대용량 SSD를 위한 호스트 단 블록 관리 기법

최시훈○, 곽현호, 신동군
성균관대학교 정보통신대학
beswan@skku.edu, gusghrhkr@skku.edu, dongkun@skku.edu

Host-Driven Block Management Scheme for Massive Capacity SSD

Sihoon Choi○, Hyunho Gwak, Dongkun Shin
College of Information and Communication Engineering,
Sungkyunkwan University

요 약

최근 SSD가 고성능 저장 장치로 널리 보급되어 쓰이고 있다. SSD는 out-of-place update하는 특성으로 인해 주소 사상을 수행하는 플래시 변환 계층(FTL)을 탑재하여야 한다. SSD의 저장 공간 지속적으로 증가함에 비례하여 주소 공간의 사상 테이블이 커지므로, SSD가 탑재해야 하는 DRAM 필요량이 증가하는 문제가 있다. 이 문제를 해결하기 위해 사상 테이블의 크기가 작은 블록 단위 사상 기법을 사용하고, 로그 구조 파일 시스템(LFS)을 사용할 수 있으나 각 계층이 중복하여 가비지 콜렉션을 수행하는 문제가 있다. 본 논문에서는 블록 단위 사상 FTL과 LFS를 동기화하여 중복된 가비지 콜렉션을 제거하는 기법과 LFS에서 명시적으로 로그 블록의 합병 대상을 선정할 있는 기법을 제안한다. 제안된 기법을 통해 쓰기 집약적인 동작 I/O 패턴의 서버 환경에서 머지로 인한 플래시 메모리의 수명 소모를 46% 감소시켰으며 대역폭을 평균 2.45배 향상시켰다.

1. 서 론

SSD는 낮은 전력 소모와 높은 성능과 안정성 등의 장점을 가지고 있어 최근 데이터 센터를 비롯한 서버의 대용량 저장장치로 널리 쓰이고 있다. SSD는 높은 대역폭을 내기 위해 여러 개의 플래시 칩들을 탑재하며 호스트 요청을 여러 플래시 칩에 분배하여 동시에 작업을 수행하도록 한다.

플래시 메모리는 erase-before-write 구조로 인하여 in-place로 데이터를 갱신할 수 없는 특징을 가져 주소 사상을 수행하는 FTL이 필요하다. 이러한 FTL을 구동하기 위해 SSD는 주소 사상 테이블을 저장하기 위한 대량의 DRAM을 탑재해야 한다. 향후 SSD의 용량이 지속적으로 증가할 전망으로 사상 테이블의 크기를 줄이는 것이 SSD 개발의 주요한 제약 사항이 되고 있다.

대용량의 SSD에서 적은 량의 DRAM으로 주소 사상을 하기 위한 FTL 기법으로 BAST[1]와 DFTL[2] 등이 있다. BAST는 데이터 영역을 블록 단위로 사상하여 주소 사상 테이블의 크기를 크게 줄일 수 있다. 임의 쓰기를 할 경우 로그 블록 쓰래싱 문제가 발생하게 되는 문제가 있다. DFTL은 전체 주소 사상 테이블 중 일부만을 DRAM에 캐싱하는 방법을 사용한다. DFTL는 임의 읽기 및 쓰기 동작 시 사상 테이블 캐쉬 교체로 인해 성능이 저하된다. BAST는 모든 사상 테이블을 DRAM에 탑재하기 때문에 임의 읽기 시 오버헤드가 발생하지 않는 장점이 있어, 본 논문에서는 BAST를 기반으로 임의 쓰기를 개선하는 기법을 구현하였다.

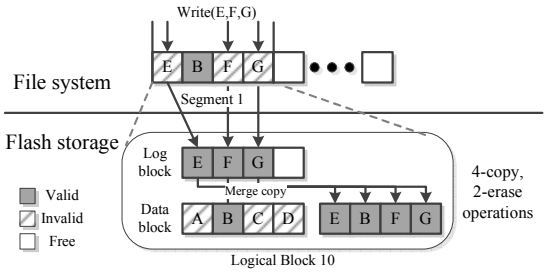
기존의 BAST의 임의 쓰기 성능 저하의 주요 원인은 비싼 로그 블록 합병 비용 때문이다. 특히 로그 블록 합병을 위해 전체 합병을 수행해야 할 경우 합병 할 블록의 모든 페이지를 새 블록으로 복사해야 한다. 합병 대상의 로그 블록의 모든 페이지가 순서를 유지하여 적혀 있다면 전체 합병에 비해 비용이 적은 부분 합병 혹은 교환 합병으로 수행할 수 있다. 이 조건을 만족하려면 로그 블록에 대한 호스트의 쓰기 요청이 순차적으로 이루어져야 한다.

