연속성을 고려한 해시 기반의 플래시 변환 계층

신재민 ○, 정일보, 리샤오창†, 김지홍

서울대학교 컴퓨터공학부

†서북공업대학교 컴퓨터공학부

{jmshin, ilbojeong, jihong}@davinci.snu.ac.kr, †lxcwilliam@gmail.com

Sequentiality Aware hash-based FTL

Jaemin Shin○, Ilbo Jeong, Li Xiaochang, Jihong Kim

Department of Computer Science and Engineering, Seoul National University

†School of Computer Science and Engineering, Northwestern Polytechnical University

요 약

최근 저장장치로서 많이 사용되는 SSD의 용량이 증가하면서 SSD를 내부 데이터 관리를 위한 메타데이터의 크기 또한 증가하였다. 이로 인해 SSD내부 DRAM의 크기도 커지면서 비용적인 측면에서 문제가 발생하게 된다. 그래서 메타데이터의 크기를 줄일 방법이 필요하고 기존에 해시를 통해 이를 해결하는 기법이 제안되었다. 이 기법에서 해시충돌을 막기 위해 둘째 테이블을 추가하여 문제를 해결하고자 하였지만 추가된 테이블을 관리하기 위한 오버헤드가 크다. 이 문제를 해결하기 위해 본 논문에서는 둘째 테이블을 대신하여 새로운 가상블록 테이블을 만들고 논리주소의 연속성을 활용하여 관리 오버헤드를 줄였다. 제안된 기법을 통해 기존에 사용되던 방식의 평균 39% 수준의 메타데이터만을 유지하면서 관리 오버헤드를 최소화하였다.

1. 서론

최근 컴퓨터 시스템에서는 플래시 메모리로 구성된 SSD(Solid State Drive)가 기존의 Hard Disk Drive (HDD)를 대체할 수 있는 저장장치로 주목 받고 있다. 또한 최근 용량 대비 가격도 점차 줄어들면서 더 많이 활용되고 있다. 하지만 SSD는 물리적인 특성으로 인해 HDD와는 다르게 플래시 메모리에 덮어 쓰기(overwrite)에 대한 제약이 존재한다. 그래서 논리 주소에 대한 업데이트가 발생할 시에 가용 가능한 공간에 새롭게 써야 한다. 이러한 특성으로 인해 SSD는 플래시 변환 계층(Flash Translation Layer, FTL)이라는 소프트웨어 계층을 두어 논리 주소에 대한 물리 주소 맵핑을 위한 테이블을 관리한다. 보통의 SSD는 페이지 단위로 맵핑을 관리 및 유지한다.

SSD는 플래시메모리를 관리하는 컨트롤러 및 메타데이터 유지를 위해 내부 DRAM을 활용한다. 주소 변환 테이블도 여기에 유지하며 이를 통해 빠른 주소 변환을 제공해준다. 테이블의 엔트리 크기는 페이지의 크기가 4 KB라고 할 때 4 byte이다. 즉, 전체 페이지 맵핑 정보를 유지하려면 SSD 용량의 0.1%의 DRAM의 공간이 필요하다. 예를 들어 1 TB의 SSD를 사용할 때 1 GB의 DRAM이 필요하게 된다.

DRAM을 활용하여 맵핑 정보를 관리하게 되면 SSD의 크기가 커져가면서 더 큰 용량의 DRAM을 필요하게 된다. 상대적으로 가격이 비싼 DRAM을 플래시 메모리의 주소 맵핑 정보만을 위해 확장한다는 것은 비용적인 측면에서 낭비가 된다. 따라서 DRAM에 유지하는 맵핑 정보의 크기를 줄일 필요가 있다.

