시간 복잡도와 메모리 사용량을 최소화하기 위한 낸드플래시메모리 최소 삭제 블록 탐색 기법

하건수, 김태진, *김지홍
서울대학교 컴퓨터공학부
e-mail : air21c@davinci.snu.ac.kr, taejin1999@davinci.snu.ac.kr, jihong@davinci.snu.ac.kr

A technique for selecting a block with the least erased counts to reduce time complexity and memory requirement for NAND flash memory

Keonsoo Ha, Taejin Kim, *Jihong Kim
School of Computer Science and Engineering
Seoul National University

Abstract

NAND flash memory is widely used in various systems as secondary storage. Since each block in NAND flash memory allows only a limited number of erase operations, dedicated file systems for NAND flash memory must manage the number of erased counts of blocks and allocates a block with the lowest erase counts when a free block is requested. As the capacity of NAND flash memory as well as the number of blocks in NAND flash memory increase, the management overhead for managing the erase counts becomes higher in terms of performance and required memory capacity.

In this paper, we suggest a new free block selection technique which stores only blocks with very small erase counts and allocates free blocks evenly with a small performance overhead and memory usage. Experiment result shows that suggested technique reduces the number of flash memory accesses and computational overhead by 63% than the full-search technique and by 83% than the sampling-based technique, respectively.

본 논문은 BK21사업에 의하여 지원되었으며, 정부(교육과학기술부)의 재원으로 한국과학재단의 지원을 받아 (No. R0A-2007-000-20116-0, R33-2008-000-10095-0) 수행되었습니다. 본 연구를 위해 연구장비를 지원하고 공간을 제공한 서울대학교 컴퓨터 연구소와 IDEC에 감사드립니다.

I. 서론

낸드 플래시 메모리는 서버에서부터 모바일 임베디드 시스템에 이르기까지 다양한 시스템의 보조기억장치로써 사용되고 있다. 낸드 플래시 메모리는 비휘발성, 내구성, 빠른 성능, 저전력 등의 장점들을 갖지만, 연산들의 수행 시간 비대칭성, 업데이트 연산의 불가, 한정된 블록 당 삭제 수 등의 특성으로 인해, 소프트웨어의 관리 없이는 그 장점들을 효과적으로 활용할 수 없다. 특히, 한정된 블록 당 삭제 수는 저장장치의 수명 및 데이터 저장의 안정성 등과 연결되는 문제로써, 반드시 블록 당 삭제 수를 관리하여 블록들이 골고루 삭제될 수 있도록 유도해야 한다.

이를 위해 낸드 플래시 메모리용 소프트웨어인 FTL (Flash Translation Layer)에서는 블록 당 삭제 횟수를 별도의 메타 데이터에 기록해서, 새로운 블록이 필요할 때마다 전체 블록 중 가장 삭제 수가 작은 블록을 탐색해서 찾아낸다(이하 '전체탐색'으로 표기). 이 기법은 하나의 블록을 할당할 때 최소로 삭제된 블록을 찾기 위해 전체 블록의 삭제 수를 비교하므로 매번 블록 개수만큼의 연산 부하가 발생하게 된다.

최근에 이런 문제를 해결하기 위해서, 임의의 블록들만을 샘플링해서, 그 중 가장 작은 블록을 할당하는 방식의 기법이 소개됐다[1](이하 '샘플링'으로 표기). 이 기법은 메모리 사용량 및 연산 부하 측면에서는 큰 이점이 있지만, 샘플링된 블록 정보를 일부 교체할 때마다, 매번 낸드 플래시 메모리를 접근하고, 추가로 정렬도 해야 하므로 오히려 낸드 플래시 메모리 접근이 증가하고 따라서 성능 감소를 유발하게 된다.

본 논문에서는 작은 메모리에서도 낸드 플래시 메모리 접근을 최소화시켜서 성능 감소를 최소화하고, 탐색 없이 작은 삭제 수를 갖는 블록을 간단한 큐를 통해 찾아내는 기법을 제안한다. 제안하는 기법은 로그 형태로 순차적으로 새로운 블록을 할당함으로써, 블록들이 골고루 사용될 수 있도록 유도한다. 또한 낸드 플래시 메모리에서 삭제 연산이 일어날 때마다, 일정 수준 이하의 삭제 수를 가진 블록만을 별도로 저장해 놓고 이에 우선순위를 부여함으로써 최소 삭제 블록을 관리한다.

