# Trim 오버헤드 제거를 위한 Delayed TRIM 기법

김정헌 ○, 신동군

성균관대학교 전자전기컴퓨터공학과

kjh990705@skku.edu, dongkun@skku.edu

# Delayed TRIM Techniques for Reducing Trim Overhead

Jungheon Kim○, Dongkun Shin

Department of Electrical and Computer Engineering, Sungkyunkwan University

**요 약**

플래시메모리는 물리적 특성 때문에 덮어쓰기가 불가능하다. 이에 따라 플래시메모리 기반의 저장 장치인 SSD는 덮어쓰기 요청 시 Out-of-place 업데이트를 활용하여 기존 데이터를 무효화한다. 무효화된 공간을 회수하기 위해서 FTL(Flash Translation Layer)은 GC(Garbage Collection)를 통해 유효한 페이지만 다른 블록으로 복사한 후, 해당 블록을 지운다. 하지만 파일 시스템에 의해 삭제된 페이지에 대한 정보가 디바이스로 바로 전달되지 않아 FTL은 삭제된 페이지에 대해서도 유효한 페이지로 잘못 인식해 복사한다. 이러한 불필요한 복사를 방지하기 위해 호스트는 디바이스에게 Trim으로 해당 블록이 더 이상 유효하지 않다는 정보를 전달한다. 본 연구에서는 Trim의 오버헤드를 분석하고, 그 오버헤드를 제거하기 위해, Trim이 도착했을 때 즉시 처리하지 않고 디바이스가 유휴 상태로 들어갔을 때 처리하는 Delayed TRIM을 제안한다. Delayed TRIM은 기존 Trim의 오버헤드를 90% 이상 제거한다.

## 1. 서 론

SSD는 플래시메모리 기반의 저장 장치로 널리 사용되고 있다. SSD의 펌웨어인 FTL은 블록 내 유효한 페이지를 복사하고 무효한 페이지를 지우는 GC 작업을 수행한다. 그러나 호스트가 데이터를 삭제했을 때, 디바이스는 해당 데이터가 삭제되었다는 정보를 바로 얻지 못한다. 파일이 저장된 논리 주소에 새로운 데이터가 기록되기 전까지는 기존 데이터가 여전히 유효하다고 간주되며, 이로 인해 GC 동작 중 불필요한 데이터 복사가 발생한다. 이러한 불필요한 데이터 복사 동작은 추가적인 쓰기 동작을 야기한다. 따라서 불필요한 쓰기 동작을 최소화는 것이 중요하다. 이를 위해 Trim[1]을 활용할 수 있다. Trim을 사용하면 호스트가 삭제된 데이터를 디바이스에 알릴 수 있으므로, FTL이 해당 페이지를 무효한 것으로 간주하고 GC 동작 중 불필요한 복사를 줄일 수 있다.

NVMe 디바이스에서는 Trim을 Deallocation으로 부르며 Dataset Management 명령에 의해 처리된다. Trim을 Discard라고 부르기도 한다.

표 1은 Samsung SSD980 1TB를 대상으로 fio[2] 벤치마크를 이용하여 10MB씩 10GB Write와 10MB씩 3GB Trim 작업을 동시에 수행한 실험과 10MB씩 10GB Write만을 단독으로 수행한 실험의 전체 워크로드 수행 시간 차이를 비교한 결과를 나타낸다. 표 1에서 볼 수 있듯 3GB에 대한 Trim 동작이 전체 워크로드의 수행 시간의 약 10%을 차지하는 것을 확인할 수 있다.

표 1. Trim Overhead

| 구분 | Write&Trim | Write |
|---|---|---|
| Write latency(msec) | 3644 | 3605 |
| Trim latency(msec) | 375 | 0 |
| Total latency(msec) | 4019 | 3605 |

Trim은 무효화할 범위에 대한 정보를 저장하기 위해 명령을 처리하는 단계, Trim 범위에 해당하는 쓰기 버퍼의 데이터를 탐색해 제거하는 단계와 Trim 범위에 해당하는 페이지를 매핑 테이블에서 무효화하는 단계로 구성된다. 본 연구에서는 각 단계별 Trim의 오버헤드를 측정하기 위해 DaisyPlus OpenSSD 하드웨어 플랫폼에 Trim 명령을 구현한 후 실험을 진행하였다. 실험에서는 4KB 논리 블록 16개씩 256개 Range로 Trim 명령을 디바이스에 전달하여 Trim 작업을 수행하였다. 표 2는 Trim 작업의 전체 오버헤드에서 각 단계별 비율을 측정한 결과를 나타낸다. 이를 통해, Trim 작업 중 쓰기 버퍼와 매핑 테이블을 처리하는 시간이 전체 시간의 약 80%를 차지한다는 사실을 확인할 수 있었다.

