Journal of Digital Contents Society Vol. 14 No. 2 Jun. 2013(pp. 215-222)
http://dx.doi.org/10.9728/dcs.2013.14.2.215

SSD를 위한 쓰기 버퍼와 로그 블록의 통합 관리 고려사항

박성민*, 강수용**

요 약

낸드 플래시 기반 SSD는 HDD와 비교하여 많은 장점을 가지고 있다. 하지만 임의 접근 쓰기 요청은 임의 접근 읽기 요청이나 연속 접급 쓰기와 읽기에 비하여 SSD의 접근성을 떨어뜨리고 수명을 단축시키는 문제점을 발생시킨다. 이런 문제점을 해결하기위해 SSD 내부에서는 낸드 플래시 메모리의 일부분을 로그 블록으로 관리하는 기법과 DRAM 혹은 비휘발성 메모리를 쓰기 버퍼로 관리하는 기법들이 제안되었다. 하지만 지금까지 로그 블록 관리와 쓰기 버퍼 관리는 다른 계층에서 연구되어왔다. 즉 로그 블록 관리는 쓰기 버퍼의 상태를 고려하지 않았고 또한 쓰기 버퍼 관리 기법도 로그 블록의 상태를 고려하지 않았다. 본 논문에서는 처음으로 로그 블록과 쓰기 버퍼 사이의 관련성을 통해 두 계층의 통합 관리의 필요성을 제시한다. 그리고 통합된 쓰기 버퍼 설계를 위해 세 가지 고려해야할 사항을 제공한다.

키워드 : 쓰기 버퍼, 로그 블록, 낸드 플래시 메모리

Considerations for Designing an Integrated Write Buffer Management Scheme for NAND-based Solid State Drives

Sungmin Park*, Sooyong Kang**

Abstract

NAND flash memory-based Solid State Drives (SSD) have lots of merits compared to traditional hard disk drives (HDD). However, random write in SSD is still far slower than sequential read/write and random read. There are two independent approaches to resolve this problem: 1) using part of the flash memory blocks as log blocks, and 2) using internal write buffer (DRAM or Non-Volatile RAM) in SSD. While log blocks are managed by the Flash Translation Layer (FTL), write buffer management has been treated separately from FTL. Write buffer management schemes did not use the exact status of log blocks and log block management schemes in FTL did not consider the behavior of write buffer management scheme. In this paper, we first show that log blocks and write buffer have a tight relationship to each other, which necessitates integrated management of both of them. Since log blocks also can be viewed as another type of write buffer, we can manage both of them as an integrated write buffer. Then we provide three design criteria for the integrated write buffer management scheme which can be very useful to SSD firmware designers.

Keywords : Write Buffer, Log Block, NAND Flash Memory

1. 서론

낸드 플래시 기반 Solid State Drive(SSD)는 빠르게 대중적인 저장 매체로서 자리를 잡아가

※교신저자(Corresponding Author): Sooyong Kang
접수일:2013년 05월 02일, 수정일:2013년 06월 17일
완료일:2013년 06월 25일
* 한양대학교 전자컴퓨터통신 공학과
email: syrilo@hanyang.ac.kr
** 한양대학교 컴퓨터공학부 컴퓨터학과
Tel: +82-2-2200-1725
email: sykang@hanyang.ac.kr

▣ 본 연구는 2013년도 정부(교육과학기술부)의 재원으로 한국연구재단의 지원을 받아 수행된 연구임 (No. 2012-047724)

고 있다. 이미 SSD는 노트북 환경에서 하드 디스크를 대체하기 시작했고 데스크톱 환경과 엔터프라이즈 환경까지 그 영역을 확장하고 있다. 이는 SSD가 하드디스크에 비해 빠른 접근 속도, 정숙성, 저전력, 내구성과 같은 많은 장점을 가지고 있기 때문이다. 아직까지는 하드디스크에 비해 바이트 당 비용이 매우 높은 편이지만, 플래시 메모리의 집적도가 높아짐에 따라 머지않은 미래에 SSD가 대부분의 하드 디스크를 대체할 것이라는 예상도 나오고 있다[1].

플래시 메모리는 하드디스크와는 달리 기계적인 움직임이 없이 데이터를 읽어 들이기 때문에 랜덤 읽기 성능이 우수하다. 이런 플래시 메모리의 장점으로 인해 SSD의 랜덤 읽기 성능은 하드디스크에 비하여 20~50배 빠르다[2]. 하지만 플래시 메모리는 하드 디스크와는 달리 덮어쓰기(In-Place update)가 불가능하기 때문에, 데이터를 업데이트를 하기 위해서는 해당 영역을 지운 후 업데이트를 해야 한다. 플래시 메모리의 읽기/쓰기 단위는 페이지 단위 (2KB ~ 4KB) 이고 지우기 단위는 블록 단위 (128KB ~ 512KB) 이므로, 데이터를 update하기 위해서는 원칙적으로 그 데이터가 저장된 블록을 지운 후에, 다시 그 블록에 저장되어 있던 update되지 않은 데이터와 update된 데이터를 함께 써야 한다. 이런 플래시 메모리의 단점이 SSD에 반영되어, SSD에 랜덤 쓰기를 많이 요청하게 되면 내부적으로 유효한(valid) 페이지의 복사와 지우기가 많이 발생하게 된다. 이는 SSD의 심각한 성능 저하와 수명 단축의 요인이 된다.

