# 플래시 스토리지 기반 Ext4 파일시스템의 효율적인 파일 삭제 기법

강동현[O1] 엄영익[2]
[1]성균관대학교 대학원 전자전기컴퓨터공학과
[2]성균관대학교 소프트웨어대학
{kkangsu, yieom}@skku.edu

# An Efficient File Deletion Mechanism of Ext4 Filesystem on Flash-based Storage Systems

Dong Hyun Kang[O1] and Young Ik Eom[2]
[1]Dept. of Electrical and Computer Engineering, Sungkyunkwan University
[2]College of Software, Sungkyunkwan University

**요 약**

Trim 명령어는 파일시스템과 플래시 기반 스토리지의 시맨틱 갭 (Semantic Gap)을 완화시키기 위해 소개되었다. 그러나 기존의 Ext4 파일시스템은 파일시스템 내부의 파일이 삭제될 때, 데이터 블록에 대해서만 Trim 명령어를 스토리지에 전달하는 문제점을 가지고 있다. 이에, 본 논문에서는 Ext4 파일시스템이 메타데이터 블록에 대해서 Trim 명령어를 전달하지 않는 이유에 대해서 알아보고 데이터 블록과 메타데이터 블록 모두에 Trim 명령어를 전달하는 새로운 파일 삭제 기법을 제안한다. 실험 결과, 제안한 기법이 기존 Ext4 파일 시스템에 비해 스토리지의 불필요한 블록의 수를 최대 5.4배 감소시키는 것을 확인하였다.

## 1. 서 론

오늘날 플래시 메모리 기반 스토리지 (예: SSDs, eMMCs, SD Cards)는 다양한 분야에서 사용되고 있으며 빠르게 HDD의 자리를 대체하고 있다. 그러나, 플래시 기반 스토리지는 HDD와 다르게 물리적인 플래시 페이지에 제자리 덮어 쓰기 (In-place Update)를 지원하지 않는 특성을 가지고 있기 때문에 Ext4 파일시스템과 플래시 기반 스토리지 사이에 시맨틱 갭 (Semantic Gap)이 발생할 수 있다[1]. 예를 들어, 만약 사용자가 Ext4 파일시스템의 파일을 삭제한다면, 파일시스템은 *unlink*() 시스템 콜을 호출하여 삭제된 파일의 데이터 블록을 삭제한 후, 해당 데이터 블록을 회수한다. 그러나, 플래시 기반 스토리지는 삭제된 파일의 데이터 블록을 여전히 유효한 (Valid) 블록으로 간주하고 있기 때문에 해당 데이터 블록에 대한 플래시 페이지를 스토리지에 유지한다. 이렇게 파일시스템 삭제 후 남겨진 플래시 페이지들은 플래시 기반 스토리지의 가비지컬렉션 (GC: Garbage Collection) 시점에 유효한 블록으로 간주되어 불필요한 복사를 초래하기 때문에 플래시 기반 스토리지의 성능 및 수명을 감소시키는 원인이 된다.

Trim 명령어는 이러한 성능 및 수명 감소 문제를 해결하기 위해 파일시스템의 파일이 삭제될 때 파일에 속하는 데이터 블록에 대해서 Trim 명령어를 플래시 스토리지에 전달하며 스토리지는 Trim 명령어와 함께 전달된 LBA (Logical Block Address) 블록을 스토리지에서 무효화 (Invalid) 시킨다[1][2]. 그러나, 현재의 Ext4 파일시스템은 Trim 명령어를 삭제된 파일의 데이터 블록에 대해서만 이용함으로써, 파일의 메타데이터 블록 (즉, 아이노드 블록)을 스토리지에 유효한 페이지로 남겨두는 문제점을 가지고 있다.

이에, 본 논문에서는 우선 Ex4 파일시스템에서 메타데이터 블록에 대해서 Trim 명령어를 전달하지 않는 이유에 대해서 살펴본다[3][4]. 그리고 Ext4 파일시스템의 파일이 삭제될 때 해당 파일의 데이터 블록과 메타데이터 블록에 대해 모두 Trim 명령어를 전달하는 새로운 파일 삭제 기법을 제안한다. 또한, 본 논문에서는 플래시 기반 스토리지의 내부를 동작을 직접 확인하기 위해 DiskSim 시뮬레이터[5]을 수정하였으며 이를 기반으로 제안된 기법의 평가를 수행하였다. 실험 결과, 제안 기법이 기존 Ext4 에 비해 파일 삭제로 인해 발생하는 불필요한 블록의 수를 최대 5.4배 감소시키는 것을 확인하였다.