낸드 플래시 메모리 시뮬레이터를 통해 실험한 결과, 기존 기법들과 비슷한 수준의 최대 블록 삭제 수를 유지하면서도, 블록 탐색 시 필요한 낸드 플래시 메모리 접근 수를 전체 탐색 기법 대비 약 63% 감소시키고, 연산 부하는 샘플링 기법 대비 약 83% 감소시켰다.

본 논문의 순서는 블록 삭제 수 관리를 위한 성능 및 메모리 요구량 부하를 먼저 소개한 후, 제안한 기법을 자세히 설명한다. 마지막으로 실험 결과를 통해서 기법을 평가한 후, 결론 및 추가 연구 방향을 기술한다.

II. 본론

2.1. 연구의 동기

낸드 플래시 메모리는 여러 개의 블록으로 구성되어 있다. FTL이나 파일시스템에서는 각 블록의 상태를 메모리에 모두 또는 일부를 저장하게 되는데, 최근 낸드 플래시 메모리의 고용량화로 인하여 낸드 플래시 메모리를 위한 주소 사상 테이블, 블록 상태, 블록 별 삭제 수와 같은 메타데이터 용량이 증가하게 되었다. 따라서 이 정보의 일부만을 메모리에 저장해서 관리하고, 필요한 정보가 메모리에 없는 경우 플래시로 부터 읽어오는 방법을 취하고 있다[2]. 이 논문에서도 블록의 상태와 삭제 횟수 등을 동일한 방식으로 관리한다고 가정한다.

각 블록은 최대 허용 삭제 수를 갖는데, 이 이상 삭제하면 저장하고 있는 데이터의 손실이 발생할 수 있다. 최근에 사용되고 있는 낸드 플래시 메모리는 1만회로 삭제 수를 제한한다. 단 하나의 블록이 최대 허용 삭제 수만큼 지워지더라도 그 블록의 데이터 유실이 발생하게 되고, 저장 장치로써의 안정성이 떨어지므로 전체 플래시 메모리의 수명이 다한 것으로 여겨지게 된다. 따라서 FTL에서는 각 블록의 삭제 횟수를 별도의 메타데이터로 기록해놓고, 새로운 블록이 필요할 때 마다, 모든 블록 중 가장 삭제 횟수가 작은 블록 탐색 후 할당함으로써, 저장장치의 수명을 연장시킨다. 이러한 전체탐색 기법은 성능 상의 이유로 사용되기 어렵다. 실험 결과, 전체탐색 기법을 사용하면 최소 삭제 블록 탐색에 소모되는 연산 부하는 FTL에서 동작하는 전체 연산 부하 중 약 14%를 차지한다. 이 수치는 가비지 콜렉션에 소모되는 연산 부하를 제외하고 가장 높은 수치다.

이러한 연산 부하를 줄이기 위해서, 샘플링 기법에서는 임의로 선택한 몇 개의 블록 정보들만 메모리에 저장하고 이들을 블록의 삭제 수 순으로 정렬 한 후, 그 중 가장 삭제 수가 작은 블록을 선택하는 방법을 쓰고 있다. 블록 할당 후 삭제 횟수가 많은 하위 몇 개의 블록 정보들은 메모리에 유지하는 의미가 없으므로 버리고 다음번 블록 할당 시 이를 보충하기 위해서 추가로 임의의 블록 정보를 플래시메모리로 부터 읽어오고 이들을 다시 정렬 하는 과정을 반복한다. 이 기법은 각 블록의 상태를 메모리에 따로 저장하고 있지 않기 때문에, 임의로 선택한 블록이 빈 블록이 아닌 경우, 비어있는 블록을 찾을 때까지 임의 블록 선택을 재 시도하게 된다. 따라서 하나의 블록을 할당 받을 때마다, 부가적인 플래시 메모리 읽기를 동반하며 반응 시간을 늦추는 단점이 있다. 실험 결과, 샘플링 기법의 탐색 부하의 경우 하나의 비어있는 블록을 얻기 위해서는 평균 3회 이상의 플래시 읽기 연산을 동반하며, 약 180μsec.이 추가로 소모되어 결과적으로 성능이 크게 떨어짐을 확인할 수 있다.

