SSD를 위한 호스트 수준의 주소 변환 기법의 메모리 요구량 개선

*최동수, 신동군
성균관대학교 정보통신공학부
e-mail : echosoul@skku.edu, dongkun@skku.edu

Memory-Efficient Host-Level Address Translation Layer for Solid State Disks

*Dongsoo Choi, Dongkun Shin
School of Information and Communication Engineering
Sungkyunkwan University

Abstract

Flash memory-based SSD is vulnerable to small random write requests. Although the host-level address remapping technique called HARD can improve the random write performance, it requires a large amount of memory space at host computer. In this paper, we propose two memory-efficient host-level address remapping techniques, called mHATL and sHATL. mHATL uses a multi-level address mapping table and sHATL applies the address remapping to only small requests. The proposed techniques show similar performances as well as require much less memory compared to HARD.

I. 서론

낸드 플래시 메모리는 저전력, 비휘발성, 높은 랜덤 접근 성능 때문에 휴대폰, 디지털 카메라, MP3 등의 휴대용 장치에 널리 쓰이고 있다. 또한, 최근에는 Solid State Disk(SSD)라는 형태로 HDD를 대체하기 시작하고 있다. 그러나 플래시 메모리는 쓰기 전 삭제 연산이 필요하고 삭제하는 단위가 쓰기 단위보다 크기 때문에 이러한 복잡한 구조를 관리하기 위한 플래시 변환 계층(Flash Translation Layer, FTL)이 필요하다. FTL은 호스트 시스템으로부터의 논리주소를 플래시 메모리의 물리주소로 변환해 주는 기능과 무효화된 페이지를 재사용 가능하도록 하는 가비지 컬렉션(garbage collection) 기능을 제공하고 있다. SSD나 eMMC와 같이 블록 인터페이스를 제공하는 플래시 메모리 장치는 이러한 FTL을 내장하고 있다.

SSD나 eMMC에서는 성능 향상을 위해 다중 채널(multi-channel)이나 다중 웨이(multi-way) 구조를 사용하여 여러 개의 칩에 병렬적으로 동시에 접근함으로서 높은 I/O 대역폭을 제공한다. 작은 크기의 낸드 칩에서는 FTL의 주소변환이 페이지 단위로 이루어지는 경우가 많지만, 대용량의 SSD에서는 주소 변환을 위한 메모리 공간을 줄이기 위해서 동시에 쓰기가 가능한 여러 칩을 묶어 슈퍼페이지 단위로 매핑을 관리하는 경우가 많다. 슈퍼페이지 매핑에서는 슈퍼페이지내의 일부만을 변경하는 작은 랜덤 쓰기 요청을 처리하기 위해서 슈퍼페이지 전체를 읽어서 일부만 변경한 후에 다른 슈퍼페이지에 기록하는 read-modify-write 연산이 필요하게 되어 성능 저하가 심하다. 또한, 이러한 작업은 결국 빈번한 가비지 컬렉션을 유도하게 되어 성능을 더욱 저하시킨다.

이러한 문제를 해결하기 위해서, HARD(Host-level Address Remapping Driver)[1]에서는 호스트 시스템에서 작은 랜덤 쓰기 요청들을 모아 슈퍼페이지 단위의 요청으로 변환하는 기법을 제안했다. HARD는 운영체제의 블록 디바이스 드라이버에 추가되어 파일

시스템이 보내주는 입출력 요청의 논리 주소를 가상 주소로 변경하게 되고, SSD의 FTL은 가상주소를 전달받아서 플래시 메모리의 물리주소로 변환한다. HARD가 가상 주소를 연속된 주소로 할당하고 슈퍼페이지 크기로 모아서 SSD에게 전송하므로, SSD에서는 순차적으로 read-modify-write 연산이 없이 빠르게 데이터를 기록할 수 있다. 이러한 작업을 하기 위해서, HARD는 가상 슈퍼페이지(VSP) 주소를 할당하게 되는데, VSP 주소공간이 부족하게 되면, 무효화된 페이지를 많이 포함하고 있는 VSP를 재사용하기 위해 유효한 페이지를 다른 VSP로 옮기는 VGC(Virtual Garbage Collection)를 수행한다.

HARD는 랜덤 쓰기 요청의 성능을 획기적으로 개선시키지만, 호스트에서 관리해야 할 매핑 테이블의 크기가 매우 크다는 단점을 가지고 있다. 파일 시스템이 보내주는 입출력 요청의 기본 단위인 섹터 단위로 주소를 변환해야 하기 때문이다.

