대용량 플래시메모리에서의 가비지 컬렉션 부하를 줄이기 위한 가상 소형 블록 기법

(Virtual Small Block Technique to Reduce Overheads from Garbage Collections in Large Block NAND Flash Memory)

하 건 수, 김 지 홍*
(Keonsoo Ha and Jihong Kim)

서울대학교 컴퓨터공학부
(Department of Computer Engineering, Seoul National Univ.)

Abstract : In log-based FTLs, the switch operation and the partial merge operations in a garbage collection are efficient methods in terms of the number of required operation to get a free block. However, as larger NAND flash memory are applied to embedded systems, chances to induce the operations are reduced. In order to trigger more number of switch and partial merge operations, we suggest a virtual small block technique which logically divides a block into small blocks. In our experiments, this technique can reduce overheads from garbage collections by 10% compared to FAST.

Keywords : 임베디드시스템, 낸드 플래시메모리, FTL, 로그블록

Ⅰ. 연구 소개

낸드 플래시메모리는 빠른 성능, 적은 에너지 소모, 강한 내구성 등의 특성으로 인해 많은 임베디드 시스템에서 저장장치로 사용되고 있다. 특히 최근에는 고성능 고사양의 PDA, 넷북, 스마트 폰 등에서는 기가급의 낸드플래시메모리가 사용되기 시작할 정도로 고용량의 플래시메모리가 탑재되기 시작했다. 그러나 낸드 플래시메모리는 동일한 위치에서의 데이터 갱신이 불가능하고, 한정된 가능 삭제 수, 읽기와 쓰기 간의 성능 불균형, 연속 쓰기 제한 등의 특성에 의한 제약들 때문에 별도의 FTL(Flash Translation Layer) 라는 계층에서 소프트웨어를 통한 관리가 필요하다는 단점이 있다.

FTL 은 시스템으로부터 요청된 논리블록주소를 플래시메모리내의 물리주소로 변환해주고, 가비지컬렉션을 통해서 플래시 메모리에서 빈 공간을 확보하는 역할을 수행한다. 가비지컬렉션이 플래시 메모리 성능에 매우 중요한 비중을 차지하므로, 이를 최적화하기 위하여 연구들이 꾸준히 진행되어 왔다. 그 중 가장 잘 알려진 FTL으로 BAST, FAST, LAST [1,2,3] 등과 같은 블록 레벨과 페이지 레벨의 사상 기법을 동시에 사용하는 하이브리드 기법들이 있다. 이 기법들은 업데이트가 발생할 때는 페이지 레벨의 로그 블록에 데이터를 저장하고 나머지는 블록 레벨의 데이터 블록에 저장하는 구조를 가지고 있다. 이 중 FAST 와 LAST 는 연속 쓰기 로그 블록을 이용하여, 연속적으로 들어오는 쓰기 요청들을 하나의 블록에 저장한 후, 데이터 블록으로 변환하는 스위칭 연산을 유도하여 가비지 컬렉

* 교신저자(Corresponding Author)
하건수, 김지홍
※ 이 연구를 위해 연구장비를 지원하고 공간을 제공한 서울대학교 컴퓨터 연구소에 감사드립니다. 이 논문은 2009년도 두뇌한국21사업에 의하여 지원되었으며, 2009년도 정부(교육과학기술부)의 재원으로 한국과학재단의 지원을 받아 수행된 연구 (No. R0A-2007-000-20116-0, R33-2008-000-10095-0) 입니다.

션을 하는데 오버헤드를 최소화하기 위해 노력했다.