표 2. Trim Overhead 분석

| 구분 | Overhead (%) |
|---|---|
| Command Handling | 20.2 |
| Write Buffer | 40.1 |
| Mapping Table | 39.7 |

Trim 작업의 오버헤드가 크기 때문에 이를 줄이기 위한 여러 실험이 수행되었다. *iDiscard*[3]는 파일 시스템의

* 이 성과는 정부(과학기술정보통신부)의 재원으로 한국연구재단의 지원을 받아 수행된 연구임 (No. NRF2021R1A2C2095125).

불필요한 inode 블록에 매핑된 플래시 페이지를 동적으로 무효화하는 방법을 제안했다. 하지만 이 방법은 Trim이 수행되는 동안 다른 I/O 작업의 지연을 막지 못하는 문제점이 있다. *iTrim*[4]은 I/O 경합을 피하기 위해 Trim 크기와 논리 주소의 무효화 데이터 패턴을 고려하는 방법을 제안했다. 이 경우에도 I/O의 경합을 줄일 수 있었지만, 경합이 발생할 때의 지연은 여전히 불가피 했다. 본 연구는 Trim 요청이 디바이스에 전달될 때 처리시간이 짧은 명령 처리 단계만 수행하고, 처리시간이 긴 작업은 디바이스가 유휴 상태로 들어갈 때 처리하며, Trim 처리 중 I/O 요청이 발생하면 I/O 요청을 우선 처리하는 Delayed TRIM 방식을 제안한다. 이를 통해, 호스트 측에서의 Trim 처리로 인한 지연을 최소화할 수 있다.

## 2. Delayed TRIM

### 2.1 Delayed TRIM 시나리오

그림 1은 Trim이 지연되면서 발생할 수 있는 문제 상황을 나타낸다. I/O 요청 뒤의 두 숫자는 각각 시작 LBA와 블록의 개수를 의미한다. 먼저, Trim이 지연되는 동안 Trim이 수행되어야 하는 영역에 대해 Write가 요청될 수 있다(그림 1의 1번). t2에 요청된 Trim은 디바이스가 IDLE한 t4 때 Trim을 수행하게 되며, LBA 100번부터 200개의 블록을 무효화한다. t1에 도착한 Write 요청의 경우는 문제가 생기지 않는다. 하지만 t3에 요청된 Write가 LBA 150부터 50개의 블록에 유효한 Data를 적었지만 Delayed TRIM에 의해 무효화 처리되는 문제가 생긴다. 두번째로, t5에 도착한 Trim이 처리되기 전에 GC가 발생할 수 있다(그림 1의 2번). 이 경우에는 Trim이 반영되지 않아 Trim이 요청된 무효화되어야 하는 블록이 유효한 상태로 남아있어 불필요한 복사 동작이 발생하는 문제가 생긴다.

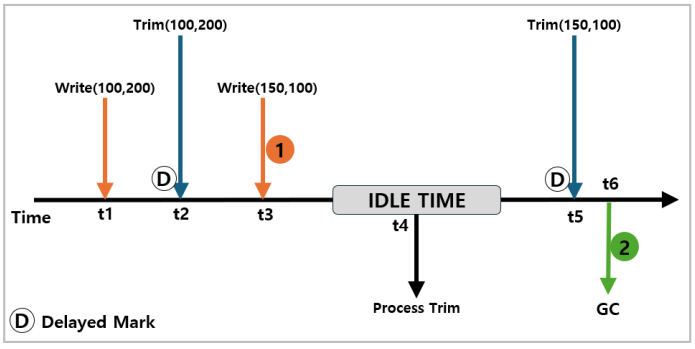


그림 1. Delayed TRIM Scenario

### 2.2 Delayed TRIM 명령 처리 단계

Delayed TRIM은 Trim 명령 처리만을 동기적으로 수행한다. 이 단계에서는 Trim의 시작 LBA와 Trim 해야 하는 블록의 개수를 저장한다. 이 과정이 완료되면, 디바이스는 호스트에게 요청이 완료되었음을 알리고 다음 I/O 요청을 처리한다. 그러나 쓰기 버퍼와 매핑 테이블에 대한 처리가 아직 완료되지 않았다. 따라서 그림 1의 1번과 같은 상황을 피하기 위해 Trim이 지연되는 동안 도착하는 쓰기 요청은 Check Write List에 기록된다. 이 리스트에는 쓰기 요청의 시작 LBA, 블록 개수, 해당 쓰기 요청이 어느 Trim 직후에 도착했는지에 대한 정보가 저장된다.