## 2. Ext4 파일시스템

Ext4 파일시스템은 모바일 환경에서부터 서버 환경까지 널리 사용되고 있는 가장 대표적인 파일시스템 중에 하나이다. Ext4 파일시스템은 전체

본 연구는 미래창조과학부 및 정보통신기술연구진흥센터의 정보통신·방송 연구개발사업의 일환으로 수행하였음. [R0126-15-1065 , (SW 스타랩) 중대형 디스플레이 기반 동시 다중 사용자 지원 UX 플랫폼 SW 개발]

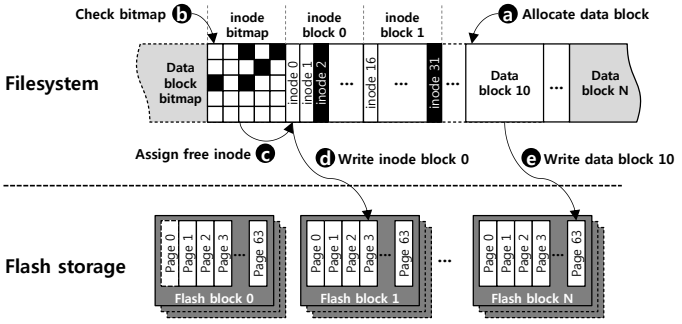


그림 1 Ext4 파일시스템의 파일 생성 및 삭제 과정

스토리지의 공간을 N개의 블록 그룹으로 나누고 있으며 하나의 블록 그룹은 데이터 블록 비트맵 (1 Block), 아이노드 비트맵 (1 Block), 아이노드 테이블 (m Blocks), 데이터 블록 (n Blocks) 등으로 구성된다[4]. 2개의 비트맵 블록은 아이노드 블록과 데이터 블록의 사용 여부를 비트 (Bit) 형태로 저장하고 있으며 아이노드 테이블의 각각의 아이노드 블록 (4KB)은 256KB의 아이노드 16개를 포함하고 있다[2].

그림 1은 Ext4 파일시스템 상에서 4KB의 파일을 생성하고 삭제하는 과정을 자세하게 보여준다. 만약, 사용자가 4KB 크기의 새로운 파일을 생성한다면, Ext4는 우선 데이터 블록 비트맵을 기반으로 사용 가능한 데이터 블록 (Free Block)을 찾는다 (그림 1a). 또한, Ex4는 아이노드 비트맵 정보를 확인하여 사용 가능한 아이노드 (Free Inode)를 찾고 해당 아이노드에 새로 생성된 파일의 정보를 저장한다 (그림 1b-c). 마지막으로, Ext4 파일시스템은 커널 이벤트에 의해서 생성된 파일에 할당된 데이터 블록과 아이노드가 포함된 아이노드 블록을 동시에 플래시 스토리지로 반영한다 (그림 1d-e). 그러나, 사용자에 의해 생성된 4KB의 파일이 삭제되는 경우, 기존의 Ext4 파일시스템은 데이터 블록에 대해서만 Trim 명령어를 플래시 기반 스토리지로 전송함으로써, 삭제된 파일의 아이노드가 포함된 아이노드 테이블의 블록은 스토리지에 그대로 유지시킨다. 이러한 이유는 Ext4의 아이노드를 포함하고 있는 아이노드 테이블의 블록들은 16개의 아이노드를 포함하고 있으며, 삭제된 파일과 동일한 아이노드 블록에 포함된 아이노드 중 일부는 파일의 유효한 메타데이터 정보를 포함하고 있을 수 있기 때문이다. 그러나, 현재의 Ext4 파일시스템은 동일한 아이노드 블록에 속하는 16개의 아이노드가 모두 무효한 메타데이터 정보를 포함하고 있는 경우에도 해당 아이노드 블록에 대한 Trim 명령어를 스토리지에 전달하지 않음으로써 무효화(Invalid)된 아이노드 블록 정보를 스토리지에 그대로 유지한다.