이러한 쓰기의 문제점을 개선하기 위해서 여러 시스템 계층에서 다양한 연구가 진행되어 왔다. CFLRU[3], WSR[4], ABM[5]은 버퍼에 있는 update 된 데이터를 플래시메모리로 쓰는 작업을 지연시키는 버퍼교체 정책이며, FAB[11]와 REF[13] 는 버퍼에 별도의 쓰기 버퍼 영역을 두고 읽기 버퍼와 별도로 관리하는 쓰기 버퍼 관리 정책을 사용하였다. 파일 시스템 계층에서는 YAFFS[6], JFFS2[7], UBIFS[8] 등과 같이 플래시 메모리를 직접 제어하는 플래시 메모리 전용 파일 시스템이 개발되었고, NILFS, BtrFS 등 일반 파일 시스템에서도 SSD를 고려한 기법들이 연구되고 있다. 호스트 인터페이스 측면에서는 SSD를 위한 새로운 명령어(ex. trim command)를 추가하여, 삭제된 파일들과 같이 더 이상 유효하지 않은 블록에 대한 정보를 SSD에 전달함으로써 유효한 페이지의 복사와 지우기 횟수를 줄이고자 하고 있다. 그러나 이와 같은 방법들은 SSD와 HDD를 함께 사용할 경우 성능 하락이 있을 수 있고(버퍼 관리 정책의 경우), SSD 사용자가 파일시스템을 별도로 설치 및 사용해야 한다는 문제점(SSD 전용 파일시스템의 경우) 등으로 인해 실제 널리 사용되는 데에는 한계가 있을 수 있다. 결국 SSD가 널리 보급되기 위해서는 SSD 내부의 문제를 SSD 내부에서 해결하는 방법이 필요하다.

SSD 내부에서 랜덤 쓰기의 문제점을 해결하기 위해서 두 방향으로 연구가 진행되어 왔다. 첫 번째는 SSD 전체 블록 중 일정 블록을 로그 블록으로 설정하고 update 되는 데이터를 원본 데이터가 저장된 블록 대신 로그 블록에 쓰는 방법이다. 로그 블록에 대한 관리는 플래시 변환 계층(FTL)이 담당하는데, 로그 블록의 수와 FTL에서의 관리 기법에 따라 랜덤 쓰기 성능과 지우기 연산의 횟수가 크게 다르게 된다. 로그 블록을 사용하는 FTL로는 BAST[9], FAST[10] 등이 있다. 다른 하나는 FAB[11]과 BP-LRU[12]와 CLC[14]와 같이 SSD 내부에서 쓰기 버퍼를 사용함으로써 플래시 메모리에의 쓰기 및 지우기 횟수를 줄이는 것이다. 특히 CLC는 power loss 시 쓰기 버퍼에 있는 데이터를 잃게 되는 문제에 대응하기 위해 FeRAM, PRAM 과 같은 비휘발성 메모리(NVRAM)를 쓰기 버퍼로 사용하는 것을 가정하였다.

지금까지 이러한 SSD 내에서의 두 접근 방법들은 서로 분리되어 연구되어 왔기 때문에, 쓰기 버퍼 관리 정책과 로그 블록 관리 정책이 서로의 정보를 이용하여 상호 연계되어 동작하는 기법은 연구되지 않았다. 우리의 이전 연구[13]에서, 우리는 쓰기 버퍼에서 플래시메모리로 요청되는 데이터의 패턴을 고려하여 FTL 이 로그 블록을 관리하게 하는 기법인 Optimistic FTL을 제안하였다. Optimistic FTL은 일반적인 쓰기 버퍼 관리 정책의 동작 원리를 가정하고, 그에 맞추어 로그블록을 관리하는 정책이다. 이와 반대로, REF[13]는 로그블록 관리 정책의 일반적인 동작 방식을 가정하고, 그에 맞추어 어떤 페이지를 교체할 것인지를 결정하는 OS의 쓰기

버퍼 교체정책이다. 이 두 가지 기법은 쓰기 버퍼 관리 정책과 로그 블록 관리 정책 간 상호 인지의 필요성을 제시하였으나, 적극적인 상호 연계 동작이 아니라 각 정책의 동작 '결과' 만을 고려하는 수준에 머무르고 있다. 그러나 쓰기 버퍼와 로그 블록은 모두 SSD 컨트롤러에서 관리하기 때문에 그들의 정보 공유와 상호 연계가 가능하다는 점과, 로그 블록은 넓은 의미에서 볼 때 쓰기 버퍼의 일종으로 볼 수 있다는 점에서, 두 기법간의 상호 연계는 필수적이라 할 수 있다.