그런데, 최근 들어 임베디드 시스템의 성능에 대한 요구사항의 증대와 임베디드 시스템용 프로세서의 발전에 힘입어 기가대의 대용량 낸드 플래시메모리가 사용되고 있다. 이런 대용량 낸드 플래시 메모리는 이를 구성하는 각 블록의 크기도 커지게 되었고, 블록 당 페이지의 개수도 증가하게 되었다. 과거 32KBytes 수준에 머물렀던 플래시 메모리 내의 블록은 최근엔 256KBytes에서 512KBytes 까지 사용되고 있다. 그 결과 연속 쓰기 요청이 그만큼 길지 않는 한 스위치 연산이 일어나지 않는 문제가 나타나기 시작했다. 또한 파셜 머지가 일어날 때도 많은 페이지가 연속 쓰기 로그 블록 블록에 저장되어 있지 않기 때문에 파셜 머지를 위해서 부가적인 읽기,쓰기 연산이 많이 따르는 문제가 나타난다.

따라서, 본 연구에서는 큰 블록 사이즈를 가진 고용량 낸드플래시메모리에서 효과적으로 스위치 연산이 발생할 수 있도록, 가상 소형 블록 관리 기법을 제안한다. 본 기법은 하나의 큰 블록을 여러 개의 논리적인 블록으로 나눠서 여러 개의 연속 쓰기 요청 집합들을 동시에 하나의 블록에 저장할 수 있도록 하며, 가상 작은 블록 단위로 스위치와 풀 머지를 수행하여 총 가비지 컬렉션 오버헤드를 감소시키도록 유도한다.

시뮬레이션을 통한 실험 결과, 제안한 기법은 기존 FAST 기법 대비, 가비지 컬렉션 동안의 지우기 연산 횟수를 최대 35% 까지 줄이고 총 가비지 컬렉션 시간을 최대 10% 줄이는 효과를 보여준다.

Ⅱ. 가상 소형 블록 기법

1. 배경 지식

낸드 플래시메모리는 여러 개의 블록으로 구성되어 있으며, 각 블록은 다시 여러 개의 페이지로 구성된다. 페이지는 보통 2KBytes 또는 4KBytes의 크기를 가지며, 하나의 블록은 128 에서 256 개의 페이지로 구성된다. 낸드 플래시메모리는 페이지 단위로 읽기 및 쓰기를 수행하며, 블록 단위로 지우기를 한다. 읽기 연산은 빠른 반면, 쓰긴 연산은 상대적으로 느린 특징을 갖는다. 낸드 플래시 메모리는 동일한 위치에서의 업데이트 연산이 불가능하며, 이로 인하여 유효한 최근 데이터가 어디에 저장되어 있는지 항상 정보를 유지해야 한다. 따라서 무효화된 데이터를 재생산하기 위해서, 가비지 컬렉션을

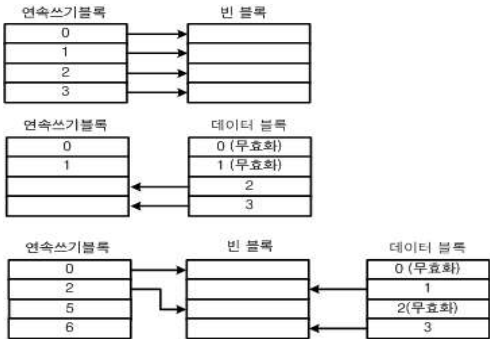

그림 1. 로그기반의 FTL 에서의 가비지 컬렉션 (a)스위치 (b)파셜머지 (c)풀머지

Fig. 1. Garbage Collection Methods in Log-based FTL. (a) Switch (b) Partial Merge (c) Full Merge

통해서 무효화된 데이터를 삭제해야 한다. 이러한 일을 담당하는 소프트웨어 계층으로 FTL 이 있다.

FTL 은 주소 변환의 단위에 따라서 페이지 레벨 블록과 블록 레벨 블록으로 나뉜다. 페이지 레벨은 가비지 컬렉션 오버헤드가 작지만 지나치게 많은 메모리가 요구되고, 블록 레벨은 메모리 요구량은 작지만 가비지 컬렉션 오버헤드가 크다는 단점이 있다. 이들의 장점을 취하기 위해서 BAST, FAST, LAST 와 같은 기법들은 업데이트 되는 데이터에 대해서는 페이지 레벨의 주소 사상을 하고 나머지 데이터는 블록 레벨의 주소 사상을 함으로써 두 기법의 장점을 취하는 접근을 하고 있다.