DRAM에 유지하는 맵핑 정보의 크기를 줄이는 방법은 크게 두 가지가 존재한다. 첫 번째는 테이블의 엔트리 개수를 줄이는 방법이다. 이를 위해서 맵핑 정보를 캐싱해서 일부만 DRAM에 유지하고 전체정보는 SSD에 저장하는 방법이 있다. [1, 2] 두 번째 방법은 한 맵핑 정보의 크기 즉, 한 엔트리 크기를 4 byte보다 작게 해서 줄이는 것이다. [3] 첫 번째 방법의 경우 캐싱 정도를 조절하면서 맵핑 정보의 크기를 필요한 만큼 유지할 수 있는 장점이 있다. 하지만 맵핑 정보가 DRAM에 없고 SSD에 존재하는 경우 접근에 걸리는 시간이 증가하고 변경된 맵핑 정보를 플래시 메모리에 저장해야 하는 오버헤드가 생긴다. 두 번째 방법의 경우 전체 맵핑 정보를 DRAM에 유지하고 있기 때문에 추가적인 쓰기가 없거나 매우 적고 맵핑 정보에 접근하는 시간이 적게 걸린다. 그러나 한 엔트리 크기를 줄여서 저장하려는 정보의 크기를 최적화하는 방식이기 때문에 저장하는 위치에 제약이 걸리게 된다.

본 논문에서는 맵핑 정보의 크기를 줄이는 방향으로 진행하기 위해 해시를 기반으로 구현하였다. 논문의 구성은 이어지는 2장에서 제안하는 기법인 연속성을 고려한 해시 기반의 FTL에 대해서 설명한다. 그 다음에 3장에서 제안하는 기법과 페이지 단위의 맵핑을 유지하는 기법과의 오버헤드를 비교한다. 마지막으로 4장에서 결론을 맺는다.

2. 해시 기반의 FTL

2.1 HPFTL

본 논문은 해시 기반의 FTL로 제안된 논문인 HPFTL의 [3] 아이디어를 기초로 한다. HPFTL은 첫째 테이블와 둘째

이 논문은 정부(과학기술정보통신부)의 재원으로 한국연구재단의 지원을 받아 수행된 연구임(NRF-2015M3C4A7065645). (교신저자: 김지홍)

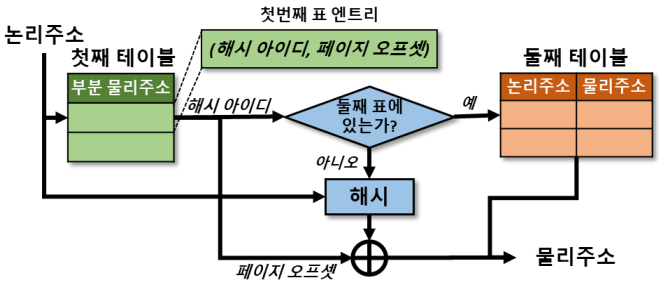


그림 1. HPFTL 기법

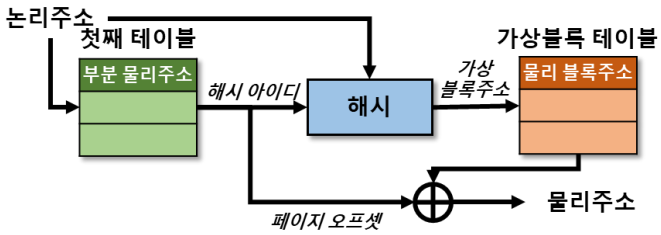


그림 2. 가상블록 기법

테이블로 구성되어 있다. 이중 첫째 테이블을 이용하여 맵핑 정보의 크기를 줄였고 둘째 테이블을 활용하여 해시 충돌을 방지하였다. 기존의 FTL에서는 맵핑 정보를 논리주소를 활용하여 물리주소를 저장하고 있다. HPFTL에서 첫째 테이블은 물리 주소가 아닌 어떤 해시함수를 사용하는지 알려주는 해시 아이디 (Hash ID, HID)와 부분 물리주소 (Partial Page ID, PPID)를 저장한다. HID를 이용하여 SSD내의 블록을 정하는데 0은 맵핑 정보가 둘째 테이블에 있음을 알려주고 나머지 숫자로 해시함수를 가리킨다. 그리고 PPID를 이용하여 블록내의 페이지를 선택한다. 둘째 테이블은 첫째 테이블을 이용하여 저장하려는 위치를 선택하지 못하는 상황 즉, 해시 충돌이 발생했을 때 해당 정보를 저장해주기 위한 공간이다. 이 테이블은 완전연관 (Fully Associative) 방식이기에 논리주소와 물리주소가 같이 저장되어 있다. 이러한 구성에서 HPFTL이 어떤 식으로 맵핑 정보를 찾는지는 그림1을 통해 알 수 있다.