2.2 최적화된 최소 삭제 블록 선택 기법

제안하는 기법의 목적은 최소 삭제 블록을 고를 때, 적은 메모리 사용하고 탐색을 수행하지 않으면서도, 낸드 플래시 메모리 접근을 최소화 시킴으로써 성능을 향상시키는 것이다. 이를 위해서 제안한 기법은 로그 방식의 순차적인 블록 할당을 통하여, 골고루 블록이 할당되는 것을 유도하고, 가장 많이 삭제된 블록과 가

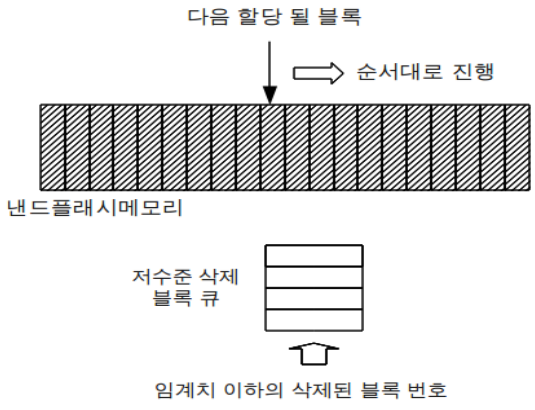

그림 1. 제안된 기법의 개관

장 적게 삭제된 블록의 격차를 줄이기 위하여 적게 삭제된 블록 중 일부를 큐에 저장해놓고, 큐잉된 블록에 우선순위를 부여한다. 본 기법의 장점은 순차적으로 블록을 할당하므로, 추가적인 메타데이타를 메모리에 유지할 필요가 없으며, 매번 정렬을 수행하지 않는다. 게다가 캐싱된 블록 삭제 정보를 순차적으로 활용함으로써, 낸드 플래시 메모리 접근을 최소화한다.

그림1은 제안한 기법의 개관을 보여준다. 그림 상단의 화살표는 새로운 블록의 할당 요청 시, 할당 될 블록을 가리키고 있는 포인터를 나타낸다. 이 포인터는 오른쪽으로 한 블록씩 전진하면서 요청한 블록을 할당한다. 가비지 콜렉션 과정에서 하나의 블록이 삭제될 때, 해당 블록의 삭제 횟수가 지정한 범위에 해당되면 그 블록의 번호는 그림 하단의 저수준 삭제 블록 큐에 저장된다.

알고리즘 1에서는 이 기법의 블록 할당 방법을 자세히 설명한다. 제안된 기법에서는 FTL에서 블록하나가 삭제될 때 마다, 그 블록의 삭제수가 가장 큰 블록 삭제수인 max 값 대비 n분의 1미만일 때, y_b에 그 블록 번호를 저장해놓는다. 만약에 이런 블록이 존재하면 바로 할당해서, 최대와 최소 삭제 수 간의 격차를 줄이려는 시도를 한다. 그런 블록이 없다면, 원래 진행방향대로, 이전에 할당된 블록 *prev_b* 에서 하나의 블록을 증가해서 다음 블록이 비어있는 블록인지, 몇 번 삭제되어있는지 확인한다. 이 블록이 비어있지 않은 블록이라면 블록 포인터를 하나 증가시켜서 똑같은 검사를 반복한다. 만약 비어있는 블록이면, 이 블록의 삭제 수를 살펴보고 지금까지 삭제됐던 블록의 최대 삭제 수인 *max*와 비교한다. 이 때 해당 블록의 삭제 수가 더 큰데도 이 블록을 할당하게 되면 최대 블록 삭제 값인 *max* 가 증가하게 되므로, 이를 피하기 위해서 저수준 삭제 블록 큐로부터 하나의 블록 번호를 꺼낸다. 저수준 삭제 블록 큐에는 이미 최대 블록 삭제 수 *max* 대비하여 일정 수준 이하로 지워진 블록의 숫자가 들어가 있으므로, 이를 활용한다. 그러나 만약 큐가 비어있는 상태라면, 불가피하게 *max* 를 갱신하면서, 그 블록을 할당한다. 이 때, 할당된 블록 b 값을 prev_b에 저장함으로써, 다음에 할당할 때, 이전 순서에 이어서 할당 될 수 있도록 한다.

```
integer prev_b  // prvious allocated block
integer b       // block number to be allocated
integer y_b     // the youngest block
integer max     // the maximum erase count
GetFreeBlock()
if y_b >= 0 then
   b ← y_b,  y_b ← -1,
else
   b ← prev_b + 1
   while TRUE
     if is_free(b) = TRUE  then
       if erase_count(b) > max then
          b ← delete_q()
          if b >= 0 then return b end if
          if b < 0 then max ←b break end if
       end if
       if erase_count(b) <= max then
          break
       end if
     end if
     b ← b + 1
   end while
   prev_b ← b
end if
return b
```

