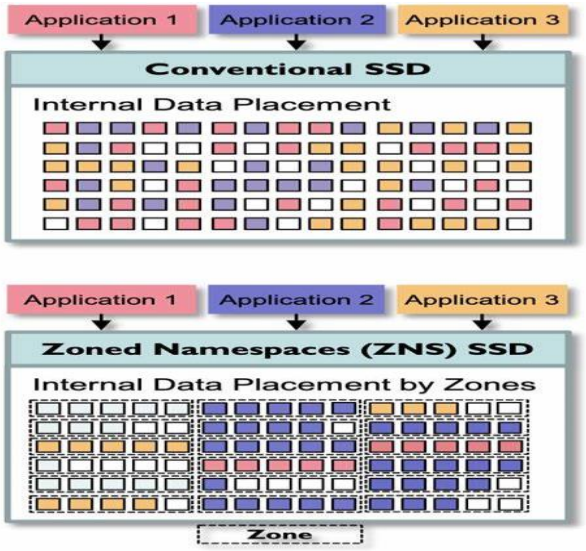
2.1연구 배경

2.1.1 Block Interface SSD

Block Interface SSD는 저장 장치를 임의의 순서로 읽고, 쓰고, 덮어쓸 수 있는 고정 크기 논리 데이터 블록의 1차원 배열로 표시한다. 여러 개의 소프트웨어에서 생성되는 데이터들은 논리적으로 원하는 영역에 저장되는 것처럼 보일지 몰라도 물리적으로는 Nand Block에 순차적으로 저장된다. Nand Flash의 특성상 이 영역에는 유효한 데이터와 가비지 데이터가 혼재하기 때문에 저장 공간을 효율적으로 사용하기 어렵다. 따라서 가비지 콜렉션 작업을 통해 공간을 확보하는데 이는 유효한 자원들을 모아 다른 Nand Block에 다시 쓰기를 하는 오버헤드가 큰 방식이다. 또한 가비지 콜렉션이 일어나는 동안은 현재 실행 중이던 읽기/쓰기 작업이 중단되기 때문에 성능 저하가 발생한다. 이러한 블록 인터페이스와 저장장치 특성 사이의 불일치를 SSD의 플래시 변환 계층(FTL)은 동적 논리-물리 페이지 매핑 구조를 위해 상당한 양의 DRAM을 사용하고, 드라이브 미디어 용량의 상당 부분을 오버 프로비저닝하여 삭제 블록 데이터의 가비지 컬렉션 오버헤드를 낮춤으로서 완화한다. 그럼에도 불구하고 가비지 컬렉션은 처리량 제한, 부가적 쓰기, 성능 예측 불가능, 높은 지연 시간을 가진다.

2.1.2 ZNS SSD

Zoned Namespace 기법을 이용한 차세대 저장장치이다. 단일 논리 블록 배열 대신 영역 단위로 순차적으로 저장하고 지우기 때문에 가비지 컬렉션으로 인한 추가적인 입출력이 발생하지 않는다. 각 영역은 쓰기 포인터를 가지며, 해당 영역으로 내려온 쓰기 명령은 쓰기 포인터에서 순차적으로 진행 된다. 영역 논리 블록은 임의의 순서로 읽을 수 있으나 반드시 순차적으로 써야 하며, 다시 쓰기 전에 영역을 지워야 한다. 영역과 물리적 미디어 경계를 정렬, 데이터 관리의 책임을 호스트로 전환해 디바이스 내 가비지 수집과 리소스 및 용량 오버 프로비저닝이 필요하지 않다.

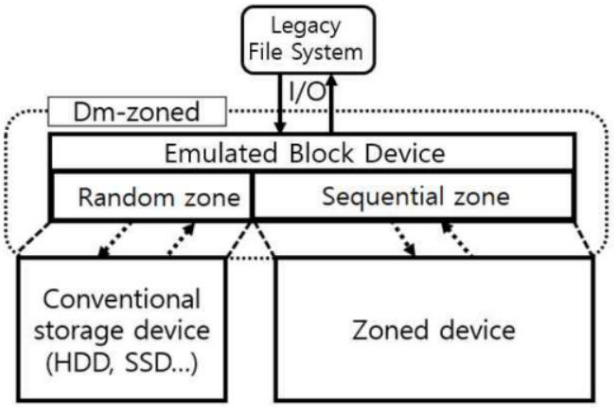


[그림 1] 일반적인 SSD와 ZNS SSD의 저장방식 비교

ZNS SSD는 덮어 쓰기를 하려면 명시적으로 재설정을 해야 하며 쓰기 제약 조건은 영역별 상태 시스템과 쓰기 포인터에 의해 적용이 된다. 영역 별 상태에는 '비어 있음', '열림', '닫힘', '가득 참' 4가지가 있다. 영역은 '비어 있음' 상태에서 시작하여 쓰기 시 '열림' 상태로 전환되고 마지막으로 완전히 쓰여지면 '가득 참' 상태로 전환된다. 디바이스 리소스 또는 미디어 제한 등으로 동시에 '열림' 상태에 있을 수 있는 영역 수에 대한 개방 영역 제한을 추가로 부과하는 가능하며 제한에 도달하여 호스트가 새 영역에 쓰기를 시도하면 다른 영역을 '열림' 상태에서 '닫힘' 상태로 전환하여 쓰기 버퍼와 같은 장치 내 리소스를 확보해야 한다. '닫힘' 상태의 영역은 여전히 쓰기가 가능하지만 추가 쓰기를 제공하기 전에 다시 '열림' 상태로의 전환이 필요하다.