그러나 ext4와 같은 기존의 파일 시스템의 경우 갱신 명령은 기존 파일의 위치에 덮어쓰기 때문에, 갱신 요청의 데이터가 적힐 위치를 임의로 바꿀 수 없다. 만약 어플리케이션이 동일한 파일 위치를 여러 번 갱신하거나 임의 패턴으로 갱신할 경우 BAST에서 전체 합병을 유발한다. 반면에 LFS는 out-of-place 로 갱신을 하는 특성을 가져 모든 쓰기 요청을 파일 시스템이 원하는 위치에 기록되도록 지정할 수 있다. 이 특성을 활용하여 쓰기 요청을 플래시 블록 내에서 주소가 증가하는 순서로만 기록하도록 할 수 있어 로그 블록을 in-place로 갱신하도록 할 수 있다.

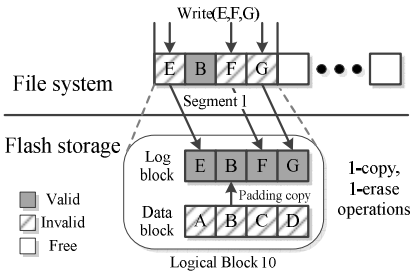
LFS는 SSD와 같이 out-of-place 갱신으로 인해 발생된 무효화된 데이터 영역을 제거하는 가비지 콜렉션 동작을 수행해야 한다. 이러한 LFS의 가비지 콜렉션 동작은 SSD의 가비지 콜렉션 동작과 중첩되어 수행되어 I/O 트래픽을 증가시킨다.[3] 기존의 파일시스템과 플래시 스토리지는 각각 별도로 개발되었으며 서로의 주소 공간에 관여할 수 없기 때문이다. LFS는 무효화된 공간을 빈 공간으로 간주하여 in-place로 갱신하는 스레드 로깅을 사용하여 가비지 콜렉션으로 추가 I/O 트래픽이 발생하지 않도록 할 수 있으나, 임의 쓰기를 발생시키는 문제가 있다.

본 논문에서는 SSD에 BAST의 로그 기반 블록 사상을 기반을 둔 FTL을 사용하여 대용량 SSD의 DRAM 탑재 요구량을 경감하였으며, LFS가 SSD와 중복된 가비지 콜렉션을 제거하기 위해 LFS가 SSD의 주소 공간을 파악할 수 있도록 하여 LFS와 SSD가 독립적으로 수행하던 가비지 콜렉션 동작을 동기화되도록 하였다. 가비지 콜렉션 동기화를 위해 LFS와 SSD의 주소 할당 단위인 세그먼트와 플래시 블록의 크기를 일치시키고 서로 대응[1]시켰으며, LFS는 스레드 로깅방식을 사용하여 가비지 콜렉션을 FTL이 전담하도록 하였다. 그 결과 중복된 가비지 콜렉션과와 전체 합병으로 발생하였던 불필요한 I/O 요청이 제거되어 애플리케이션의 I/O 성능을 향상시키며 플래시 저장 장치의 수명을 연장하는 효과를 얻을 수 있었다.

본 연구는 미래창조과학부 및 정보통신기술진흥센터의 서울어코드활성화지원사업의 연구결과로 수행되었음. (IITP-2015-R0613-15-1062)
본 연구는 미래창조과학부 및 정보통신기술진흥센터의 SW중심대학-ICT/SW창의연구과정 지원사업의 연구결과로 수행되었음. (IITP-2015-R2215-15-1005)