그림 1은 이들 기법에서 갖는 3가지 가비지 컬렉션 방법을 보여주고 있다. 이중 FAST 와 LAST 의 경우에는 연속쓰기블록을 별도로 할당해서 요청된 데이터의 논리블록주소를 블록당 페이지 수로 나누었을 때, 0 으로 떨어지는 경우 연속쓰기 블록에 데이터를 저장하고, 이어서 연속된 논리블록주소를 가진 데이터가 요청되면 그 블록에 순차적으로 기록한다. 만약 이 연속쓰기블록이 가득 차게 되는 경우 데이터 블록에 스위치 연산을 하여 가비지 컬렉션 오버헤드를 최소화하고자 노력했다. 또는 연속적인 요청이 끊겨도 파셜 머지가 발생하도록 결과적으로 이들 기법들은 가급적 스위치 연산, 파셜 머지를 늘려서 가비지 컬렉션 오버헤드를 최소화 하고자 노력했다.

2. 연구의 동기

향 후, 임베디드 시스템이 점차적으로 데스크톱 환경과 유사한 환경으로 수렴한다고 가정하고, 윈도

우즈 XP를 사용하는 환경에서 추출한 트레이스들을 FAST 가 돌아가는 시뮬레이터에서 실험했을 때, 스위치 연산과 파셜 머지의 빈도를 관찰했다. 총 27644 회의 머지 연산 발생 중, 스위치 는 1 회 뿐이였다. 파셜 머지는 4278 회이고, 파셜 머지가 발생할 때 연속쓰기블록에 남아있던 페이지 개수의 평균은 4 개에 불과하여 블록 내 128개의 페이지 극히 일부만이 채워진 채 파셜 머지 연산이 발생하는 것을 확인할 수 있었다. 이는 파일시스템에 의한 메타데이터의 잦은 업데이트와 웹브라우저, 워드프로세서 등과 같은 비교적 짧은 데이터를 요청하는 응용, 멀티태스킹에 의한 지역성이 낮은 데이터 요청 등에 의해서 생성된 결과라고 할 수 있다.

3. 가상 소형 블록

위의 관찰로부터 짧은 길이의 연속 쓰기 요청을 가급적 스위치 연산 또는 파셜 머지로 유도하기 위해서 본 연구에서는 가상 소형 블록 기법을 제안한다. 가상 소형 블록은 하나의 물리 블록 안에 짧은 연속적인 데이터를 스위치 연산할 수 있도록 하기 위한 논리적인 블록이다. 논리적인 개념에서의 블록이므로, 연속 쓰기 블록을 사용하는 모든 FTL 에 적용 가능하다. 가상 소형 블록은 연속 쓰기 블록을 2의 제곱 등분으로 나눠서 구성하고, 가장 위쪽에 위치한 가상 소형 블록은 0번부터 시작하여 1씩 증가하는 논리적 주소를 가진다.

가상 소형 블록은 하나의 데이터 블록에 위치하고 있는 데이터에 업데이트가 발생한 경우, 해당 데이터의 논리주소가 가상 소형 블록의 크기로 나누었을 때, 나머지가 0 이 나올 때, 가상 소형 블록에 저장하게 된다. 그렇지 않은 경우에는 다른 임의쓰기 로그 블록에 저장된다. 가상 소형 블록이 가득 차게 되거나, 가상 소형 블록 내에서 연속 요청이 깨지는 경우에는 현재 가상 소형 블록을 놔두고 다음 가상 소형 블록을 이용한다. 가상 소형 블록은 하나의 블록 안에 여러 개 존재할 수 있기 때문에, 이런 경우가 발생하더라도 가비지 콜렉션이 바로 일어나지 않는다.

가상 소형 블록은 기존의 블록 주소 사상 단위와 다르기 때문에 별도의 주소 사상 방법을 갖는다. 하나의 데이터가 저장될 때마다 아래와 같은 정보가 사상 테이블에 추가된다.