2.1.3 Device Mapper (dm-zoned)

ZNS SSD는 기존의 블록 저장장치와 달리 순차쓰기제약(Sequential write constraint)이 있기 때문에 기존 파일시스템을 그대로 사용할 수 없다. 이 때 리눅스 커널에서는 기존 파일시스템이나 애플리케이션이 수정없이 ZNS SSD를 사용할 수 있도록 dm-zoned라고 하는 영역 기반 저장장치를 위한 디바이스 매퍼를 제공하고 있다. ZNS SSD는 순차 쓰기만 가능하나 dm-zoned는 사용자의 임의 쓰기 요청을 처리하기 위해 표준 블록 저장장치를 이용한다. 이를 통해 레거시 파일시스템과 애플리케이션들은 수정없이 ZNS SSD에 접근이 가능하다. 이렇듯 dm-zoned는 사용자의 요청과 하위 저장장치 사이에서 중개자 역할을 하여 요청의 유형과 저장장치의 기능에 따라 요청을 적절한 장치로 재설정한다. 또한, 사용자가 기존 소프트웨어를 업데이트하지 않고도 ZNS SSD의 성능 및 효율성 이점을 활용할 수 있으며, 순차쓰기제약과 같은 영역 기반 저장장치 관리의 복잡성을 처리하여 사용자가 하위 저장 기술이 아닌 애플리케이션에 집중할 수 있도록 한다.



[그림 2] dm-zoned를 통한 ZNS 가상화

dm-zoned는 표준 블록 저장장치와 ZNS SSD를 하나로 묶어 마치 하나의 블록 인터페이스 저장장치처럼 사용할 수 있도록 가상화한다. 또한, 쓰기 및 읽기 요청을 해당 요청이 속하는 chunk 기반으로, 동일한 chunk 영역으로 들어오는 요청들을 모아 하나의 작업 단위로 모아둔 요청을 처리한다.

2.1.4 Linux Cgroup

Linux 커널 시스템에서 단일 또는 task 단위의 프로세스 그룹에 대해 시스템 자원 할당, 우선순위 지정, 거부, 관리, 모니터링 등과 같은 세밀한 제어를 가능하게 해준다. 계층적으로 구성되어 있으며 하위 cgroup은 부모 cgroup 속성의 일부를 상속받도록 되어 있으며 여러 다른 cgroup 계층이 시스템에 동시에 존재할 수 있다. 각 계층은 하나 이상의 서브 시스템 (단일 자원 - CPU 시간, 메모리 등)에 연결되므로 cgroup에 대해 여러 분리된 계층이 필요하다. 특정 경로 아래에 있는 디렉토리 또는 파일을 조작하여 cgroup을 제어할 수 있으며 파일들 중 task 파일 안의 프로세스 PID 들을 수정해 프로세스들의 시스템 자원을 제한할 수 있다.