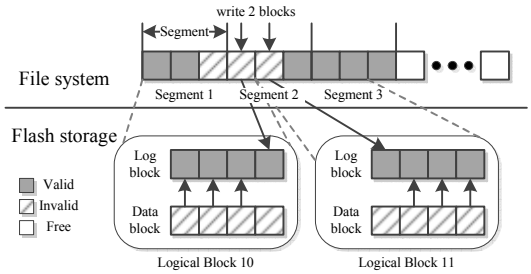


(a) 로그 기반 블록 사상 기법

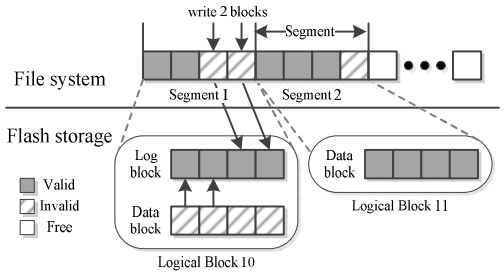


(b) in-place 갱신 로그 기반 블록 사상 기법

그림 1. FTL 동작 비교



(a) 플래시 블록과 세그먼트가 미정렬된 상태



(b) 플래시 블록과 세그먼드가 정렬된 상태

그림 2. 플래시 블록과 세그먼트의
정렬 여부에 따른 FTL 동작 비교

2. 관련 연구

중복된 가비지 콜렉션 문제를 해결할 수 있는 한 가지 방법으로 YAFFS [4], UBIFS와[5] 같은 FTL의 기능을 포함하고 있는 플래시 파일 시스템을(FFS) 사용하는 것이다. 그러나 FFS는 플래시 칩에 대한 저수준의 인터페이스를 포함하기 때문에, 탑재하는 플래시 칩의 구조를 정확히 알아야 구현이 가능하다. 또한 SSD는 개별 플래시 칩들의 낮은 대역폭을 극복하기 위해 여러 칩을 병렬적으로 구성하는 방법을 사용하는데, 이러한 병렬 구조를 정확히 파악할 수 없는 FFS는 SSD의 병렬 구조를 활용하기 어렵다.

REDO는[6] FTL을 블록 사상 기법을 사용하도록 간소화하고, LFS에서 가비지 콜렉션을 전담하도록 하는 기법을 제안했다. 그러나 REDO에서 제안하는 순수 블록 사상 기법은 모든 Page가 반드시 순차적으로 쓰여야 하여, LFS에서 스레드 로깅을 사용하는 환경에서는 많은 가비지 콜렉션을 발생시킨다. 또한 LFS의 쓰기 단위와 플래시 페이지 크기가 일치하지 않을 경우에 대한 고려가 없어 LFS의 세그먼트와 플래시 블록의 주소 공간이 불일치하게 되어 추가적인 가비지 콜렉션을 발생시킨다. 본 논문에서 제안하는 기법은 REDO보다 개선된 LFS와 FTL모델을 제안하여 스레드 로깅의 쓰기 패턴과 다양한 플래시 페이지 크기에 대응할 수 있는 장점을 가진다.

3. 기법 설계

3.1. 블록 사상 기법 FTL 설계

본 기법의 FTL은 로그 기반 블록 단위 사상기법을 사용하며, 로그 블록의 페이지들은 데이터 블록의 페이지 위치와 동일한 위치에 in-place로 기록되도록 하였다. 그 결과, 로그 블록 내의 데이터 위치가 논리 주소에 의해 결정되기 때문에 파일 시스템이 로그 블록 내의 어떤 페이지에 데이터가 기록될지 지정할 수 있게 된다.

out-of-place로 기록할 수 있는 로그 블록을 사용한 블록 단위 사상 기법을 사용하면, 로그 블록의 페이지는 쓰기 요청을 받는 순서대로 순차적으로 기록된다. 그로 인해 그림 1의 (a)와 같이 합병 할 때 로그 블록의 페이지들이 논리 주소에 의해 결정된 페이지 위치로 다시 복사되어야 하는 낭비가 발생한다. 로그 블록을 in-place로 기록하면 쓰기 요청이 발생하지 않은 플래시 페이지는 데이터 블록으로부터 padding하여 복사한다.

그림 1의 (b)를 보면 로그 블록을 in-place로 기록하기 위해 1번의 복사가 필요하지만, 페이지 복사가 필요 없는 교환 합병이 일어나 그림 1의 (a) 대비 3번의 복사와 1번의 지우기 연산을 아낄 수 있다.

기존의 로그 기반 블록 단위 사상 FTL은 합병 대상 로그 블록이 재사용 되지 않을 것이라는 예측을 할 수 있는 근거가 없다. 그 결과, 잘못된 합병 대상을 선정하여 이른 합병과 재 할당을 반복하는 로그 블록 쓰래싱 문제가 발생한다.

이 문제를 해결하기 위해 파일 시스템이 직접 로그 블록의 합병 대상을 선정할 수 있도록 명령을 내려줄 수 있는 인터페이스를 구현하였다. 파일 시스템에서 특정 로그 블록에 해당하는 논리 주소에 더 이상 보낼 쓰기 명령이 없을 경우, 해당 로그 블록을 빨리 합병 하여 빈 로그 블록을 확보할 수 있다. 이러한 기법을 모두 적용한 FTL을 Synchronized-FTL (S-FTL)이라 한다.