2. 연구 배경

2.1 NAND Flash Memory

낸드 플래시 메모리는 크게 small block NAND와 Large block NAND로 구분되며, Large block NAND는 SLC와 MLC로 구분된다. Small block NAND는 주로 임베디드 시스템에서만 사용되고 플래시 디스크(USB, SSD)에서는 더 이상 사용되지 않는다. SLC는 셀 당 1 비트를 저장할 수 있고, MLC는 셀 당 2 비트의 정보를 저장하는 방식이기 때문에 SLC 보다 많은 양의 데이터를 저장할 수 있다. 최근에는 셀 당 3 비트의 정보를 저장하는 TLC도 생산되고 있다. MLC는 SLC보다 Density가 높기 때문에 가격이 저렴한 장점을 가지고 있다. 하지만 쓰기 속도가 SLC에 비하여 3~4 배 느리며 지우기 횟수가 SLC 의 약 1/10 에 불과하다. 하드 디스크와의 가격 경쟁으로 일반 노트북이나 데스크톱 환경에서는 MLC 기반의 SSD가 일반적으로 사용되고 있으며, SLC 기반의 SSD는 주로 서버용으로 사용된다. 본 논문에서 MLC 기반의 SSD를 가정한다.

2.2 플래시 변환 계층(FTL)

플래시 변환 계층(FTL)은 플래시 메모리와 파일 시스템 사이에 위치해서 파일 시스템이 플래시 메모리를 하드 디스크와 같은 블록 디바이스로 에뮬레이션 해주는 계층이다. FTL은 호스트 시스템에 소프트웨어로 구현되거나 디바이스에 직접 탑재되어 구현될 수도 있다. FTL은 블록 주소 매핑, 웨어레벨링, 베드 블록 관리, ECC 체크 등 많은 기능을 담당하고 있다. 그 중, 주소 매핑 방식에 따라서 로그 블록을 관리하는 방법이 다르게 되고 이것은 읽기, 쓰기, 지우기 연산에 큰 영향을 미친다.

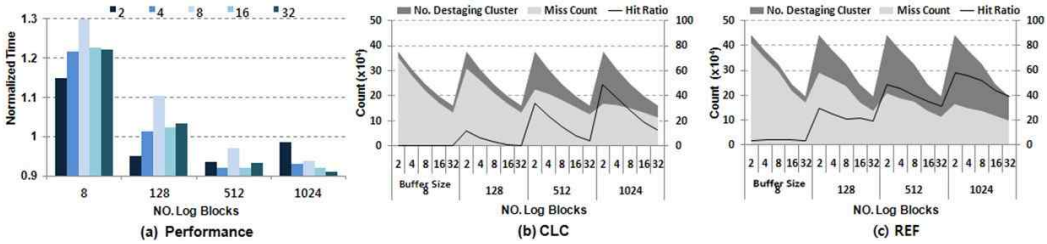
3. 본문: 로그 블록과 쓰기 버퍼사이의 연관성

SSD의 쓰기 버퍼 관리 정책과 로그 블록 관리 정책은 하나의 컨트롤러 내에서 구현됨에도 불구하고 지금까지 그들은 각각 분리되어 설계되었다. 쓰기 버퍼 관리 정책에서는 현재 사용중인 로그 블록의 정보가 없는 상태에서 희생 블록을 선정하였고, 로그 영역 관리 정책에서도 쓰기 버퍼에 저장된 클러스터의 정보를 모르는 상태에서 희생 로그 블록을 선정하여 병합 연산을 하였다. 이로 인해 기존의 쓰기 버퍼 관리 정책들은 워크로드 특성, 쓰기 버퍼의 크기, 그리고 로그 블록의 개수에 따라서 성능의 변화가 매우 클 수 있다. 이 섹션에서는 이 현상을 밝히고 그 원인을 정밀하게 분석하기 위해 워크로드 특성, 로그 블록의 수, 쓰기 버퍼의 크기를 다양하게 변화시켰을 경우 쓰기 버퍼 정책의 성능 변화를 분석하고, 그것을 바탕으로 SSD를 위한 통합 버퍼 관리 정책 설계를 위한 고려해야할 사항을 도출한다.