따라서 본 논문에서는 HARD를 개선한 HATL (Host-level Address Translation Layer)를 제안하는데, HARD의 단점인 매핑 테이블용 메모리공간을 줄이기 위한 기법으로 mHATL과 sHATL을 제안한다. mHATL은 다중 수준의 주소 변환 기법을 사용하고, sHATL은 일부의 쓰기 요청에 대해서만 선택적으로 가상주소로 변환하는 기법을 사용하여 매핑정보를 위한 메모리 공간을 크게 줄일 수 있다.

본 논문의 구성은 다음과 같다. 2장에서는 본 연구에서 참고한 다중 주소 변환 기법 및 ReSSD라는 기법에 대해서 소개한다. 3장과 4장에서는 제안하는 mHATL과 sHATL에 대해서 자세히 설명한다. 그리고 5장에서는 SSD 시뮬레이터를 이용한 실험 결과를 제시하고, 6장에서 결론을 맺는다.

II. 관련 연구

2.1 다중 수준 주소 변환 기법

다중 수준 주소 변환 기법[2]은 SSD FTL의 주소변환 테이블의 크기를 줄이기 위해서 제안된 기법으로 2^N개의 블럭 단위의 주소 변환을 제공하여 워크로드에 따라 다른 크기의 주소 변환을 사용함으로서 큰 성능 저하 없이 매핑 테이블 크기를 줄이는 기법이다. 본 논문에서는 2^N개의 섹터 단위의 다중 수준 주소 변환을 HARD에 적용한 mHATL을 제안한다.

2.2 ReSSD

ReSSD[3]는 HARD와 유사한 접근법으로서, SSD를 normal area와 reserve area로 나누고, 쓰기 패턴 인식기가 작은 크기의 랜덤 데이터를 reserve area에 기록한 후에 reserve area가 다 차면, 다른 쓰기 요청과 병합하여 normal area로 옮기는 작업을 한다.

본 논문에서는 ReSSD와 같이 작은 크기의 랜덤 데이터만을 가상 주소로 변환하여 SSD의 hidden 영역에 기록하는 sHATL을 제안한다. sHATL은 ReSSD를 device-aware하게 보완하여 호스트가 SSD의 매핑단위를 우선 측정한 뒤에, 슈퍼페이지보다 작은 랜덤 요청만 hidden 영역에 기록한다. 또한, 갑작스러운 전원 차단 시 호스트가 관리하는 매핑 정보가 사라지는 문제를 보완하기 위해서, 호스트가 관리하는 매핑 테이블을 주기적으로 플래시 메모리에 기록함으로서 전원 차단 후에 복구가 빠르게 되도록 개선하였다. 그리고 hidden 영역의 저장 공간이 부족할 때, user area 영역으로 옮겨질 victim 가상 슈퍼페이지를 선정하는 정책에 대해서 제안하고 있다.

III. mHATL (Multi-level HATL)

mHATL에서는 논리주소와 가상주소간의 매핑을 2^N개의 섹터 단위의 멀티레벨로 관리를 한다. 큰 크기의 쓰기 요청은 큰 단위의 매핑을 사용하며, 작은 크기의 쓰기 요청은 작은 단위의 매핑으로 관리를 한다. 제공되는 최소 매핑 단위는 한 섹터이며, 최대 크기는 SSD FTL의 매핑 단위인 슈퍼페이지와 동일하다. 예를 들어, 그림 1과 같이 1개의 슈퍼페이지가 4개의 섹터로 구성이 되어 있는 구조를 가정하면, 제공되는 매핑 단위는 4개, 2개, 1개의 섹터가 되며, 각 단위별 매핑 테이블을 유지한다.