3.2. 로그 구조 파일 시스템 설계

본 기법의 LFS는 다음의 특징을 갖는다. 첫 번째, 파일 시스템 세그먼트의 크기는 SSD의 플래시 블록 크기와 일치하도록 설정한다. 두 번째, LFS 세그먼트와 논리 주소로 대응되는 플래시 블록의 데이터 기록 순서가 일치하도록 한다. 세 번째, 클리닝 기법으로 스레드 로깅만을 사용하며, 스레드 로깅을 수행하는 세그먼트 내의 무효화된 공간을 오름차순으로 사용한다. 본 기법에서 제안하는 LFS를 Synchronized-LFS (S-LFS)라 부른다.

그림 2는 LFS의 세그먼트 크기와 플래시 블록의 크기 차이에 따른 동작을 나타낸다. 그림 2의 (a)를 보면 LFS의 세그먼트가 플래시 블록보다 크기가 작은 경우 LFS는 세그먼트 1번을 빈 세그먼트로 간주하고 스레드 로깅을 수행한다. 그러나 세그먼트 1번은 세그먼트 0번과 같은 플래시 블록에 연관되어 있기 때문에 LFS는 FTL 내에서 발생하는 Padding하여 복사하는 비용을 예측할 수 없다. 반면에 그림 2의 (b)의 경우 세그먼트와 플래시 블록의 크기가 동일하여 정렬된 상태로, LFS가 예측하

지 못한 FTL단 Padding 동작이 발생하지 않는다.

LFS가 스레드 로깅 방식으로 쓰기 요청을 보낼 때, 파일 시스템의 쓰기 단위인 블록과 SSD의 쓰기 단위인 플래시 페이지의 크기가 다르면 FTL 내에서는 read-modify-write를 수행하게 된다. 그 결과 순차적으로 쓰이도록 한 FTL의 로그 블록의 사용 규칙과 상술한 LFS의 두 번째 특징을 어기게 되어 LFS에서 FTL의 블록 상태를 파악이 불가능해진다. 이 문제를 해결하기 위해 본 논문에서는 LFS에서 플래시 페이지의 논리 주소 경계를 고려하여 주소를 할당하는 기법을 사용한다. 이 기법은 두 개 이상의 무효화된 블록이 논리 주소 상 동일한 플래시 페이지 단위 내에 흩어져 존재할 경우, 무효화된 블록의 사이의 유효한 블록을 읽어들어 무효화된 블록들에 쓸 데이터와 병합하여 1번의 재기록을 수행한다. 또한 fsync 명령 등으로 이미 논리 주소 상 플래시 페이지 단위에 쓰기 요청을 보낸 경우 해당 플래시 페이지에 해당하는 논리 주소를 사용하지 않는다. 이 기법을 통해 로그 블록에 기록된 플래시 페이지에 대해 갱신 요청이 발생하지 않도록 하여 로그 블록의 페이지 순차 갱신 규칙을 지킬 수 있도록 한다.

4. 실험 및 분석

4.1. 실험 환경 및 구현

실험은 Jasmin OpenSSD Platform을 사용하여 수행하였다. OpenSSD의 FTL로는 본 논문에서 제안하는 S-FTL과 REDO, DFTL을 구현하여 사용하였다. 모든 FTL은 공통적으로 3%의 오버프로비전 영역을 가지며 주소 사상 테이블을 저장하도록 DRAM 16KB를 할당하였다. 내부 병렬성을 활용하기 위한 쓰기 명령의 칩 할당은 논리 주소로 정적 스트라이핑 하는 방식을 사용한다. S-FTL에는 합병 대상 선정 커맨드를 받도록 구현하여 LFS가 로그 블록의 마지막 페이지 주소를 가리키는 명령을 보내면 해당 로그 블록을 합병 하도록 하였다. S-LFS와 REDO는 F2FS[7]를 기반으로 수정하여 구현하였다.