3. 실험 환경 구성

3.1 실험 환경

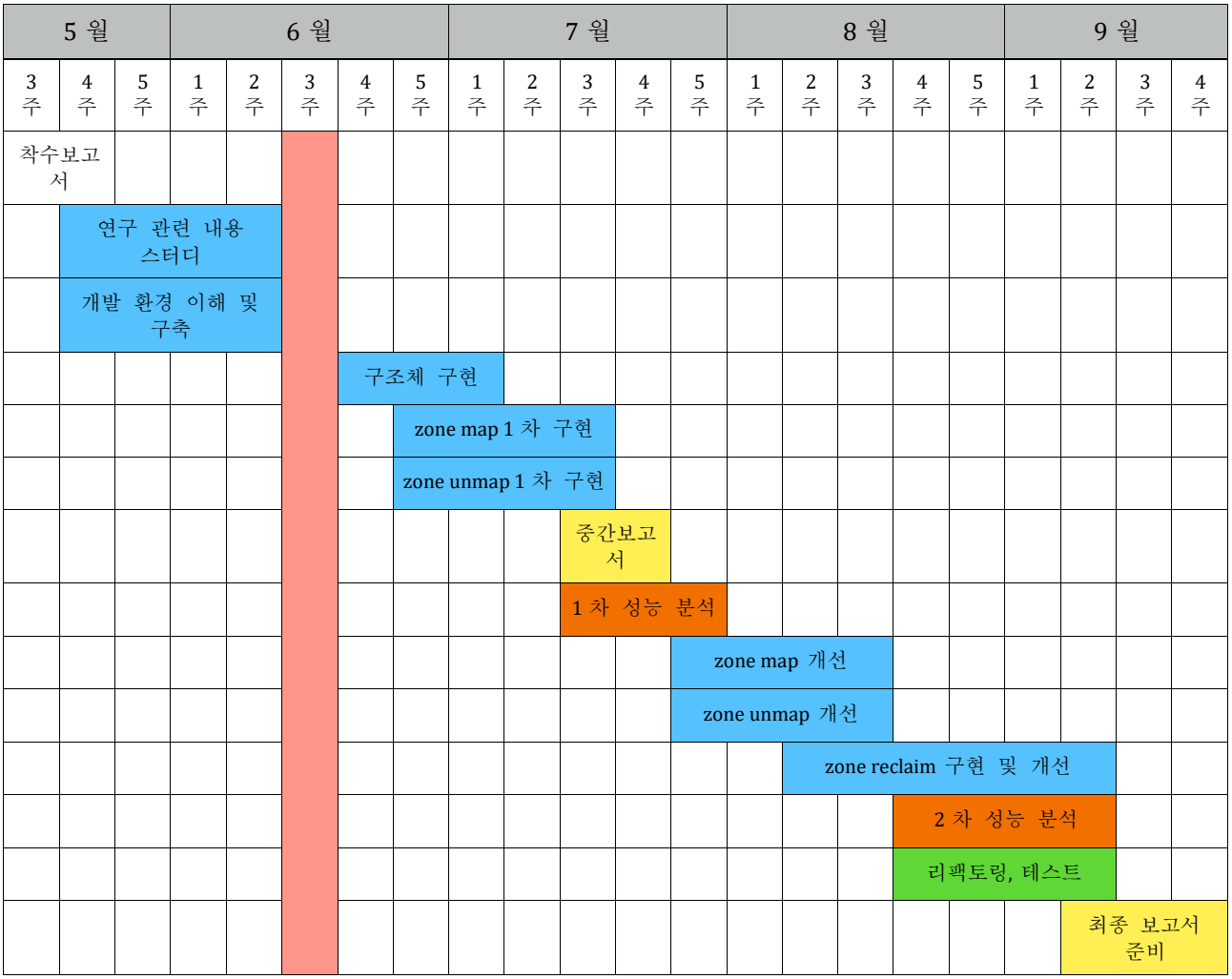
CPU	Intel(R) Core(TM) i7-1165G7 @ 2.80GHz
Memory	122GB
OS	Ubuntu 20.04.4 LTS, Linux 5.10
Virtual Environment	Femu

[표 1] 시스템 사양

실물 ZNS SSD를 구할 수가 없어서 ZNS SSD 에뮬레이팅을 위한 가상환경인 FEMU를 설치해서 실험 환경을 구성한다.
FEMU의 설치 과정은 다음 github 주소의 REAME 파일을 통해서 알 수 있다. https://github.com/vtess/FEMU

4. 개발 일정 및 역할 분담

4.1개발 일정



4.2 구성원 역할

이름	역할
윤건우	- 구조체 이해 및 구현 - map, unmap 알고리즘 구현 - reclaim 알고리즘 구현 - 총괄 및 보고서 작성, 발표
황인욱	- 구조체 이해 및 구현 - map 알고리즘 구현 - reclaim 알고리즘 구현 - 리팩토링 및 테스트
조준서	- 구조체 이해 및 구현 - unmap 알고리즘 구현 - reclaim 알고리즘 구현 - 성능 분석

참고 문헌.

1. ZNS SSD (Zoned Namespaces SSD)와 NVMe (Non-Volatile Memory Express)의 개요,
https://smartits.tistory.com/243

2. SSD란 무엇인가? [SSD와 HDD의 차이]
https://cosyp.tistory.com/148

3. 컨테이너란? 리눅스의 프로세스 격리 기능
https://www.44bits.io/ko/keyword/linux-container#%EC%BB%A8%ED%85%8C%EC%9D%B4%EB%84%88%EC%9D%98-%EC%A2%85%EB%A5%98-%EC%8B%9C%EC%8A%A4%ED%85%9C-%EC%BB%A8%ED%85%8C%EC%9D%B4%EB%84%88%EC%99%80-%EC%95%A0%ED%94%8C%EB%A6%AC%EC%BC%80%EC%9D%B4%EC%85%98-%EC%BB%A8%ED%85%8C%EC%9D%B4%EB%84%88

4. 리눅스Linux) cgroups 개념, 특징, 기능, 계층 구조, 실습 예시
https://seungyooon.tistory.com/142

5. TypingIsBetterThanLooking/TBTL-Server
https://github.com/TypingIsBetterThanLooking/TBTL-Server

6. [그림 1] Zone Namespaces SSD — Wiki for Computer System and Architecture (0x10.sh)

7. SSD https://pages.cs.wisc.edu/~remzi/OSTEP/file-ssd.pdf
8. ZNS https://www.usenix.org/system/files/atc21-bjorling.pdf