이러한 Ext4 파일시스템과 플래시 기반 스토리지의 시맨틱 갭을 정확하게 확인하기 위해 우리는 DiskSim 시뮬레이터를 수정하였으며 FIO Benchmark을 이용하여 Ext4 파일시스템의 블록 I/O 요청 (I/O Trace)을 수집

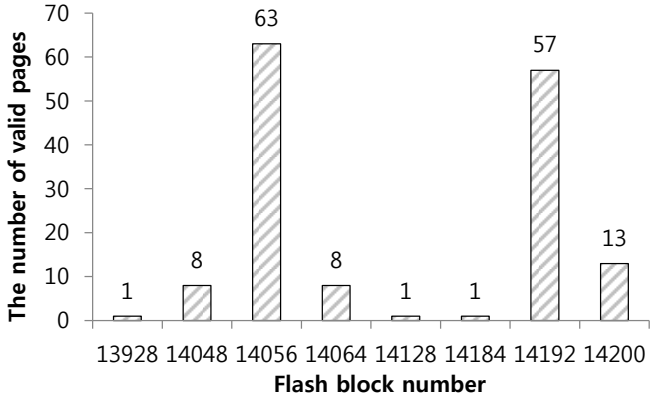


그림 2 스토리지 내부의 유효한 (Valid) 페이지 개수

하였다. 마지막으로, 수집된 I/O 요청을 DiskSim 시뮬레이터[5]의 입력으로 사용하였다. 그림 2는 FIO Benchmark[6]을 이용하여 2000개의 파일을 생성하고 삭제한 후, 플래시 스토리지에 남겨진 플래시 (즉, 유효하다고 간주되는 플래시 페이지) 페이지 개수를 보여준다. 그림 2에서 보여주듯이, 파일이 모두 삭제된 이후에도 스토리지는 152개의 플래시 페이지가 존재하는 것을 확인할 수 있다. 물론, 일부는 파일시스템 자체의 메타데이터를 위해 필요한 플래시 페이지이다. 하지만 대부분의 남겨진 플래시 페이지들은 이미 삭제된 파일의 메타데이터 정보를 포함하고 있는 불필요한 아이노드 블록들이며, 이렇게 남겨진 플래시 페이지들은 플래시 기반 스토리지의 가비지컬렉션 (GC: Garbage Collection) 비용을 증가시킴으로써 스토리지의 전반적인 성능 및 수명을 감소시키는 주요한 원인이 된다.

## 3. 제안 기법

본 논문에서 제안하는 기법은 Ext4 파일시스템에서 아이노드 블록들에 대한 Trim 명령어를 지원하기 위해 기존 Ext4 파일시스템의 구조 (Layout)을 최대한 활용하였다.

사용자에 의해 파일이 삭제될 때, Ext4 파일시스템은 데이터 블록과 삭제된 파일의 아이노드를 모두 회수한다. 그리고 해당 블록과 아이노드를 재사용하기 위해 데이터와 아이노드 비트맵 정보를 '1'에서 '0'으로 변경시킨다. 이에, 제안 기법은 Ext4 파일시스템의 파일이 삭제되어 아이노드 비트맵 정보가 '0'으로 변경될 때, 해당 아이노드와 인접한 16개의 아이노드 비트맵 정보를 확인한다. 즉, 회수되는 아이노드와 동일한 아이노드 블록에 속하는 아이노드 16개의 비트맵 정보를 확인한다. 만약, 하나의 아이노드 블록에 속하는 16개의 아이노드가 모두 '0'인 상태를 가지고 있다면 모든 아이노드가 유효한(Valid) 메타데이터 정보를 포함하고 있지 않으므로 제안 기법은 해당

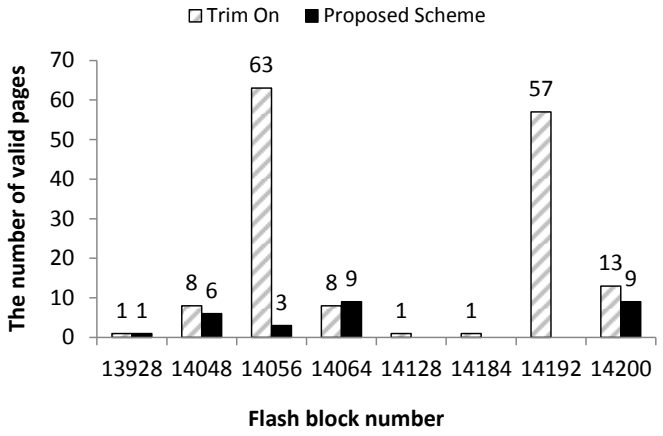


**그림 3 Ext4 파일시스템의 Trim 명령과 제안 기법의 유효한 페이지 개수 비교 결과**

아이노드 블록에 대해 Trim 명령어를 플래시 스토리지에 전달한다. 만약, 16개의 비트 중에서 '1'인 비트가 존재한다면 제안 기법은 기존 Ext4 파일시스템과 동일하게 Trim 명령어를 스토리지로 전달하지 않는다. 예를 들어, 그림 1에서 31번 아이노드에 해당되는 파일이 삭제되는 경우, 16개의 비트맵 정보가 모두 '0'이기 때문에 제안 기법은 31번 아이노드가 속한 블록에 대해서 Trim 명령어를 스토리지로 전달한다. 반면에, 2번 아이노드에 해당되는 파일이 삭제되는 경우, 3번, 5번, 9번 11번 13번 아이노드에 해당하는 비트가 '1'이기 때문에 제안 기법은 Trim 명령어를 스토리지로 전달하지 않는다.