첫째 테이블은 맵핑 정보의 크기를 기존의 값보다 절반 이하로 만들었다. HID의 경우 3~6 bit을 사용하고 PPID의 경우 블록내의 페이지를 가리키기 위해 전체 페이지 개수 혹은 그보다 작은 개수를 가리키기 위해 5~8 bit을 사용한다. 따라서 최대 14 bit으로 2 byte를 넘지 않게 되고 원래 4 byte가 필요하기 때문에 맵핑 정보의 크기를 절반 이하로 줄어든다.

HPFTL에서 둘째 테이블을 사용하는 이유는 크게 2가지이다. 첫째 이유는 FTL이 가비지 컬렉션 (Garbage Collection, GC)를 수행할 때 블록을 선정하고 지우면서 유효 페이지들은 새로운 블록으로 옮겨진다. 이때 옮겨지는 모든 유효 페이지들이 첫째 테이블의 HID를 통해 새로운 블록을 가리킬 수 없기에 둘째 테이블을 이용해 맵핑 정보를 저장해주는 것이다. 둘째 이유는 HPFTL이 블록 내에 갈 수 있는 위치를 제한해서 맵핑 정보를 줄이기 위함이다. SSD에서 쓰기를 수행할 때는 블록 내에서 순서대로 써야 하는 제한이 있기 때문에[4] 해시충돌이 많이 발생하게 되고, 이것을 둘째 테이블에 기록하게 된다.

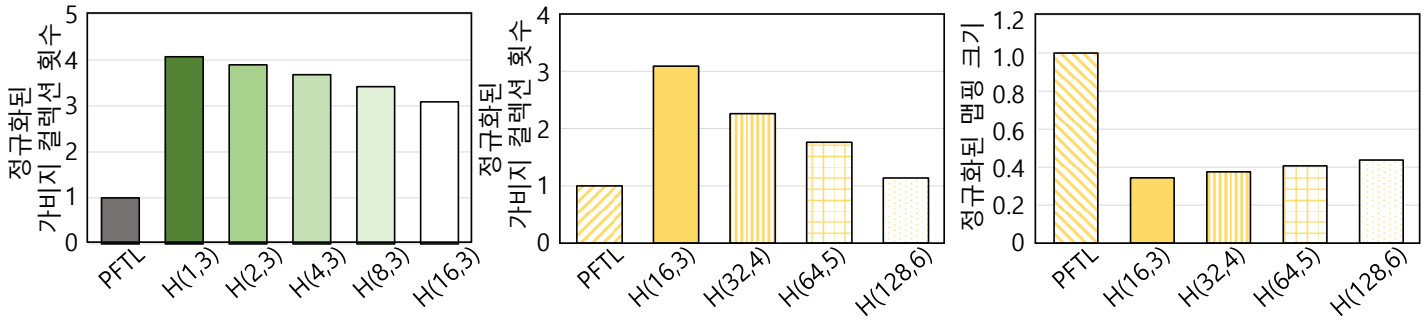
HPFTL은 둘째 테이블의 크기는 모든 맵핑 정보를 담지 못하는 크기라는 문제가 있다. 그래서 둘째 테이블의 빈 공간이 부족하게 되면 둘째 테이블에 있는 맵핑 정보들을 첫째 테이블로 옮겨서 빈 공간을 확보해주어야 한다. 이를 위해서 HPFTL은 GC를 통해 블록 내에 빈 페이지가 생기면 옮길 수 있다고 주장한다. 하지만 GC를 수행하는 과정에서 둘째 테이블에 저장되는 유효 페이지의 맵핑 정보 수가 GC 후 첫째 테이블로 옮길 수 있는 맵핑 정보들의 수보다 항상 작다고 보장할 수 없다. 만약 둘째 테이블의 빈 공간이 확보되지 못하고 오히려 줄어들게 되면 FTL이 멈출 가능성이 매우 높다.