3.1 로그 블록 개수의 효과

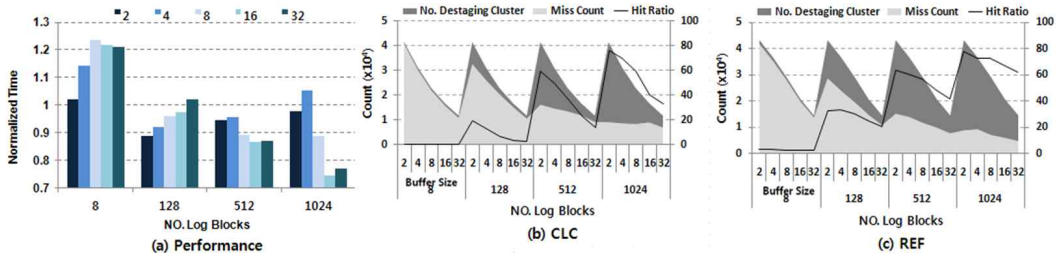
로그 블록의 수가 쓰기 버퍼 관리 정책의 성능에 미치는 영향을 알아보기 위해 우리는 로그 블록의 수와 버퍼의 크기를 변화시키면서 쓰기 버퍼 관리 정책 두 가지(CLC, REF)의 성능 변화를 측정하였다. 로그 블록 관리 기법으로는 BAST를 사용하였다. BAST는 데이터 블록과 로그 블록을 1:1로 매핑하기 때문에, 쓰기 버퍼에서 요청된 클러스터에 해당되는 데이터 블록과 매핑되는 로그 블록이 이미 존재하는 경우(Log block hit)에는 그 로그 블록에 요청 된 페이지들을 기록하게 되고, 그러한 로그 블록이 없는 경우(Log block miss)에는 새로운 로그 블록을 할당 받아서 요청 된 페이지들을 기록해야 한다. 따라서 로그 블록 참조율이 높아질수록 로그 블록의 이용률이 높아지게 되어 전체적으로

(그림 1) 버퍼 관리 정책 성능 분석: FAT 워크로드



(Figure 1) Performance Analysis of Buffer Management Policies: FAT workload

(그림 2) 버퍼 관리 정책 성능 분석: Explorer 워크로드



(Figure 2) Performance Analysis of Buffer Management Policies: Explorer workload

성능이 좋아지는 경향을 보인다.

(그림 1-a) 와 (그림 2-a)는 CLC와 REF의 성능을 나타낸다. Y축은 REF의 수행 시간을 CLC의 수행시간을 기준으로 Normalize 시킨 값이다. FAT 워크로드 의 경우 로그 블록의 개수가 작을 때는 CLC가 REF보다 최대 30% 좋은 성능을 보인다. 로그 블록의 개수가 많아질수록 REF의 성능이 좋아지면서 로그 블록이 1,024개일 때는 REF가 CLC보다 최대 13% 더 좋은 성능을 보인다. Explorer 워크로드의 경우도 비슷한 경향을 보인다는 것을 알 수 있다. 이 결과에서 알 수 있듯이, 쓰기 버퍼 관리 정책의 성능은 로그 블록의 수에 영향을 받게 되는데, 그 이유는 (그림 1-b,c) 과 (그림 2-b,c) 를 통해서 알 수 있다. (그림 1) 과 (그림 2) 들은 버퍼 교체에 따라 요청된 클러스터의 수, 로그 블록 miss의 수, 그리고 로그블록 참조율을 보여준다. 요청된 클러스터의 수는 쓰기 버퍼에서 얼마나 잘 페이지들을 클러스터링하여 FTL에 쓰기 요청을 했는가를 의미하며, 로그 블록의 수와는 상관이 없고 쓰기 버퍼의 크기에만 크게 의존한다. (그림 1) 과 (그림 2) 에서 볼 수 있듯이, CLC 는 REF 에 비해 더 적은 수의 클러스터를 요청 한다. 이것은 CLC가 REF 보다 쓰기 버퍼에서의 클러스터링을 잘한다는 것을 보여준다. 그러나 CLC 는 로그블록의 수가 128개 이상인 경우 REF에 비해 로그 블록 miss를 더 많이 발생시키고 있다. 로그 블록 miss가 발생하면 로그 블록 관리 정책에서는 희생 로그 블록을 선택하고 병합 연산을 수행하게 된다. 따라서 로그 블록 miss의 횟수는 병합 연산의 횟수와 직결된다. 결과적으로, REF 의 경우 페이지 클러스터링 효과는 CLC보다 떨어지지만 로그 블록의 재 참조율이 CLC보다 높음으로 인해 로그 블록의 수가 많은 경우에는 CLC보다 더 좋은 성능을 보이게 된다. 앞서 기술했듯이 REF는 이미 로그 블록이 존재할 가능성이 높은 클러스터를 요청 하는 정책이므로, 쓰기 버퍼에서 클러스터링 하지 못했던 페이지들을 로그 블록에서 클러스터링하는 효과를 보게 되고, 그로 인해 로그 블록의 수가 충분히 많아져서 로그 블록 참조율이 높아지는 시점부터는 좋은 성능을 내게 된다.

이러한 결과를 통해 우리는 다음과 같은 결론을 내릴 수 있다. 로그 블록의 수가 작을 때는

로그 블록의 참조율이 낮기 때문에 쓰기 버퍼에서 클러스터링하는 효과가 큰 쓰기 버퍼 관리 정책이 좋은 성능을 보일 수 있고, 로그 블록의 수가 많아질수록 로그 블록의 참조율이 높아지게 되고 따라서 로그 블록에 대한 재 참조를 고려하는 정책이 좋은 성능을 보이게 된다.