## 3. 성능 평가

본 논문에서는 제안 기법의 성능 평가를 위해 리눅스 커널의 (버전 3.18.25) Ext4 파일시스템을 수정하였으며 Trim 명령어를 지원하기 위해 DiskSim 시뮬레이터 [5]를 수정하였다. 파일 생성 및 삭제를 동일한 환경에서 재현하기 위해 FIO Benchmark[6]을 사용하였으며 2000개의 4KB 파일을 생성 및 삭제하는 동안의 I/O 요청을 blktrace을 이용하여 수집하였다. 마지막으로 수집된 I/O 요청을 수정된 DiskSim의 입력 파일로 사용함으로써, 플래시 기반 스토리지에 남겨진 플래시 페이지들을 직접 확인하였다.

그림 3은 기존 Ext4 파일시스템과 제안 기법을 각각 수행한 후, DiskSim 시뮬레이터에 남겨진 플래시 페이지의 수를 비교하여 보여준다. 그림 3에서 보여주듯이, 제안 기법은 기존 Ext4 파일시스템에 비해 5.4배 플래시 페이지의 수를 감소 시킨다. 이러한 결과는 제안 기법이 효과적으로 Trim 명령어를 플래시 기반 스토리지에 전달하기 때문이다. 특히, 제안 기법은 14128, 13184, 14192 플래시 블록들의 모든 페이지를 Trim 명령어로 무효화시킴으로써 가비지컬렉션 없이도 해당 플래시 페이지들을 재사용할 수 있도록 도와준다. 이러한 실험 결과는 제안 기법이 플래시 기반 스토리지의 가비지컬렉션(GC: Garbage Collection) 비용을 감소시키고 스토리지의 전반적인 성능 및 수명을 향상시킬 수 있다는 사실을 분명하게 보여준다.

## 4. 결론 및 향후 계획

본 논문에서는 현재의 Ext4 파일시스템이 메타데이터 블록에 대해서 Trim 명령어를 지원하지 않는 이유에 대해서 알아보았으며, Ext4 파일시스템에서 메타데이터 블록과 데이터 블록에 Trim 명령어를 모두 전달하기 위한 새로운 파일시스템의 삭제 기법을 제안하였다.

또한, 플래시 기반 스토리지에 남겨진 유효한 플래시 페이지의 개수를 직접 확인하기 위해 DiskSim 시뮬레이터를 수정하여 제안 기법의 성능을 평가하였다. 실험 결과, 제안 기법이 기존 Ext4 파일시스템에 비해 5.4배의 남겨진 플래시 페이지를 감소시킬 수 있다는 사실을 확인하였다. 향후에는 실제 워크로드를 기반으로 제안 기법과 Ext4 파일시스템과의 성능 차이를 확인하기 위한 실험을 진행할 예정이다. 또한, 제안 기법의 성능을 향상시키기 위한 최적화 기법을 연구하고자 한다.

## 참고 문헌

[1] C. Hyun, J. Choi, D. Lee, and S. Noh, "To TRIM or not to TRIM: Judicious Triming for Solid State Drives," *Proc. of ACM Symposium on Operating Systems Principles*, Poster, 2011.

[2] B. Kim, D. H. Kang, C. Min, and Y. I. Eom, "Understanding Implications of Trim, Discard, and Background Command for eMMC Storage Device," *Proc. of IEEE Global Conference on Consumer Electronics*, 2014.

[3] Ext4 [Online]. Available: https://en.wikipedia.org/wiki/Ext4.

[4] Ext4 Layout [Online]. Available: https://ext4.wiki.kernel.org/index.php/Ext4_Disk_Layout

[5] J. S. Bucy, J. Schindler, S. W. Schlosser, and G. Ganger. DiskSim 4.0 [Online]. Available: http://www.pdl.cmu.edu/DiskSim

[6] FIO - Flexible I/O Tester Synthetic Benchmark [Online]. Available: http://git.kernel.dk/?p= fio.git