2.2 연속성을 고려한 해시 기반의 FTL

본 논문에서는 둘째 테이블을 사용하게 되는 이유 두 가지를 해결하고 해시를 사용하였을 때 GC 횟수가 증가하는 문제를 해결할 수 있는 페이지의 연속성을 고려한 해시 기반의 FTL (Sequentiality Aware hash based FTL, SEQhFTL)을 제안한다.

둘째 테이블을 사용한 첫째 이유를 해결하기 위해 가상 블록이라는 개념을 사용했다. 첫째 테이블에서 HID를 이용하여 물리블록주소를 가리키는 것을 가상블록주소로 사용하도록 했다. 그리고 가상블록주소를 인덱스로 하는 가상블록 테이블을 추가하여 이 가상블록을 물리블록으로 변환시켜주도록 했다. 변환과정은 그림 2에 나타나있다. 이 개념을 사용하게 되면 둘째 테이블을 사용하는 첫째 이유를 제거할 수 있다. GC를 수행하면 물리블록주소가 바뀌게 되지만, 가상블록 테이블의 맵핑 정보를 업데이트 해주면 블록에 있던 모든 유효페이지들의 가상블록주소를 유지해주면 되기 때문에 둘째 테이블이 필요 없어진다. 둘째 이유의 경우는 블록 내에 가리킬 수 있는 페이지의 위치를 제한하는 경우에 문제가 생기기 때문에 본 논문에서는 블록 내에서는 모든 페이지의 위치를 가리킬 수 있도록 하였다. 가상블록 테이블은 블록단위의 맵핑이어서 상대적으로 페이지 기반 FTL 대비 0.8%의 크기만을 사용하기 때문에 전체 크기 오버헤드에 영향을 끼치지 않는다.

해시를 이용하였을 때 갈 수 있는 블록이 제한적이어서 GC 횟수가 증가하는 문제가 발생한다. 모든 블록에 퍼져서 저장을 하면서 무효화된 페이지 또한 퍼져서 존재하게 되고 이는 GC를 수행할 때 적은 페이지만을 확보하게 된다. 따라서 무효화된 페이지들이 모여있어야 하고 이를 위해서 페이지의 연속성을 고려한 방식을 제안한다. 최근에 RocksDB와 [5] 같이 덧붙여서 기록하는 데이터 관리 방식 (Append-only)을 활용한 응용들이 많이 사용되기 시작하였다. 이러한 응용들에서는 데이터를 연속적으로 쓰기 때문에 SSD에 오는 논리 주소들이 높은 확률로 동시에 무효화된다. 이러한 특성을 고려하여 연속적인 논리주소들이 해시를 사용해도 최대한 같은 블록들을



(a) 연속성만 증가한 경우 (b) HID와 연속성이 증가한 경우 (c) 맵핑 테이블 크기

그림 3. 가비지 컬렉션 동작 횟수 및 맵핑 테이블 크기 분석

선택하도록 해야 한다. 제안하는 기법에서는 해시 함수의 입력으로 논리주소를 넣을 때 원하는 만큼 시프트 하도록 구현하였다. 여기서 논리주소를 시프트해주면 연속된 페이지들이 동일한 해시 값을 사용하게 된다. 따라서 GC를 수행할 때 고른 블록이 무효화된 페이지가 많아지고 GC의 발생 횟수가 줄어들어서 FTL의 관리 오버헤드가 줄어들게 된다.

