SHRD: SSD의 임의쓰기 성능향상을 위한 로깅 기법

김혁중O, 신동군
성균관대학교 정보통신대학
wangmir@gmail.com, dongkun@skku.edu

SHRD: Sequentializing in Host, Randomizing in Device on SSD

Hyukjoong KimO, Dongkun Shin
College of Information and Communications, Sungkyunkwan University

요 약

SSD에서는 낸드 플래시 메모리의 특성을 가리고 기존 블록 디바이스와 동일하게 사용할 수 있도록 플래시 주소 변환 계층을 사용한다. 최근에는 SSD의 성능을 최대한 활용하기 위하여 페이지 단위의 주소 변환 기법을 많이 사용하는데, 이 경우 주소 변환을 위한 매핑 테이블로 인해 많은 DRAM을 필요로 하게 된다. 매핑 테이블을 낸드 플래시 메모리에서 관리하면서 부분적으로 DRAM에 캐싱하는 디맨드 로딩 기법을 사용할 수 있지만, 임의 접근 시 매핑 테이블을 DRAM으로 로드하는 부하로 인하여 성능이 저하되는 문제가 존재한다. 본 논문에서는 OS 단에서 임의 쓰기를 별도의 예약된 주소 공간에 순차 쓰기로 변환하여 기록한 후, SSD 펌웨어에서 관리하는 매핑 테이블만을 변환하여, 임의 쓰기의 성능을 개선한 SHRD 기법을 제안하며, 이를 실제 SSD에 구현하였다. 실제 SSD에서 실험 결과 전체 맵 크기 대비 1%의 DRAM을 가지고 약 50% 성능을, 20%의 DRAM을 가지고 약 80%의 성능을 낼 수 있었으며, 일반적인 맵 캐싱 기법 대비 최대 13.7배의 성능을 낼 수 있었다.

1. 서 론

낸드 플래시 기반 SSD는 기존 하드디스크 대비 높은 성능과 저전력, 작은 크기와 충격에 강한 장점을 가지고 있으며, 이러한 장점으로 인하여 eMMC (embedded Multi-Media Card)와 같은 포터블 디바이스용 저장장치부터 PC용 solid state drives (SSDs)까지 다양한 분야에서 사용이 증가하고 있다. 특히 이러한 증가세는 기존 하드디스크 대비 높았던 SSD의 비트 당 가격(per bit cost)가 지속적으로 떨어지면서 가속화 되고 있다.

낸드 플래시 메모리는 덮어쓰기가 불가능하고(erase-before-write), 지우기 연산의 단위인 블록의 크기가 읽기 및 쓰기 연산의 단위인 페이지의 크기보다 수십 배 이상 크며, 각 물리 블록마다 정해진 수명이 있는 등의 특성이 있기 때문에, 이러한 특성을 가리고 기존 하드디스크와 동일하게 사용하기 위하여 플래시 변환 계층(Flash Translation Layer, FTL)이라 불리는 소프트웨어를 필요로 한다. FTL은 운영체제에서 요청한 논리 주소를 플래시 메모리의 물리 [1]주소로 변환하여 운영체제가 기존의 블록 디바이스를 사용하는 방식 그대로 플래시 메모리 기반 스토리지를 사용할 수 있도록 한다[1].

FTL 중 페이지 단위 주소변환 기법(Full Page-level Mapping FTL, FPM)[2]은 페이지 단위의 매핑 테이블을 전부 SSD에 내 DRAM에 저장하고, 읽기, 쓰기 연산마다 DRAM에 존재하는 페이지 단위의 맵을 참고하여 주소를 변환하는 기법이다. FPM은 주소를 변환하는 부하를 최소화하고, 높은 성능을 낼 수 있어 SSD에서 많이 사용된다. 하지만 매핑 테이블을 유지하기 위한 DRAM의 크기가 너무 큰 단점이 존재한다. 낸드 플래시 메모리의 원가는 지속적으로 떨어지고 있는 반면 DRAM의 원가는 고정적이기 때문에 이러한 문제는 SSD의 용량이 증가함에 따라서 더욱 커질 것이다.

FPM의 문제를 해결하기 위하여 낸드 플래시 메모리에 맵을 유지하면서 작은 크기의 DRAM에 디맨드 로딩(demand loading)을 하는 DFTL (Demand-loading FTL) 기법[3]을 사용할 수 있다. 하지만, DFTL을 사용하는 경우 임의쓰기(random write) 접근 시 해당하는 맵을 낸드 플래시 메모리에서 DRAM으로 캐싱하고, 이전 맵을 반대로 낸드 플래시 메모리에 쓰는 과정에서 WAF (Write Amplification Factor)가 증가하고, 병렬적인 연산이 불가능하게 되며, 성능이 저하된다. F2FS[4]와 같은 로그기반 파일시스템의 경우 임의쓰기 접근을 파일시스템 레벨에서 로깅하여 순차쓰기(sequential write)로 변환하였지만, 메타데이터의 양이 많아지는 문제와 가비지

[1] 이 논문은 2013년도 정부(교육부)의 재원으로 한국연구재단의 기초연구사업 지원을 받아 수행된 것임(2013R1A1A2A10013598)
이 논문 미래창조과학부 및 정보통신기술진흥센터의 서울어코드활성화지원사업의 연구결과로 수행되었음 (IITP-2015-R0613-15-1062)

컬렉션으로 인한 성능 저하 문제를 해결하지 못하였다.