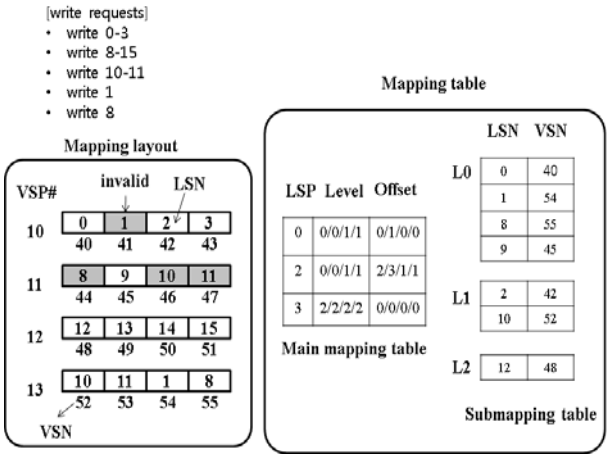

그림 1. mHATL의 매핑 관리 기법

그림 1은 여러 개의 쓰기 요청들에 대해서 mHATL이 주소변환을 한 뒤에, VSP의 구조와 관련된 매핑 테이블의 모습을 보여주고 있다. 해당 쓰기 요청들은

mHATL에 의해서 VSP 10번부터 13번까지를 할당받았으며, 매핑 테이블은 논리 섹터 번호(Logical Sector Number, LSN)에 대한 가상 주소 상의 가상 섹터 번호(Virtual Sector Numbe, VSN)를 저장하고 있다. 매핑 테이블은 크게 두 개로 구성이 된다. 하나는 논리 섹터별 사용하는 매핑 레벨(level)과 해당 매핑레벨 테이블에서의 위치(offset)를 가지고 있는 메인 테이블이며, 나머지 하나는 각 매핑 레벨별로 주소 변환 정보를 가지고 있는 서브 매핑 테이블이다. 그림 1에서 L0는 2^0개의 섹터별 매핑을 관리하는 테이블이며 L1은 2^1개의 섹터, L2는 2^2개의 섹터를 묶어서 관리하는 테이블이다.

예를 들어, LSP 0에 포함되는 연속된 논리 섹터들인 LSN 2와 LSN 3은 mHATL에 의해서 VSP 10의 3번째와 4번째 섹터들인 VSN 42와 VSN 43으로 매핑되었고, 메인 매핑 테이블을 보면 LSP 0의 3번째와 4번째 섹터들(LSN 2와 LSN 3)이 사용하는 서브 매핑 테이블 level은 L1이며 offset은 0이므로 L1 테이블로부터 VSN 42를 알 수 있고 거기서부터 연속된 2개의 가상 섹터 번호를 사용함을 알 수 있다.

이러한 멀티 레벨 매핑 테이블을 사용하면, 연속된 구간에 대해서는 섹터단위의 매핑보다 훨씬 적은 메모리가 매핑 정보를 위해서 사용됨으로 메모리 공간을 절약할 수 있다. mHATL의 매핑정보는 변경된 내용이 한 개의 슈퍼 페이지 분량이 되며 SSD에 저장함으로서 갑작스런 전원 차단에 대응한다. 단, 파일 시스템에서 synchronous write를 보내거나 fsync 명령어를 보냈을 때는 바로 매핑 정보도 함께 저장한다.

mHATL에서는 쓰기 요청을 여러 번 처리하다보면 낮은 레벨의 매핑이 증가하고, 높은 레벨의 매핑이 감소하여 메모리를 많이 요구하게 될 수 있으므로, 서브 매핑 테이블의 크기가 제한된 메모리 크기보다 커지지 않도록, 낮은 레벨의 매핑으로 관리되는 섹터들을 모아 높은 레벨의 매핑으로 바꾸는 과정이 필요하다. 이를 위해서, 낮은 레벨의 매핑을 많이 사용하면서도 최근에 변경되지 않은 VSP를 victim으로 선정하여 해당 VSP의 모든 섹터들을 모아서 새로운 VSP에 논리적 섹터 번호순으로 기록하는 작업을 하는데 이를 mHATL의 VGC인 mVGC라고 한다.

Ⅳ. sHATL (Selective HATL)

sHATL은 그림 2와 같이 SSD의 저장 공간을 user 영역과 hidden 영역으로 분리하고 파일 시스템에게는 User 영역만 제공한다. 슈퍼페이지 크기 이상의 쓰기 요청들은 논리 주소를 그대로 사용해서 SSD의 user 영역에 저장하며, 슈퍼페이지보다 작은 랜덤 쓰기 요청만 hidden 영역에 가상 주소로 변환해서 기록하는 기법이다. 따라서 sHATL은 hidden 영역에 기록된 섹터에 대해서만 매핑 정보를 관리하므로 매핑 정보를 줄일 수 있다.