3. 실험

3.1 실험 환경

FTL동작을 시뮬레이션 할 수 있는 FlashDriver를 [6] 활용하여 기법을 구현하였다. HID의 경우 3~6 bit을 사용하였고 블록내의 페이지 개수가 256개로 PPID는 8 bit을 사용하였다. 연속성의 효과는 시프트해주는 bit의 수를 0에서 7까지 늘려가면서 확인하였다. 시프트해주는 bit의 수는 연속된 페이지수를 의미한다. 실험결과는 기존의 방식대로 맵핑 정보를 유지하는 페이지 FTL (PFTL)과 비교하였다. 기법의 효과를 확인하기 위해서 RocksDB에서 DBbench[7]를 수행한 트레이스를 사용하였다.

3.2 기법 평가

그림 3의 H(X, Y)는 X는 연속된 페이지 수, Y는 HID에 사용한 bit 수를 나타낸다. 그림 3 (a)는 HID를 3 bit으로 고정하고 연속성을 증가시켰을 때 GC 횟수를 PFTL의 횟수로 정규화한 것이다. 그림에서 확인할 수 있듯이 연속성을 증가시키면 GC 횟수가 감소한다. 하지만 PFTL과 비교했을 때 여전히 3배 이상 GC 횟수가 많은 것을 확인할 수 있고 이는 SEQhFTL 자체의 성능 감소로 이어졌다.

추가적인 연속성 증가를 시도하면 논리주소들이 갈 수 있는 위치가 제한되기 때문에 해시 충돌이 발생하였다. 그래서 그림 3 (b)는 이를 방지하기 위해 HID도 동시에 같이 증가시켜 주었을 때의 결과이다. 연속성을 더 증가시킴에 따라 PFTL의 3배정도 수준이던 GC 횟수는 점차 줄어들었고 최종적으로는 PFTL보다 14% 증가한 수준까지 감소하였다.

그림 3 (c)에 HID를 증가시켰을 때 메모리에 유지되는 맵핑 정보 테이블의 크기를 PFTL에 정규화하여 나타내었다. 그림에서 보듯이 SEQhFTL의 맵핑 테이블 크기는 PFTL의 35%~40%정도만을 사용한다. GC 오버헤드를 감소시키기 위해 해시 함수의 개수를 늘렸는데 실제로 HID를 1비트 증가시키는 것은 전체 3%의 추가적인 공간만을 사용하기 때문에 용량에 끼치는 영향은 증가하는 성능 대비 매우 적다.

4. 결론

본 논문에서는 SSD를 위한 변환계층에서 변환을 위한 맵핑 정보 테이블의 크기를 줄일 필요가 있음을 지적하였다. 그리고 이를 해결하기 위해 연속성을 이용한 해시기반의 SEQhFTL을 제안하였다. 맵핑 정보 테이블에 필요한 정보만을 담아서 원래 크기의 절반 미만의 크기로 비슷한 수준의 GC 횟수를 보여주는 것을 성공하였다.

제안한 방식에서는 해시 충돌의 발생 가능성이 매우 낮다. 하지만 분명히 발생 가능성이 존재하기에 충돌을 해결할 방법이 필요하고 이는 향후 선형 탐사방식을 통하여 해결방법을 모색할 것이다.

5. 참고문헌

[1] Aayush Gupta et al., “DFTL: A flash translation layer employing demand-based selective caching of page-level address mappings”, International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS). ACM, 2009

[2] You Zhou et al., "An efficient page-level FTL to optimize address translation in flash memory." Proceedings of the Tenth European Conference on Computer Systems. ACM, 2015

[3] Fan Ni et al., "A Hash-based Space-efficient Page-level FTL for Large-capacity SSDs." 2017 International Conference on Networking, Architecture, and Storage (NAS). IEEE, 2017

[4] “Micron technical report (tn-29-07): Small-block vs. large-block nand flash devices.”

[5] RocksDB, Available on: https://github.com/facebook/rocksdb/

[6] FlashDriver, Available on: https://github.com/dgist-datalab/FlashDriver

[7] DBBench, Available on: https://github.com/memsql/dbbench