앞서 언급하였듯이, SSD는 FTL을 사용하여 운영체제에서 요청된 논리 주소를 낸드 플래시 메모리의 물리주소로 변환을 한다. 이러한 특성을 이용하면 맵 테이블의 수정만으로 운영체제의 관점에서 데이터의 위치를 변경하는 것이 가능하다[6]. 본 논문에서는 디바이스 드라이버 단에서 임의쓰기 패턴을 예약된 SSD의 논리 주소 공간으로 로깅하여 순차쓰기 패턴으로 변환(sequentializing)하고, 추후에 맵 데이터만을 수정하여 본래의 논리주소로 변환(randomizing)하는 호스트레벨 직렬화 기법(SHRD, Sequentiaizing in Host, Randomizing in Device)을 제안하며, SHRD 기법을 실제 SSD 디바이스와 리눅스 커널에 구현하였다. 실제 디바이스에서 실험 결과 임의쓰기 패턴에 대하여 DFTL 대비 맵 로딩 부하를 개선하여 최대 13.7배의 성능을 낼 수 있었으며, FPM 대비 20%의 DRAM을 사용하여 FPM의 80%에 가까운 성능을 낼 수 있었다.

2. SHRD 기법

SHRD 기법에서는 SSD의 논리주소 영역 중 일부를 로깅 영역(log area)로 할당하고, 임의쓰기 요청이 들어올 경우 해당 임의쓰기 요청을 로깅 영역에 순차적으로 기록한다. 또한, 로깅 영역에 기록된 임의쓰기 데이터에 대한 사상 테이블(redirection table)을 운영체제에서 유지한다. 이 상태에서 로깅 영역에 저장된 임의쓰기 데이터에 대하여 읽기 요청이 들어오는 경우, 사상 테이블을 참고하여 요청된 데이터의 논리주소를 로깅 영역의 주소로 변경한다. 그리고, 로깅 영역이 일정 수준 이상 채워진 경우, 사상 테이블에 저장된 매핑 데이터를 참고하여 SSD에 리맵(remap) 명령을 보내, SSD에서 로깅 영역으로 매핑된 임의쓰기 데이터를 본래 임의쓰기가 요청된 논리주소로 변환하고, 해당 로깅 영역을 회수한다.

오류! 참조 원본을 찾을 수 없습니다.은 SHRD의 동작 구조에 대한 예시를 보여주는 그림이다. 운영체제에서 블록디바이스로 임의쓰기 요청 4개가 각각 4KB 단위의 논리주소 {32, 512, 142, 33}로 요청되었을 때, SHRD 디바이스 드라이버에서는 해당 임의쓰기 연산들을 모아서 주소를 로깅 영역의 주소로 변환하고, 순차쓰기 명령으로 SSD에 보낸다. 이 때, 기존의 쓰기 명령과는 다른 twrite()란 명령을 사용하게 되는데, twrite는 기존 쓰기 명령에 추가적으로 로깅 영역에 저장되는 각각의 4KB 페이지 별 원본 논리주소를 포함한다. 그 다음 호스트 사상 테이블(Host Redirection Table)에 해당 임의쓰기 데이터의 4KB 페이지 단위 원본 주소(original Logical Page Number, oLPN)와 로깅된 주소(temproal Logical Page Number, tLPN)를 저장한다.

스토리지에서는 요청된 twrite 명령을 처리하여 해당 데이터에 대한 매핑 테이블을 갱신하고, 낸드 플래시 메모리에 데이터를 저장한다. 이 때 SPO (Sudden Power Off) 복구를 위하여 기존 쓰기 명령에서는 낸드 페이지의 스페어 영역에 요청된 논리주소를 저장하는데, twrite 명령의 경우 로깅된 논리주소 대신 twrite 명령에 포함되어 있는 원본 논리주소를 스페어 영역에 저장한다.