알고리즘 1. 할당될 블록을 선택하는 알고리즘

Ⅲ. 실험

제안된 기법을 평가하기 위해서 자체 개발한 낸드 플래시 메모리 시뮬레이터를 활용하였다. 사용한 시뮬레이터의 플래시메모리 블록 당 사이즈는 256KBytes에 2K개의 블록들로 설정하여, 총 512MBytes 용량의 낸드 플래시 메모리로 실험하였다. 입력 트레이스로는 [3]에서 사용되었던, 윈도우즈 XP 에서 멀티미디어 응용프로그램들과 인터넷 익스플로러 등을 수행시키면서 추출된 I/O 트레이스를 사용하였으며, 많은 블록 당 삭제 수를 유도하기 위해서 동일한 트레이스를 반복해서 시뮬레이션했다. 제안한 기법에 사용된 임계치 값들로는, 블록이 삭제 될 때, 삭제 수가 *max*의 3분의 1이하이면 *y_b*에 설정되며, 3분의 2 이하이면 저수준 삭제 블록 큐에 블록 번호를 삽입한다. 또한 비교 대상인 샘플링 기법의 경우에는 총 30개의 블록정보를 임의로 뽑은 후, 하위 5개의 블록정보만을 매번 교체하도록 설정했다. 낸드 플래시 메모리의 읽기연산 소모시간은 60μsec로 결정했다[4]. 소프트웨어인 FTL은 FAST 를 사용하였다[5].

그림2는 쓰기 요청이 증가함에 따라, 각 기법 별 가장 많이 삭제된 블록의 삭제 수를 보여준다. X축은 쓰기 요청 수를, Y축은 블록 중에서 가장 많이 삭제된 블록의 삭제 수를 나타낸다. 모든 기법들의 최대 삭제 수가 비슷한 숫자로 증가함을 보이므로, 제안한 기법이 별도의 탐색 과정이 없이도, 탐색을 이용하는 다른 기법들에 비해 플래시 메모리의 수명을 빠르게 단축시키지 않음을 보여준다.