슈퍼페이지보다 작은 크기의 쓰기 요청을 걸러내기 위해서는 SSD의 슈퍼 페이지 크기를 알아야 하는데, sHATL은 SSD를 처음 시스템에 탑재할 때, 다양한 크기의 쓰기 요청들을 보내면서 응답시간을 측정하여 해당 SSD의 슈퍼 페이지 크기를 추출해 낸다.

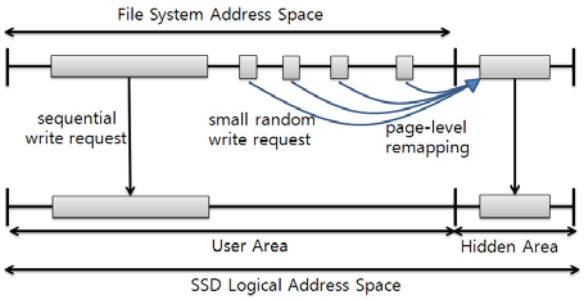

그림 2. sHATL의 구조

sHATL도 hidden 영역의 가상 슈퍼페이지가 소진되면 가상 주소에 대한 GC이 필요하다. 이를 sVGC 라고 하는데, sVGC는 victim으로 선정된 가상 슈퍼페이지의 유효 섹터들을 사용자 영역으로 옮기는 작업을 한다. 이때 해당 유효 섹터들이 포함된 LSP의 모든 섹터들을 읽어 와서 논리적으로 순차적인 슈퍼페이지들로 구성한 후에 User 영역에 쓰게 된다. sVGC에서 섹터 복사를 최대한 줄이기 위해서 해당 VSP와 연관된 LSP들이 가장 적은 VSP를 victim으로 선택한다.

그림 3은 sVGC의 동작을 보여주고 있다. Victim 슈퍼페이지로 관련 LSP가 가장 적은 VSP 0이 선택되었고, 유효 섹터인 LSN 5와 LSN 45를 user 영역으로 옮기기 위해서, LSP 1과 LSP 11에 포함되는 모든 유효 섹터를 호스트의 버퍼로 읽어온다.(그림 3(a)). 호스트에서 LSP 단위로 모아진 데이터는 user 영역으로 한꺼번에 전송되어 SSD내에서 새로운 물리 슈퍼 페이지에 기록된다(그림 3(b)). 데이터의 이동이 끝나면 hidden 영역에서 user 영역으로 복사된 섹터(즉, LSN 5, 45, 46, 47, 7)는 HATL에 의해 가상 주소 상에서 무효화가 되어 새로운 데이터의 가상 주소로 재활용이 되며, user영역의 슈퍼 페이지에서 다른 슈퍼페이지로 복사된 섹터(즉, LSN 4, 6, 44)는 SSD에 의해서 물리 주소 상에서 무효화되어 SSD 내부의 GC에 의해서 재활용된다. sVGC에 의해서 가상 주소 상에서 무효화된 VSN은 매핑 테이블에서 삭제된다.

sHATL의 매핑정보의 저장도 mHATL과 같이 하나

의 슈퍼페이지 분량이 되면 SSD에 저장하게 된다.

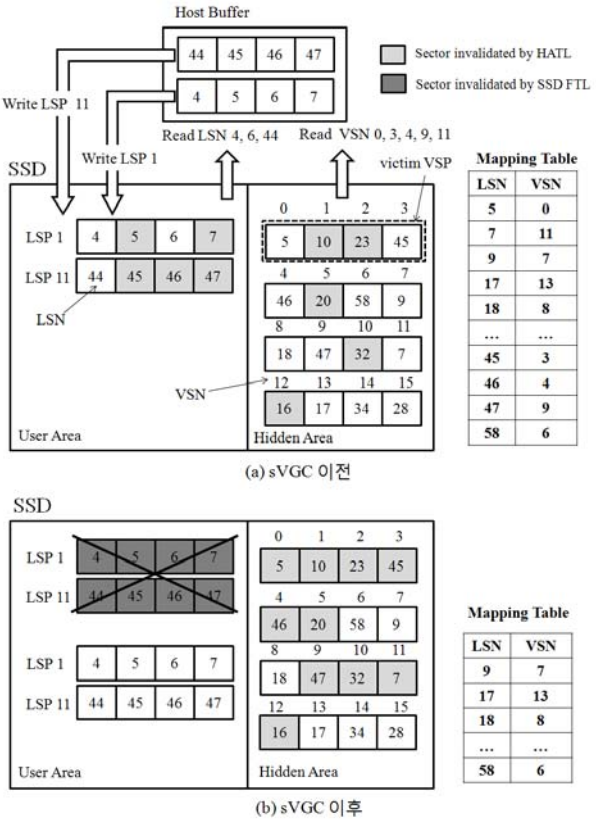

그림 3. sHATL의 sVGC 이전(a), 이후(b)