지 수로 나누면 구할 수 있고, 논리가상소형블록 주소는 논리블록주소를 소형 논리 블록 당 페이지 수로 나눈

> {<논리 블록 주소, 논리 가상 소형 블록 주소>,
> <물리 블록 주소, 물리 블록 내 오프셋>}

후, 한 블록 내의 소형 논리 블록 수로 나누었을 때 나머지 값이 된다. 물리 블록 주소는 FTL에서 새로운 블록을 할당 할 때 결정되며, 데이터가 저장될 때 저장된 위치를 물리 블록 내 페이지 오프셋으로 저장한다. 큰 블록의 페이지는 연속적인 쓰기만을 허락하기 때문에 가장 마지막에 쓴 위치가 물리 블록 내 오프셋이 된다.

FTL 에서는 먼저 가상 소형 블록 주소 변환 테이블을 탐색해보고, 해당하는 주소 정보가 존재하면 사용하고 그렇지 않은 경우에는 기존의 블록 주소 변환 테이블을 탐색하여 원하는 데이터를 읽을 수 있게 된다.

만약 모든 가상 소형 블록에 더 이상 데이터를 기록할 수 없는 경우, 즉 연속 쓰기 블록 내에 가상 소형 블록이 다 소모되면 떨어지게 되면, 그 때 가상 소형 블록은 새로운 연속 쓰기 블록이 할당받게 된다. 이 가상 소형 블록을 저장하고 있는 블록은 데이터 블록으로 변환되며, 추후에 풀 머지 시에 새로운 데이터 블록을 구성할 때 사용된다. 만약, 가상 소형 블록이 유효한 가득 차 있는 경우에는 풀 머지 시에 사용하지 않아서 가비지 컬렉션 오버헤드를 줄인다.

임의쓰기 로그 블록이 가득차서 풀 머지가 발생할 때, 흩어져 있는 모든 데이터를 하나로 모은 후, 주소 사상 정보를 삭제 한다. 풀 머지가 발생하게 되면, 임의쓰기 블록 내에 저장되어 있는 페이지와 같은 논리 블록 주소를 갖는 머든 가상 소형 블록 내의 유효한 데이터를 읽어온 후, 새로운 블록에 저장한다. 그리고 같은 논리 블록 주소를 갖는 데이터 블록 내의 유효한 데이터 역시 읽어서 쓴다. 이 과정에서 가상 소형 블록으로 구성된 데이터 블록 내의 유효한 페이지의 개수가 0 인 경우 해당 블록을 삭제함으로 써, 빈 공간을 확보하게 된다.

Ⅲ. 실험 결과

1. 실험 환경

본 기법의 효과를 평가하기 위해서 FTL 시뮬레이터를 활용하였다. 시뮬레이션을 위해서 데스크톱 환경에서 마이크로소프트의 DiskMon[4] 이라는 툴을 활용하여 트레이스를 추출하였고, 일반적인 컴퓨

표 1. 실험 파라미터

Table 1. Parameters

파라미터	값
읽기	25 us
쓰기	200 us
지우기	2 ms
페이지 크기	4 KBytes
블록 크기	512 KBytes
블록 당 소형블록 수	4개~16개

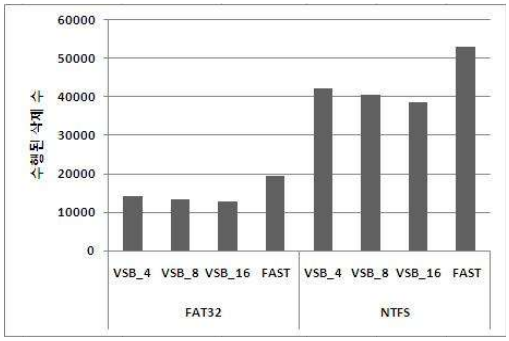

그림 2. 가비지 컬렉션 동안 수행된 삭제 수의 비교

Fig. 2. The number of erase operations during garbage collections

터 사용 시나리오를 가정하고, NTFS 파일시스템과 FAT32 파일시스템 하에 2가지 트레이스를 추출하였다. 실험에 사용된 플래시메모리 파라미터는 표 1과 같다.