### 2.3 Delayed TRIM 처리 단계

I/O 요청이 없어 디바이스가 유휴 상태에 들어가면, 지연된 Trim 작업을 반영한다. 이때 Trim 대상 영역에 Trim이 지연되는 동안 새로운 데이터가 쓰였다면 Trim이 수행되지 않아야 한다. 이를 방지하기 위해, Delayed TRIM은 Trim 할 페이지 중 Check Write List에서 해당 Trim 이후에 도착한 Write 요청과 겹치는 페이지가 있는지 탐색한다. 겹치는 페이지가 있으면, 유효한 논리 블록을 제외하고 Trim 해야 하는 논리 블록만 쓰기 버퍼와 매핑 테이블에서 무효화한다. 지연된 Trim을 처리하는 도중 새로운 I/O 요청이 도착하면, 수행 중이던 Trim 작업의 Context를 저장하고 Trim을 종료한다. 이 Context는 Trim 처리가 다시 시작될 때 복원된다. 이 과정을 통해 Trim 작업 중 I/O 요청으로 인한 중단을 처리하고 작업을 재개할 수 있다. 또한 Trim이 지연되는 동안 GC가 발생하면, 지연된 모든 Trim을 반영한 뒤 GC가 동작하도록 한다. GC에 의해 Trim이 동작된 경우, Trim 처리 중간에 I/O 요청이 도착하는지 별도로 확인하지 않는다.

**Algorithm 1** Delayed Trim Process

```
1: Input: Trim Range, Write Buffer, Mapping Table
2: restoreTrimContext()
3: for LPN in Trim Range do
4:     Trim Index = getTrimIndex(LPN)
5:     hashEntry = checkWriteList(LPN, Trim Index)
6:     if hashEntry is valid then
7:         Valid Blocks = CompareValidBlocks(Trim Range[LPN],
   checkWriteList[hashEntry])
8:     end if
9:     if LPN is Valid in Write Buffer then
10:        invalidateWriteBuffer(LPN, Valid Blocks)
11:    end if
12:    if (LPN is Valid in L2P Table) or (PPN is Valid in P2L Table) then
13:        invalidateMappingTable(LPN, Valid Blocks)
14:    end if
15:    if !GarbageCollection && CheckNewIORequest() then
16:        saveTrimContext()
17:        trimInProgress = true
18:        return
19:    end if
20: end for
```

알고리즘 1. Delayed TRIM 처리 과정

## 3. 구 현

본 연구에서는 Delayed TRIM을 DaisyPlus OpenSSD의 GreedyFTL 3. 0. 0을 수정하여 구현했다. GreedyFTL3.0.0은 논리 블록을 4KB로 관리하며 연속된 4개의 논리 블록을 16KB 크기의 논리 페이지에 저장한다. DaisyPlus OpenSSD의 펌웨어에는 Write, Read, Flush, Write Zero 네 가지 I/O 명령만 구현되어 있었기 때문에, NVMe 명령 중 Dataset Management를 추가 구현하여 Trim 작업을 수행할 수 있도록 구현했다.

Delayed TRIM에서는 Check Write List를 탐색할 때 시간이 오래 걸리는 문제가 있다. 이를 개선하기 위해 Check Write List의 해시 엔트리를 관리하는 구조를 도입했다. 해시 구조는 Buffer Check 정책과 유사하게 설계되었다. 각 LPN에 대해, 해시 함수는 LPN을 해시 엔트리 길이로 나눈 나머지를 계산한다. 이 결과값을 해시 테이블의 인덱스로 사용하여, 해당 인덱스에 리스트 형태로 LPN과 관련된 Check Write List 정보를 연결 지었다. 이러한 개선을 통해 Delayed TRIM이 효율적으로 작동하도록 설계되었으며, Trim 처리를 빠르고 정확하게 수행할 수 있다.

## 4. 실 험

### 4.1 실험 환경

본 연구는 DaisyPlus OpenSSD 하드웨어 플랫폼을 사용하였다[5]. DaisyPlus는 256GB 낸드 플래시 메모리와 2GB DRAM이 장착되어 있다. 호스트 PC는 인텔 i7-7700 3.60GHz 프로세서와 40GB DRAM, 1TB Samsung SSD 980을 장착했다. 실험은 Ubuntu 18.04.6 LTS와 Linux 4.15.0-210-generic 커널에서 진행했다.

### 4.2 실험 방법

실험에서는 fio 벤치마크를 사용하여 쓰기와 Trim 워크로드를 테스트했다. 논리 블록 크기는 4KB로 설정했고, direct I/O, sync io engine, stone wall 옵션을 사용해 100MB 파일을 쓰고 삭제하는 방식으로 진행했다. Baseline(Normal Trim)과 Delayed TRIM에 대해 한 번의 Trim 작업에서 삭제되는 블록 수를 256KB와 4MB로 설정하여 실험했다. Trim 처리 시간을 측정하기 위해, 디바이스 코어의 클럭 수를 기반으로 계산하여 시간을 산출했다. 이를 통해 Trim 작업의 효율성과 Delayed TRIM이 Baseline에 비해 얼마나 효과적으로 지연 시간을 줄일 수 있는지 평가했다.