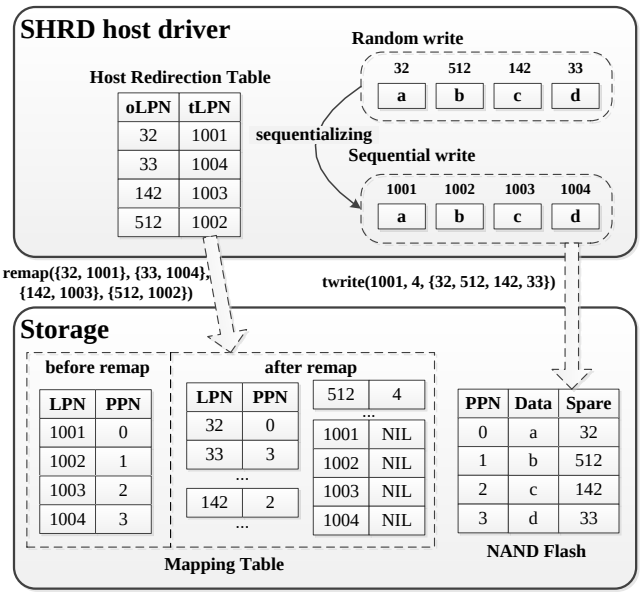

그림 1 SHRD의 구조

리맵 명령 처리 시 디바이스 드라이버는 사상 테이블을 참고하여 회수할 로깅 영역에 해당하는 페이지들에 대한 oLPN과 tLPN의 페어를 행렬 구조로 구성하여 리맵 명령으로 요청한다. 이 때, 행렬은 oLPN을 기준으로 정렬하여 SSD에서 맵 로딩 부하를 줄이도록 한다. 스토리지에서는 해당 리맵 명령의 데이터를 참고하여 tLPN 맵에 저장된 PPN (Physical Page Number)을 oLPN의 PPN으로 변환하는 작업을 한다.

3. 구 현

본 논문에서는 SHRD 기법을 실제 SSD 디바이스의 펌웨어 및 리눅스 커널의 디바이스 드라이버에 구현하였다. SSD는 삼성 서버향 SSD SM843 120GB를 사용하였으며, 리눅스 커널 3.17.4 버전의 SCSI 디바이스 드라이버를 수정하였다.

SHRD를 위해서는 SSD에서 twrite()와 remap()과 같은 특수한 명령을 지원해야 한다. 본 논문에서는 일반 NCQ 쓰기 연산을 그대로 사용하면서, 특정 논리주소 이상(110GB)으로 요청되는 명령을 실제 쓰기 연산이 아닌 특수한 명령으로 펌웨어에서 처리하도록 하였다. 그리고, twrite 명령을 헤더 명령과 데이터 명령으로 분리하여 헤더에는 각 페이지에 대한 원본 주소가 행렬로 저장되고, 데이터에는 실제 로깅 영역에 저장될 데이터를 저장하였다. 데이터 명령은 일반 쓰기 명령으로, 헤더 명령은 110GB 이상의 주소로 요청되는 특수한 명령으로 처리된다. twrite의 헤더 명령과 리맵 명령은 실제로 낸드 플래시 메모리에 저장되지 않고 펌웨어의 메모리에서만 관리한다.

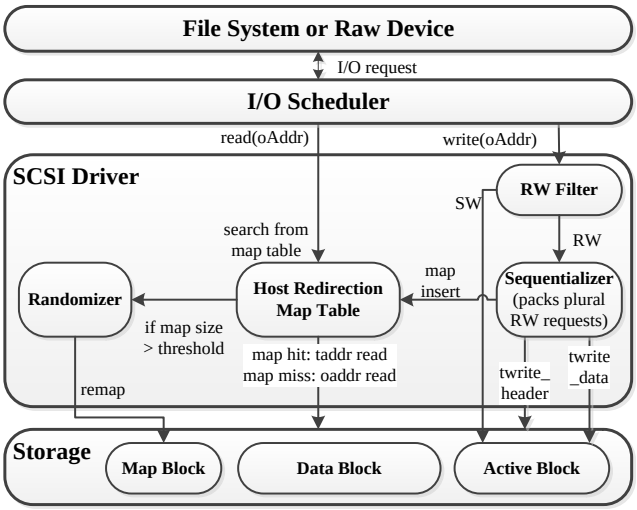

그림 3 SHRD의 구현 및 동작 구조

그림 3는 SHRD의 실제 구현 구조를 보여주는 그림이다. 쓰기 연산이 요청되는 경우, RW Filter에서 해당 쓰기 데이터가 임의쓰기인지를 판단한다. 현재는 연산이 요청된 크기를 통하여 I/O 크기가 작을 경우 임의쓰기로 판단하도록 하였다. Sequentializer에서는 I/O 스케쥴러에서 임의쓰기 패턴의 데이터를 모으고, 하나의 twrite 명령으로 구성하며, twrite 헤더와 데이터 명령을 전송하고, 호스트 사상 테이블을 갱신한다. 읽기 연산이 요청되면 사상 테이블을 탐색하여 로깅 영역에 저장된 데이터인 경우 주소를 변환한다. 리맵 처리시에는 randomizer에서 사상테이블을 참고하여 리맵 명령을 처리한다. 2장에서 언급하였듯이 리맵 시에는 oLPN 기준으로 행렬을 정렬해야 하기 때문에, 정렬 시 오버헤드를 줄이기 위해 호스트 사상 테이블은 oLPN 기준의 RB 트리로 구성하였다.