3.2 블록 패딩의 효과

BPLRU에서 제안된 블록 패딩 기법은 쓰기 버퍼에서 희생으로 선택된 클러스터의 missing 페이지들을 플래시메모리로부터 쓰기버퍼로 읽고 하나의 완전한 데이터 블록을 만든 후에 요청함으로써, FTL에서는 항상 비용이 적은 switch 병합만 발생하도록 한다. 따라서 이 경우에는 로그 블록이 불필요해지게 된다. 호스트로부터의 쓰기 요청이 순차적으로 발생하여 쓰기 버퍼에서의 클러스터링 효과가 크고, 이에 따라 로그 블록에 대한 재참조율이 낮아지게 되면 블록 패딩 기법은 매우 효과적이다. 하지만 그렇지 않은 상황에서는 데이터 패딩 오버헤드로 인해 오히려 SSD의 성능을 떨어뜨릴 수 있다. FTL에서 블록 패딩을 통해 switch 병합이 발생할 때의 비용(CBP)와 블록 패딩을 사용하지 않고 full 병합이 일어날 때의 비용(CFM)는 각각 Eq.1 과 Eq. 2 와 같이 나타낼 수 있다.

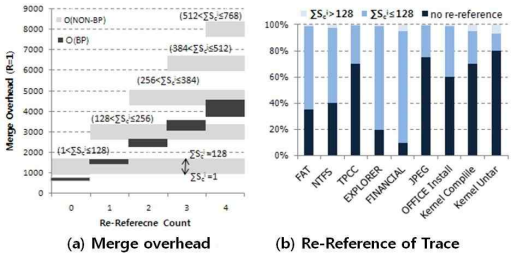
$$C_{BP} = (N_p - S_c) * R + N_p * W + E \quad \text{(Eq. 1)}$$

$$C_{FM} = S_c * W + N_p * (R + W) + 2E \quad \text{(Eq. 2)}$$

(Np: 블록 안의 페이지 개수, Sc: 클러스터 크기 ($\leq$ Np), R: 읽기 비용, W: 쓰기 비용, E: 지우기 비용)

플래시메모리 특징을 따르고 R을 1로 정하였을 때 (Np = 128, R = 1, W =6, E = 10), 위 식은 각각 CBP = 906 - Sc, CFM= 6 * Sc + 916가 된다. 블록 패딩을 사용하지 않을 경우, 쓰기 버퍼에서 선정된 희생 클러스터 가 로그 블록에 쓰인 후, 그 로그블록에 매핑되는 데이터 블록의 페이지들을 저장하고 있는 클러스터 가 다시 희생으로 선정될 경우, 그 로그블록은 재참조가 일어나게 된다. 여기서 처음으로 선정된 희생 클러스터 의 크기를 이라 하고, 두 번째로 선정된 희생 클러스터의 크기를 S_c^1라 할 때, $S_c^1 + S_c^2 > 128 (= Np)$ 인 경우 로그블록 초과 가 발생하게 된다. 이 경우 128개의 페이지가 모두

(그림 3) 로그 블록 재참조 고려의 필요성



(Figure 3) Necessity of considering log block re-reference

사용된 로그블록에 대해 full 병합이 발생하게 되고, 나머지 $(S_c^1 + S_c^2)\%128$ 개의 페이지들은 새로운 로그블록에 기록된다. 새로운 로그블록에 대한 재 참조가 다시 일어나지 않는 경우, 그 로그블록에 대해서도 full 병합이 결국 발생하게 된다. 이 과정에서의 총 비용을 계산하면, $(6*128+916)+6*((S_c^1+S_c^2)\%128)+916$ 가 된다. 이것을 일반화시켜서 쓰기 버퍼에서 희생 클러스터로 선정된 블록이 로그 블록에 쓰인 이후 로그블록에서 다시 n-1번의 재참조가 일어나는 경우 (로그 블록에 대해 총 n 번의 참조가 일어난 경우), 필요한 전체 비용은 다음과 같다.

$$C_{FM} = \left\lfloor \sum_{i=1}^{n} S_c^i / 128 \right\rfloor * (6*128+916) + 6*(\sum_{i=1}^{n} S_c^i \% 128) + 916$$

(Eq. 3)

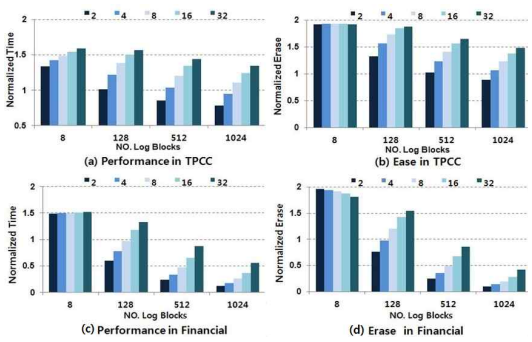
여기에서는 해당 로그블록에 요청 되는 i번째 희생 클러스터의 크기를 말한다.