4.2. 실험 결과

실험 시나리오는 tiobench로 파일 시스템의 사용률을 조절한 후에 Filebench의 Varmail 서버 워크로드를 수행하는 것으로 하였다. 그림 3과 같이 제안하는 기법의 대역폭이 DFTL 대비 최대 2.59배 뛰어난 것을 알 수 있다. DFTL과 ext4의 경우 임의 쓰기로 인한 잦은 사상 테이블 교체로 인해 낮은 성능을 보인다. REDO의 성능이 가장 낮은 것은 그림 4와 같이 플래시 크기보다 작은 크기의 쓰기 요청과 fsync 명령으로 인해 전체 합병 동작이 많이 일어나기 때문이다. 반면에 제안하는 기법은 플래시 크기보다 작은 크기의 쓰기 요청에도 로그 블록을 반드시 in-place로 갱신하는 규칙으로 인해 부분 합병과 교환 합병 동작만 발생하게 되어 성능이 뛰어나다. 또한 세그먼트의 사용이 끝났음을 알려주는 명령어로 인해 로그 블록 쓰레싱을 억제하는 효과를 얻을 수 있다.

5. 결론

이 논문에서는 블록 사상 기법 FTL을 위한 로그 구조 파일 시스템을 제안하여, 주소 사상 테이블의 크기를 억제하여 SSD의 DRAM 탑재 요구량이 제한된 환경에서 FTL의 블록 할당과 합병 동작을 파일 시스템이 조절하여 중복된 가비지 콜렉션로 인한 낭비를 최소화하였다. 본 논문에서 제안한 기법의 효과를 측정하기 위해 실제 하드웨어에 대해 실험을 수행한 결과 서버 워크로드 상에서 쓰기 성능과 SSD 수명을 크게 개선하였음을 확인하였다.

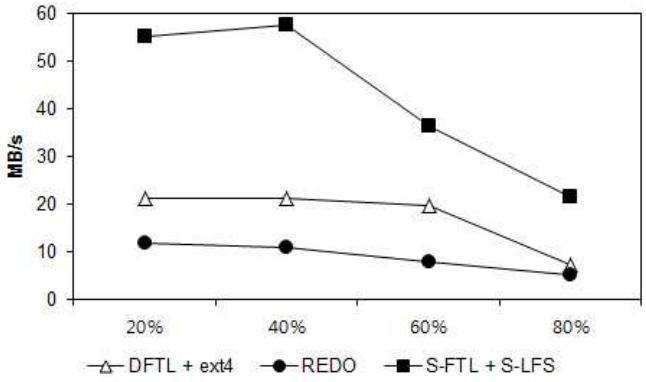


그림 3. Filebench 대역폭

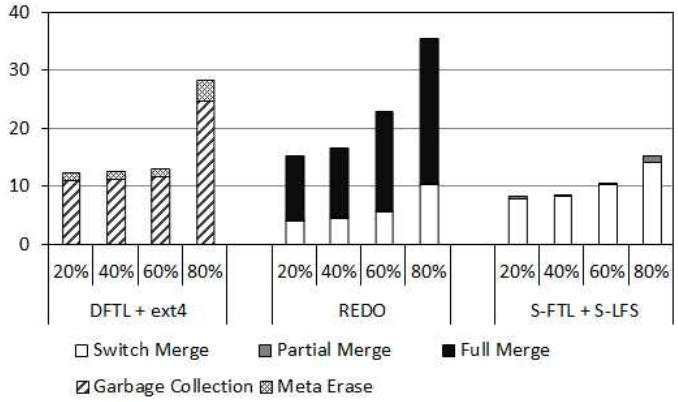


그림 4. Filebench 명령 1000개 수행 시 FTL에서 발생한 블록 지우기 연산

6. 참고문헌

[1] A. Gupta, et al.., "DFTL: a flash translation layer employing demand-based selective caching of page-level address mappings," In Proc. of the USENIX ASPLOS, pp. 229-240, 2009.

[2] J. Kim, et al., "A Space-Efficient Flash Translation Layer for Compactflash Systems," IEEE Trans. on Consumer Electronics, 48(2):366-375, 2002.

[3] Jingpei Yang, et al, "Don't Stack Your Log On My Log," In Proc. of the USENIX INFLOW, 2014

[4] YAFFS2. http://www.yaffs.net/

[5] T. Gleixner, et al., "UBIFS: unsorted block images file system, http://www.linuxmtd.infradead.org/doc/ubifs.html, 2006

[6] S. Lee, et al., "Refactored Design of I/O Architecture for Flash Storage," Computer Architecture Letters, pp. 99:1-1, 2014.

[7] C. Lee, et al., "F2FS: A New File System for Flash Storage," In Proc. of the USENIX FAST, 2015.