### 4.3 실험 결과

그림 2는 100MB의 파일에 대해 Trim 작업을 수행할 때, Baseline 방식과 제안된 Delayed TRIM 방식의 디바이스 내부 Trim 처리 시간을 비교한 결과를 나타낸다. 그래프의 값은 각 방식의 처리 시간을 동일한 블록 수로 TRIM을 처리한 Baseline의 처리 시간으로 정규화 한 값을 나타낸다. 그래프에서 각 막대의 하단 영역은 요청 명령이 Request Queue에서 I/O 요청을 Fetching하는 공통 영역을 나타낸다. 상단 영역은 Trim을 처리하는 시간을 나타낸다. 이를 통해 Baseline과 Delayed TRIM 사이의 처리 시간 차이와, Trim 작업이 전체 I/O 요청에서 차지하는 비중을 시각적으로 비교할 수 있다. 결과적으로 Delayed TRIM이 Baseline 대비 효율적인 지연 관리를 통해 I/O 지연을 최소화하는 효과가 있는지 평가할 수 있다. 100MB파일에 대해 256KB씩 Trim을 처리하는 경우 Trim 수행시간은 Delayed TRIM이 Normal Trim에 비해 약 91.95% 줄어든 것을 확인할 수 있다. 4MB씩 Trim을 처리하는 경우는 약 99.44% 줄어들었다. 표2에서 확인할 수 있듯 Trim에 대한 데이터를 저장하는데 필요한 처리 시간은 전체 Trim 처리 시간의 일부이기 때문에 데이터를 저장만 하고 처리를 완료하는 Delayed TRIM의 수행시간이 90% 이상 빠르게 완료되었다. 4MB씩 Trim하는 경우는 256KB씩 Trim 하는 경우는 보다 쓰기 버퍼와 매핑 테이블을 처리하는 비율이 더 크기 때문에 처리 시간이 줄어드는 비율이 더 크다

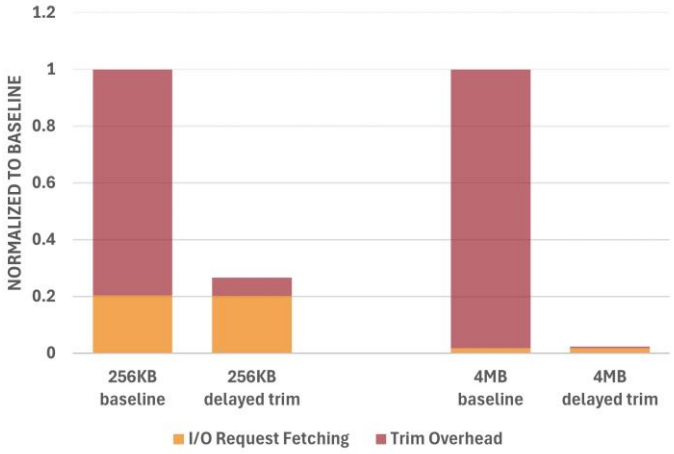


그림 2. Delete 100MB File

## 5. 결론 및 향후 연구 계획

본 연구에서는 기존 Trim을 개선한 Delayed TRIM을 제안한다. 본 연구는 Trim의 처리 시간을 분석하여 가장 많은 영역을 차지하는 쓰기 버퍼와 매핑 테이블처리하는 단계를 디바이스가 유휴상태일 때 처리하게 함으로써 Trim의 처리 시간을 90%이상 가리는 것을 확인하였다. 향후 연구에서는 유휴상태 일 때 Trim을 처리하는데 필요한 시간을 줄이는 최적화 연구를 진행할 계획이다.

## 참고 문헌

[1] F. Shu, Notification for Deleted Data Proposal for ATA-ACS2, http://t13.org/, 2007.

[2] J. Axobe, “Fio-flexible io tester,” URL http://freecode.com/projects/fio, 2014.

[3] D. H. Kang, et al., "iDiscard: Enhanced Discard() Scheme for Flash Storage Devices," 2018 IEEE International Conference on Big Data and Smart Computing (BigComp), Shanghai, China, 2018, pp. 360-366, doi: 10.1109/BigComp.2018.00060.

[4] Y. Liang et al., "iTRIM: I/O-Aware TRIM for Improving User Experience on Mobile Devices," in IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, vol. 40, no. 9, pp. 1782-1795, Sept. 2021, doi: 10.1109/TCAD.2020.3027656.

[5] Daisyplus, http://crztech.iptime.org:8080/Release/OpenSSD/DaisyPlus-OpenSSD/