적응적 N : N+K 매핑을 사용하는 플래시 변환 계층

김기택*, 신동군*
*성균관대학교 정보통신공학부
e-mail : mijurang@korea.com, michaelk92@skku.edu, dongkun@skku.edu

Flash Translation Layer Using Adaptive N : N+K Mapping

Ki Tak Kim*, Dongkun Shin*
*School of Information and Communication Engineering,
Sungkyunkwan University

요 약

플래시 메모리(Flash Memory) 기술이 빠르게 발전하면서, 플래시 메모리 기반의 저장 장치가 개인용 컴퓨터나 엔터프라이즈 서버 시스템과 같은 시스템에 2차적인 저장 장치로써 사용가능해지고 있다. FTL(Flash Translation Layer)의 기본적인 기능은 플래시 메모리의 논리 주소를 물리 주소로 바꾸는 것임에도 불구하고, FTL의 효율적인 알고리즘은 성능과 수명에 상당한 효과를 가지고 있다. 이 논문에서는 MP3 플레이어와 디지털 카메라, SSDs(Solid-State Disk)와 같은 낸드 플래시 메모리(NAND Flash Memory) 기반의 어플리케이션을 위한 N : N+K 매핑을 사용하는 새로운 FTL 설계를 제안한다. 성능에 영향을 미치는 매개변수들을 분류하여, 다양한 워크로드 분석을 기반으로 FTL을 조사했다. 우리가 제안하는 FTL을 가지고, 낸드 플래시 어플리케이션 가동에 따라 어떤 매개변수가 최대 성능을 낼 수 있는지 알아낼 수 있고, 그 변수들을 유연하게 조정하여 성능을 더 향상시킬 수 있다.

1. 서론

낸드 플래시 메모리는 비휘발성, 신뢰성, 저전력, 물리적 충격 내구성의 특성 때문에 MP3 플레이어와 MMC 카드, 휴대폰, PDA와 같은 모바일 장치에 많이 사용되고 있다. 낸드 플래시 메모리의 bit당 가격이 점점 떨어짐으로써 낸드 플래시 기반의 SSDs는 고성능의 대용량 저장장치로써 노트북 PC 시장에 침투하고 있다.

낸드 플래시 메모리는 쓰기(write) 작업을 하기 전에 삭제(erase) 작업을 먼저 해야 하는 특징을 가지고 있다. 그것은 주어진 위치에 새로운 데이터를 쓰기 전에 제일 먼저 삭제를 해야만 하고, 이 때문에 읽기(read), 쓰기, 삭제 작업의 연산 순서를 다루기 위한 FTL이라 불리는 플래시 관리 소프트웨어를 필요로 한다.

2. 배경지식 및 관련연구

2.1 매핑 기법 (Mapping Scheme)

FTL은 낸드 플래시 메모리의 논리 주소를 물리 주소로 바꾸는 역할을 한다. 쓰기 작업 전에 삭제 작업을 하는 특징 때문에 매핑 기법이 낸드 플래시 메모리 기반 제품의 성능과 수명을 결정하는데 중요하다. 주소 변환은 매핑 테이블을 기반으로 하고, 매핑 정보가 관리되는 방법에 따라, 블록 매핑과 페이지 매핑, 2가지 종류의 매핑 기법이 있다.

페이지 매핑에서는 페이지 기반으로 관리된다. 쓰기 요청이 어떤 논리 페이지 주소로 주어질 때, 거기에 상응하는 물리 페이지 주소를 매핑 테이블(Map Table)을 사용해서 찾는다. 페이지 매핑 기법은 데이터를 플래시 메모리의 어떤 프리 페이지(Free Page)에도 쓸 수 있다는 장점을 가지고 있으며, 그것은 좀 더 유연한 저장 관리를 가능하게 해준다. 그러나 페이지 매핑 기법은 매핑 테이블을 위한 거대한 양의 메모리 공간을 요구한다는 큰 문제가 있다.

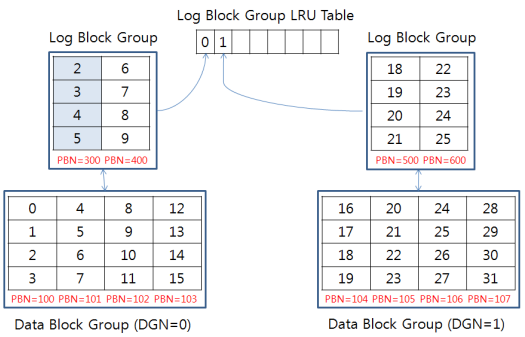
매핑 테이블 사이즈를 줄이기 위하여, 블록 매핑이 고안될 수 있다. 블록 매핑에서는 논리 페이지 주소를 논리 블록 번호와 페이지 오프셋으로 나눌 수 있다. 맵 테이블이 블록 넘버 목록으로 구성되어 있기 때문에, 맵 테이블의 사이즈를 줄일 수 있다. 그러나 블록 매핑은 순차적인 액세스 패턴에 대해서는 더 좋은 성능을 보여주지만, 랜덤 액세스 패턴에 대해 상당한 성능 저하를 보여준다.