블록 패딩을 사용하는 경우에는, 쓰기 버퍼에서 희생으로 선정된 클러스터 가 플래시메모리로 요청 될 때마다 switch 병합이 발생하게 되므로, 동일한 data block 에 해당하는 희생 클러스터 들이 총 n 번 요청 될 경우의 전체 비용은 다음과 같다.

$$C_{BP} = \sum_{i=1}^{n} (906 - S_c^i) \quad \text{(Eq. 4)}$$

(그림 3-a) 는 Eq. 3과 Eq. 4를 기반으로, 블록패딩과 full 병합의 오버헤드를 재참조 횟수에 따라서 비교한 것이다. 로그블록에 대한 재참조가 발생하지 않을 경우에는 데이터 패딩 기법이 항상 full 병합 기법보다 적은 오버해드를 보인다. 한 번 이상 재참조가 발생하는 경우에는 희생 클러스터의 크기에 따라 서로 다른 결과를

(그림 4) 블록 패딩의 유무에 따른 성능과 지우기 횟수 비교



(Figure 4) Performance and erase count comparison between w/ and w/o Block Padding (B: buffer size)

보이게 되는데, $\sum_{i=1}^{n} S_c^i \leq 128$인 경우에는 full 병합 기법이 데이터 패딩 기법보다 병합 오버헤드가 작다. 따라서 재참조가 일어나지 않는다는 가정하에서는 블록패딩을 항상 사용하는 것이 좋으며, 재참조가 일어나는 경우에는 그 횟수와 희생 클러스터의 크기 분포에 따라 선별적으로 블록 패딩 기법을 사용하는 것이 필요하다. (그림 3-b) 는 로그블록의 개수가 1,024 이고 쓰기 버퍼의 크기가 2MB 일 경우, 각 워크로드의 로그 블록 재참조율을 보여준다. 로그 블록이 재참조되는 경우는, 그 로그블록이 참조될 때마다의 희생 클러스터 크기를 모두 합하여서 그 크기가 128보다 큰 경우와 그 이하인 경우로 나누었다. (그림 3-b) 에서 알 수 있듯이, 모든 워크로드에서 일관되게 나타나는 현상은, 로그블록에 대한 재참조는 크게 1)아예 발생하지 않는 경우(블록 패딩을 사용하는 것이 좋은 경우)와, 2)횟수와 상관없이 $\sum_{i=1}^{n} S_c^i \leq 128$인 재참조들이 발생하는 경우(블록 패딩을 사용하지 않는 것이 좋은 경우)로 나누어진다는 사실이다. 또한, 재참조가 일어나지 않는 로그블록의 비율은 워크로드마다 크게 달라진다. 따라서 블록패딩 기법은 각 로그블록의 재참조 여부에 따라서 적응적으로 사용할 필요가 있다.

MLC 낸드 플래시 칩의 지우기 사이클은 10K에 불과하므로, SSD의 성능을 높이 것도 중요하지만 Erase의 횟수를 줄여 수명을 연장시키는 것 또한 중요한 문제이다. (그림 4) 는 TPCC와 Financial 워크로드에서 블록 패딩이 지우기 횟수에 미치는 영향을 보여준다. 쓰기버퍼의 관리를 블록 단위 LRU로 하고, 추가적으로 블록패딩을 사용하지 않았을 경우의 수행시간과 지우기 연산의 횟수를 블록 패딩을 사용했을 때를 기준으로 정규화 시켰다. (그림 4) 에서 두 경우의 성능 차이에 비해 지우기 횟수의 차이가 더 큰 것을 알 수 있다. 즉, 블록 패딩을 사용하면, 많은 경우 성능 향상 효과 보다 지우기 횟수를 줄이는 효과가 더 크다는 것을 보여준다. 앞 절에서 설명했듯이 한 번도 재참조가 일어나지 않은 로그 블록은 블록 패딩을 안 했을 경우 full 병합이 발생하면서 2번의 지우기가 발생하는 반면, 블록 패딩을 했을 경우에는 한번의 지우기 연산만 발생한다. (그림 1, 2) 에서 보여주듯이 로그블록의 개수가 작을 때(8개)는 로그 블록 재참조가 거의 일어나지 않으므로, (그림 4) 에서 볼 수 있듯이 로그블록의 개수가 8개인 경우는 블록패딩을 사용하지 않았을 때의 지우기 연산의 횟수가 블록패딩을 사용했을 경우의 두 배 가까이 된다.