4. 실험 결과

성능 실험은 실제 SSD 디바이스를 통해 진행되었으며, SSD는 4KB page 단위의 페이지 매핑 FTL을 사용하였다. 맵 캐시(CMT, cached mapping table)의 크기는 128KB부터 전체 맵이 올라가는 FPM까지 변환하면서 실험했다. 호스트 PC는 Intel 2세대 i7-2600 CPU에 8GB DRAM으로 구성되어 있으며, Ubuntu-Gnome 14.04 플랫폼에서 Linux kernel 3.17.4를 사용하였다.

로깅 영역의 크기는 128MB로 SSD 용량(110GB)의 약 0.1%이며, 해당 로깅 영역에 대한 호스트 사상 테이블의 크기를 유지하기 위하여 운영체제에서 추가적으로 사용하는 메모리는 928KB이다. 리맵은 로깅 영역의 반인 64MB까지 데이터가 저장되었을 때 요청하도록 하였고, 리맵 도중 나머지 64MB에 대한 로깅이 병렬적으로 수행될 수 있도록 하였다.

실험은 tiotest를 통하여 16GB의 논리주소 영역에 8GB의 임의쓰기를 4KB의 I/O 크기로 8개의 스레드가 처리하면서 진행하였다. 16GB의 논리주소를 표현하기 위한 전체 맵 테이블의 크기는 16MB이며, CMT의 크기를 그에 약 0.8%인 128KB부터 두 배씩 늘려가며 동일하게 실험했다. **오류! 참조 원본을 찾을 수 없습니다.**은 tiotest 실험에 대한 성능 및 맵 로딩 횟수를 DFTL과 SHRD 기법에 대해 비교한 그래프이다. 실험 결과 SHRD의 성능이 CMT 크기가 128KB일 때 약 13.7배 이상 DFTL보다 좋은 것을 볼 수 있었으며, 이는 임의쓰기 패턴을 직렬화함으로 인한 맵 로딩 부하 감소로 인한 것을 볼 수 있었다. 또한, FPM의 25%인 4MB CMT 크기에서 FPM의 성능의 80%와 유사한 성능을 나타내었다.

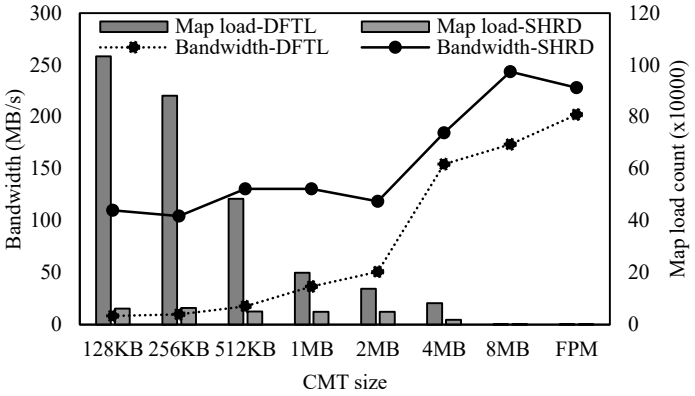

그림 2 tiotest에서 SHRD와 DFTL의 성능 비교

5. 결 론

본 논문에서는 DFTL에서 임의쓰기로 인한 성능저하를 개선하기 위하여 호스트 레벨 직렬화 및 리맵 기법을 제안하였으며, 이를 실제 SSD 및 리눅스 커널에 구현하여 성능 분석을 진행하였다. 실험 결과 직렬화를 통해 맵 로딩 부하를 줄이고 적은 DRAM 크기로도 FPM과 유사한 성능이 나타남을 확인할 수 있었다.

참 고 문 헌

[1] P. Desnoyers, “What systems researchers need to know about NAND flash,” in Proc. USENIX HotStorage’13, 2013.

[2] A. Ban, “Flash filesystem,” United States Patent, No. 5,404,485, 1995.

[3] Gupta et al., “DFTL: a flash translation layer employing demand-based selective caching of page-level address mappings,” in Proc. ASPLOS’09, 2009.

[4] Lee et al., “F2FS: A new file system for flash storage,” FAST’15, 2015.

[5] Lee, Y., Kim, J.-S., AND Maeng, S., “ReSSD: A software layer for resuscitating SSDs from poor small random write performance,” SAC ’10, 2010.

[6] Kang et al., “X-FTL: transactional FTL for SQLite databases,” SIGMOD’13, 2013.