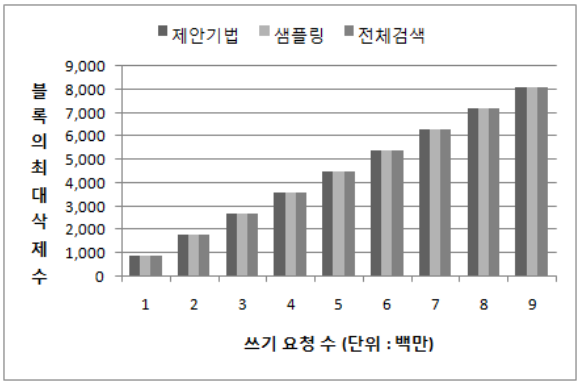

그림 2 기법 별 가장 많이 삭제된 블록의 삭제 수

	제안기법	샘플링	전체탐색
평균응답시간 (μsec)	15	188	41
평균플래시 접근수(회)	0.26	3.14	0.69

표 1. 기법 별 블록 할당 시 평균 응답 시간 및 플래시메모리 접근 수

표 1은 각 기법 별 블록 요청 당, 평균 반응시간과 평균 플래시 접근 수를 보여준다. 제안한 기법의 경우에는 한번 블록을 할당받는데 0.26회 정도 읽기를 수행하므로, 반응 시간에 있어서 다른 기법들에 비해서 상당한 이득이 있음을 볼 수 있다. 샘플링 기법이 크게 좋지 않은 이유는 매번 5개씩 새로운 임의 블록을 얻어오기 위해서 임의 읽기 접근을 하므로, 연속된 블록의 정보를 가지고 있는 삭제 수 정보 캐시를 활용하지 못하기 때문이다.

그림3은 기법 별로 하나의 블록을 할당할 때, 블록 선택에 필요한 연산 부하를 보여준다. Y축은 전체탐색의 연산 부하로 정규화된 값를 보인다. 이 기법이 가장 큰 이유는 모든 메타 블록을 접근해서 정렬하기 때문이다. 샘플링은 30개만 정렬을 반복하므로, 부하가 많이 줄었다는 것을 확인 할 수 있고, 제안한 기법에서는 정렬하지 않으므로, 연산 부하가 거의 없는 것을 확인 할 수 있다.

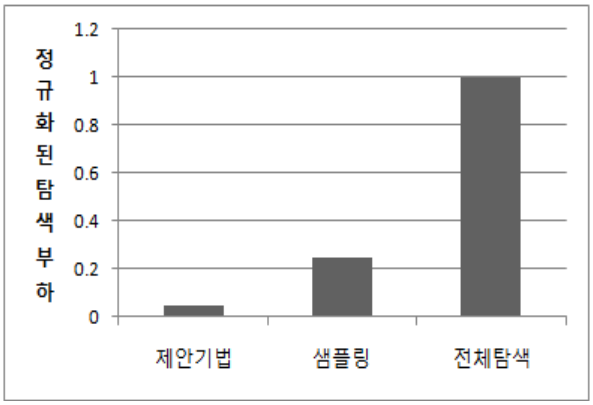

그림 3. 기법 별 정규화된 탐색 연산 부하

Ⅳ. 결론 및 향후 연구 방향

본 논문에서는 낸드 플래시 메모리에서 안정적인 데이터 저장을 위해 필수적인 삭제 블록 관리 기법을 제안했다. 제안한 기법은 매우 작은 메모리 사용량으로도, 최소 삭제 블록을 낸드 플래시 메모리 접근에 의한 성능 저하와 정렬을 위한 연산 부하 없이, 최소 삭제된 블록을 효과적으로 찾아낸다.

향후 연구 방향으로는, 제안한 기법이 메모리 크기가 충분치 않은 열악한 환경을 가정했기 때문에, 각 블록에 저장되는 데이터의 지역성을 고려한 동적인 블록 교체 기법[6]과의 상호 작용을 아직 고려하지 않고 있으므로 이를 함께 고려한 연구를 계획하고 있다.

참고문헌

[1] Biplob Debnath et al, “Sampling-based Metadata Management for Flash Storage,” Technical Report No. ARCTiC 10-01, 2010.

[2] Aayush Gupta et al, “DFTL: a flash translation layer employing demand-based selective caching of page-level address mappings,” The 14th International Conference on Architectural support for Programming Languages and Operating systems, 2009.

[3] Li-Pin Chang et al, “An Adaptive Stripping Architecture for Flash Memory Storage Systems of Embedded Systems,” IEEE 8th Real-Time and Embedded Technology and Applications Symposium, 2002.

[4] Samsung Electronics, Datasheet(K9LBG08U0M), 2007.

[5] Sang Won Lee et al, “A log buffer based flash translation layer using fully associative sector translation,” ACM Transactions on Embedded Computing Systems, vol. 6, no. 3, 2007.

[6] Yuan-Hao Chang et al, “Endurance Enhancement of Flash Memory Storage Systems: An Efficient Static Wear Leveling Design”, Proceedings of the 44th annual Design Automation Conference, 2007.