재참조가 일어나는 로그블록의 경우, 블록 패딩을 사용하면 재참조가 발생할 때 마다 한 번의 지우기 연산이 추가적으로 발생하는 것으로 볼 수 있으며, 블록 패딩을 사용하지 않을 경우에는 재참조로 인해 로그블록 초과가 발생할 경우에만 한 번의 지우기 연산이 추가로 발생하게 된다. 그러나 (그림 4) 에서 볼 수 있듯이, 재참조가 일어나더라도 초과가 발생하지 않는 경우($\leq$128 인 경우)가 대부분을 차지하므로, 블록패딩을 사용하지 않으면 재참조로 인한 추가적인 지우기는 거의 발생하지 않는다고 볼 수 있다. 따라서 한 번의 재참조만 발생하면 두 경우의 지우기 연산 횟수는 비슷하게 나타나게 되고, 그 이상의 재참조가 발생하면 블록패딩을 사용할 경우 오히려 더 많은 지우기 연산이 발생하게 된다. (그림 4-b,d) 에서 로그블록의 개수가 많고 쓰기 버퍼의 크기가 작은 경우, 로그블록에 대한 재참조가 빈번하게 발생하면서 블록패딩을 사용하는 경우 지우기 연산의 횟수가 더 많아지는 것을 볼 수 있다. 따라서 지우기 횟수를 줄이는 측면에서 볼 때에도, 블록패딩 기법은 각 로

그블록의 재참조 여부에 따라서 적응적으로 사용할 필요가 있다.

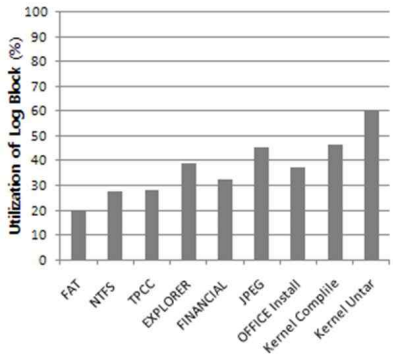
지금까지의 결과를 통해 우리는 다음과 같은 결론을 내릴 수 있다. 블록패딩 기법은 각 로그 블록의 재참조 여부에 따라서 적응적으로 사용할 필요가 있으므로, 각 로그블록에 대해 재참조 여부를 판단할 수 있는 방법이 필요하다. 이를 통해, 재참조가 일어나지 않는 블록에 대해서는 블록 패딩 기법을 사용하고 재참조가 발생하는 블록에 대해서 블록패딩 기법을 사용하지 않는 정책이 좋은 성능을 보이게 된다.

3.3 로그 블록 사용률

앞 절에서 언급했듯이 MLC 낸드 플래시 칩에서는 지우기 사이클이 중요한 문제이다. 따라서 지우기 연산은 꼭 필요한 경우에만 발생하도록 유도하는 것이 필요하다. 만약 지우기 연산을 수행할 대상이 되는 로그블록에 오직 하나의 페이지에만 데이터가 써져 있고 나머지 페이지는 깨끗한(clean) 상태라면, 지우기 연산을 통해 추가적으로 확보되는 공간은 하나의 페이지에 불과하므로 머지않은 미래에 추가 공간 확보를 위한 지우기 연산이 다시 발생할 확률이 높아진다. 그러나 로그블록 내의 전체 페이지가 사용 중인 상태라면, 지우기 연산을 통해 하나의 블록에 해당하는 추가 공간을 확보할 수 있게 되므로 추가적인 지우기 연산의 필요성이 상대적으로 줄어들게 된다. 결과적으로, 전체적인 지우기 연산의 횟수를 줄이기 위해서는 로그블록의 사용률을 높게 유지하는 것이 필요하다.

(그림 5) 는 로그블록의 개수가 1,024개 이고 쓰기 버퍼의 크기가 2MB 일 경우, 재참조가 발생한 로그블록들이 병합(full 병합)되는 시점에서의 평균 사용률을 나타낸 것이다. 재참조가 발생하지 않는 로그블록에 대해서는 블록패딩을 사용한다고 가정하고, 그 경우에는 블록 사용률이 100% 라고 볼 수 있으므로, (그림 5) 에서 그 경우는 제외했다. (그림 5) 에서 볼 수 있듯이, 로그 블록 사용률은 워크로드에 따라 20% ~ 60% 사이로 전반적으로 매우 낮게 나타난다. 이것은 로그 블록에 대한 한번 지우기 연산에 의해 확보되는 공간이 지워진 블록의 크기보다 훨씬 작다는 것을 의미한다. 따라서 로그 블록의 낮은 사용률을 높임으로써 전체적인 지우기 횟수를 줄일 수 있는 기법이 필요하다.

(그림 5) 로그 블록 사용률



(Figure 5) Utilization of Log Blocks

쓰기 버퍼의 크기가 커질수록 데이터가 버퍼에서 클러스터링 될 확률이 높아지게 되어, 로그 블록에서의 재참조 확률은 떨어지고 그에 따라 지우기 연산의 횟수가 증가하게 된다.