V. 실험

mHATL과 sHATL이 기존의 HARD에 비해서 사용하는 메모리가 적은 것은 분명한 장점이지만, VGC가 더 자주 발생하기 때문에 성능적인 측면에서는 불리하다. 그래서 본 실험에서는 mHATL과 sHATL이 기존의 HARD 대비 얼마나 성능차가 있는 지 확인해 보았다. 실험은 SSD 시뮬레이터[2]를 이용하였고, 4채널, 4웨이로 16개의 페이지를 하나의 매핑단위로 사용하는 슈퍼페이지 매핑을 사용하도록 설정하였다. 그리고 플래시 메모리에 대한 연산 속도로 읽기는 60us, 쓰기는 800us, 삭제는 1500us로 설정했다. SSD의 용량은 32GB로 설정하여 HARD를 사용 시에 512MB의 메모리가 필요한 환경이며, mHATL의 경우에는 매핑 테이블의 크기를 256KB로 제한하고, sHATL은 hidden영역의 크기를 128MB로 설정하고 실험을 진행하였다.

그림 4의 그래프는 여러 가지 벤치마크 트레이스를 처리하는 동안 SSD가 busy했던 전체 시간을 HATL을 사용하지 않았을 때의 시간에 대해서 정규화하여 보여주고 있다. 전체 걸린 시간을 세 가지 종류로 구분하였는데, 쓰기 요청 데이터를 순수하게 기록하는 데 걸린 시간(real_request)과 SSD 내부 FTL이 read-modify-write를 위해서 copyback 연산에 소모한 시간(copyback), 그리고 HATL의 VGC가 소모한 시간(VGC)이다. HARD와 mHATL은 copyback 연산이 발생하지 않지만, sHATL은 슈퍼 페이지 크기보다 크지만 align되지 않은 요청을 처리하는 과정에서 SSD 내부에서의 copyback이 발생한다. mHATL과 sHATL의 경우, VGC가 HARD보다는 자주 발생하여 HARD에 비해 성능이 떨어지지만, 큰 차이가 없었으며 HATL을 사용하지 않는 경우보다는 성능 향상이 크다는 것을 알 수 있다.

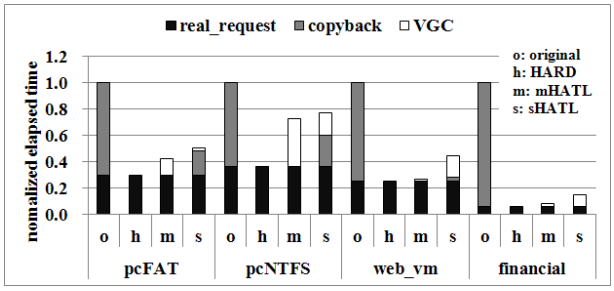

그림 4. HATL의 쓰기 성능

VI. 결론

SSD를 위해서 랜덤 쓰기 요청을 순차 쓰기 요청으로 호스트에서 변환시키는 기존의 HARD 기법이 메모리를 많이 요구하기 때문에 본 논문에서는 매핑 테이블을 멀티레벨로 관리하는 mHATL과 슈퍼페이지보다 작은 랜덤 쓰기 요청만을 주소 변환하는 sHATL을 제안했다. 제안된 두 기법은 메모리 요구량을 많이 감소시키면서도 성능 저하가 크지 않음을 확인했다.

참고문헌

[1] Young-Joon Jang and Dongkun Shin. HARD: Host-Level Address Remapping Driver for Solid-State Disk. International Conference IT Convergence and Security, 2011.

[2] Hyunchul Park and Dongkun Shin. Buffer Flush and Address Mapping Scheme for Flash Memory Solid State Disk. Journal of Systems Architecture, 2010.

[3] Youngjae Lee, Jin-Soo Kim and Seungryoul Maeng. ReSSD: a software layer for resuscitating SSDs from poor small random write performance. In Proceedings of the ACM Symposium on Applied Computing, 2010.