2. 실험 결과

그림 2 는 FAST 와 제안된 기법 간의 가비지 콜렉션 동안의 수행된 삭제 수를 보여준다. X 축은 VSB 는 가상소형블록을 의미하고 숫자는 블록 당 소형블록의 수를 의미한다. Y 축은 가비지 컬렉션 동안에 사용된 총 삭제 횟수를 보여준다. FAST 대비 20~35% 가량 삭제 횟수가 줄었으며, 블록 당 소형 블록의 수가 증가할수록 삭제 횟수를 크게 줄일 수 있었다. 이는 기존에 짧은 길이의 요청들이 가상 소형 블록에 저장되어 이동하지 않음으로써 임의 쓰기 블록의 머지 연산 발생을 줄인 것으로 추정된다. 또한 유효한 데이터로 가득 찬 가상 소형 블록은 풀 머지에 개입되지 않기 때문에, 삭제에 연관된 블록의 수가 감소되는 효과로 보여진다. 그림 3은 가비지 컬렉션에 소모된 총 시간을 보여준다. 삭제 횟수의 감소에 따라서 가비지 컬렉션이 4~10% 가량 감소했다. 읽기와 쓰기에서도 파라미터와 기법에 따라서 약간의 증감이 보이긴 하지만, 연산 당 비용이 지우기에 비해서 매우 작기 때문에, 성능에는 크게 영향이 없는 것으로 나타났다.

IV. 결 론

본 논문에서는 큰 사이즈의 블록으로 구성된 고용량

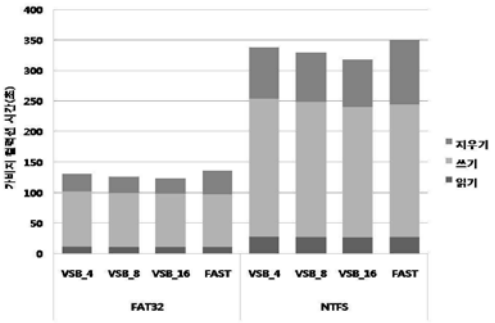

그림 3. 가비지 컬렉션 수행시간

Fig. 3. Total amount of time for garbage collections

낸드플래시 메모리가 스위치 연산에 부적합정도로 큰 점을 고려하여, 가상 작은 블록을 적용하여 가비지 컬렉션 오버헤드를 줄인 결과를 보여준다. 가상 소형 블록은 작은 논리 블록을 이용함으로써 스위치와 풀 머지를 많이 유도함으로써 풀 머지에서 발생하는 삭제 수를 지울 수 있었다. 실험 결과 최대 35 % 까지 삭제 횟수를 줄일 수 있었고, 전체 가비지 컬렉션 오버헤드를 10 % 까지 줄일 수 있었다. 본 기법은 정적으로 가상 소형 블록의 크기를 정하고 하였으나, 가상 소형 블록의 크기가 가변하는 동적인 기법을 연구할 예정이다.

참 고 문 헌

[1] 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 Transactions on Consumer Electronics, Vol. 48, No. 2, May 2002.

[2] S.W. Lee, D.J. Park, T.S. Chung, D.H. Lee, S. Park, H.J. Song, "A Log Buffer based Flash Translation Layer using Fully Associative Sector Translation", ACM Transactions on Embedded Computing Systems, Vol 6. No. 3, July 2007

[3] S. Lee, D. Shin, Y.J. Kim and J. Kim, LAST: Locality-Aware Sector Translation for NAND Flash Memory-Based Storage Systems, International Workshop on Storage and I/O Virtualization, Performance, Energy, Evaluation and Dependability (SPEED) in conjunction with HPCA-14, pp. 52-59, February 2008

[4] http://technet.microsoft.com/en-us/sysinternals/bb896646.aspx