4. 결론

지금까지 우리는 워크로드의 특성, 쓰기 버퍼의 크기, 로그 블록 개수 및 재참조 여부가 쓰기 버퍼 정책의 성능과 지우기 횟수에 미치는 영향을 실험을 통해 분석하였다. 이러한 분석 결과를 바탕으로 얻을 수 있는 교훈을 다시 정리하면 다음과 같다.

1. 로그 블록의 개수에 따라서 쓰기 버퍼에서의 데이터 클러스터링에 가중치를 두는 정책과 로그 블록에서의 재참조에 가중치를 두는 정책의 성능이 달라지므로, 이 두 정책의 장점을 살리는 새로운 쓰기 버퍼 관리 정책의 개발이 필요하다.

2. 재 참조가 일어나지 않을 로그 블록은 데이터 패딩 기법을 사용하고 재참조가 일어날 로그 블록에 대해서는 데이터 패딩 기법을 사용하지 않은 것이 SSD의 성능과 수명에 유리하다. 따라서 로그 블록의 향후 재참조 여부를 예측할 수 있는 방법이 개발될 필요가 있다.

3. 로그 블록의 사용률을 높임으로써, 지우기 연산에 따른 추가 확보 공간의 크기를 증가시켜 전체적으로 지우기 연산의 횟수를 줄일 수 있는

방법의 개발이 필요하다.

References

[1] Gray, J. Tape is dead, disk is tape, flash is disk, RAM locality is king. Pres. at the CIDR Gong Show, Asilomar, CA, USA, 2007.

[2] J. Gray and B. Fitzgerald. Flash disk opportunity for server-applications. Online, 2007.

[3] S.-Y. Park, D. Jung, J.-U. Kang, J.-S. Kim, and J. Lee, "CFLRU: A Replacement Algorithm for Flash Memory," Proc. Int'l Conf.Compilers, Architecture and Synthesis for Embedded Systems, Oct.2006.

[4] Hoyoung Jung, Kyunghoon Yoon, Hyoki Shim, Sungmin Park, Sooyong Kang, and Jaehyuk Cha. "LIRS-WSR: Integration of LIRS and writes sequence reordering for flash memory" Lecture Notes in Computer Science, 4705:224-237, 2007.

[5] Hoyoung Jung, Sooyong Kang, and Jaehyuk Cha. "An Asymmetric Buffer Management Policy for SSD", Journal of Digital Contents Society, Vol. 12, no. 2, pp. 141-150, 2011.

[6] Aleph One Company "Yet Another Flash Filing System."

[7] D. Woodhouse, "JFFS: The Journaling Flash File System."

[8] UBIFS Documentation, Nokia & University of Szeged[Z]. 2008. http://www.linux-mtd.infradead. org/doc/ubifs.html.

[9] J. Kim, J.M. Kim, S.H. Noh, S.L. Min, and Y. Cho, "A Space-Efficient Flash Translation Layer for Compact Flash Systems," IEEE Trans. Consumer Electronics, vol. 48, no. 2, pp. 366-375, May 2002.

[10] S.-W. Lee, D.-J. Park, T.-S. Chung, D.-H. Lee, S. Park, and H.-J. Song, "A Log Buffer Based Flash Translation Layer Using fully Associative Sector Translation," ACM Trans. Embedded Computing Systems, vol. 6, no. 3, 2007.

[11] H. Jo, J.-U. Kang, S.-Y. Park, J.-S. Kim, and J. Lee, "FAB: Flash-Aware Buffer Management Policy for Portable Media Players," IEEE Trans. Consumer Electronics, vol. 52, no. 2, pp. 485-493, May 2006.

[12] H. Kim and S. Ahn, "BPLRU: A Buffer Management Scheme for Improving Random Writes in Flash Storage," Proc. Sixth USENIX Conf. File and Storage Technologies, Feb. 2008.

[13] D. Seo and D. Shin. Recently-evicted-first buffer replacement policy for flash storage devices. Trans. On Consumer Electronics, 54(3):1228-1235, 2008.

[14] Sooyong Kang, Sungmin Park, Hoyoung Jung, Hyoki Shim, Jaehyuk Cha, "Performance Tradeoffs in Using NVRAM Write Buffer for Flash Memory-based Storage Devices", IEEE Transactions on Computers, vol. 58, no. 6, June, 2009

박 성 민

2007년 : 한양대학교 전자컴퓨터통신 대학원(공학석사)
2007년 ~ 현 재 : 한양대학교 전자컴퓨터통신 대학원(박사과정)

관심분야 : 플래시 메모리, 임베디드 시스템, 운영체제

강 수 용

1998년 : 서울대학교 컴퓨터 과학 대학원 (공학석사)
2002년 : 서울대학교 컴퓨터 과학 대학원 (공학박사)

2002~현 재: 한양대학교 컴퓨터공학부 교수
관심분야 운영체제, 멀티미디어시스템, 스토리지 시스템, 플래시 메모리, 차세대 메모리, 분산 컴퓨팅