페이지 매핑과 블록 매핑의 합의점으로써, 매핑 테이블 사이즈뿐만 아니라 블록 카피 오버헤드를 줄이기 위해 하이브리드 매핑 기법이 사용될 수 있다. 하이브리드 매핑에서는 몇몇 블록이 로그 블록(Log Block)이라 불리는 쓰기 버퍼 블록에 할당된다. 이 로그 블록 기법의 기본 아이디어는 덮어쓰기 작업을 위한 쓰기 버퍼로 제공하기 위해 플래시 메모리안의 적은 수의 로그 블록을 유지하는 것이다.

2.2 로그 버퍼 기반의 FTL 기법

하이브리드 매핑 기법을 사용하는 BAST(Block-Associative Sector Translation)[1]라 불리는 로그 버퍼 기반의 FTL이 고안되었다. BAST의 주요 목적은 순차 쓰기와 랜덤 쓰기를 모두 효율적으로 다루기 위한 것이다. 만약, 쓰기 요청이 있다면, BAST 기법은 데이터를 로그 블록에 연속적으로 쓰고, 로그 블록을 위해서 분리된 페이지 레벨 매핑 정보를 유지한다. 적은 양의 로그 블록만이 FTL에 사용되기 때문에 추가적인 매핑 오버헤드를 낮게 유지할 수 있다. BAST의 결점중 하나는 하나의 로그 블록이 오직 한 데이터 블록만 연관되기 때문에 로그 블록의 활용도가 낮은 경향이 있다는 것이다. 그 결과 블록 삭제 카운트의 수가 사용되지 않은 프리 페이지의 양과 함께 증가 할 수 있다.

BAST의 문제를 해결하기 위해 FAST(Fully-Associative Sector Translation)[2]기법이 제안되었다. FAST에서는 로그 블록이 모든 데이터 블록에 의해 공유되고, 모든 쓰기 요청은 현재의 로그 블록에 쓰인다. 이것은 로그 블록의 저장 활용도를 효율적으로 향상시키며, 머지(Merge) 작업을 가능한 미룰 수 있다. 그렇지만, 하나의 로그 블록이 여러 데이터 블록에 속해있는 페이지들을 포함하고 있기 때문에 BAST 기법보다 더 자주 머지가 발생할 수 있어, 머지가 자주 발생하게 되면, 머지 오버헤드가 커질 수 있다.



(그림 1) SAST 기법

최근에 N to N+K Mapping이라는 SAST(Set-Associative Sector Translation)[3]기법이 제안되었다. 이 기법은 어플리케이션의 동작에 따라 N과 K라는 매개변수에 의해 유연한 매핑 방식을 정적으로 배열하는 것이 가능하다.

그림 1에서와 같이(그림 1의 경우 N이 4이고, K가 2), 하나의 데이터 그룹은 일련의 데이터 블록들로 이루어져 있고, 그것은 연속적인 N개의 블록들이고, 매개변수 N은 하나의 데이터 블록 그룹에 있는 데이터 블록들의 개수이다. 로그 블록 그룹은 특정 데이터 블록 그룹에 상응하는 로그 블록들의 집합이고, 매개변수 K는 하나의 로그 블록 그룹의 최대 로그 블록 개수를 의미한다. 하나의 데이터 블록 그룹은 오직 하나의 로그 블록 그룹을 가질 수 있다. 로그 블록에서 페이지 주소에 상응하는 데이터 블록 그룹에 있다면, 그 페이지는 순서를 고려하지 않고 쓰일 수 있다.

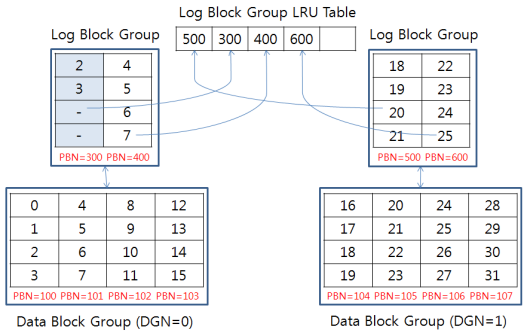
하지만 이 SAST기법은 상황에 따라 서로 다른 매개변수가 적용될 수 있고, 작업 기간 동안 내내 같은 매개변수에 최적화되어있지는 않을 수 있다. 특정 작업을 하는 중에도 어떤 시점에는 더 큰 매개변수 값이 필요할 수 있고, 어떤 시점에는 더 작은 매개변수 값이 필요할 수 있다. 이런 경우까지 모두 고려하지 않고 평균적인 매개변수만을 사용하면 FTL의 성능을 효율적으로 끌어낼 수 없게 된다.

3. A-SAST
(Adaptive Set-Associative Sector Translation)

3.1 희생(Victim) 로그 블록 선택

기존 SAST 논문에서는 로그 블록 그룹 LRU(Least Recently Used) 테이블을 사용한다. 이 테이블은 로그 블록 그룹 단위로 LRU를 계산하는데, 로그 블록 그룹의 마지막 로그 블록에 따라 LRU가 계산되기 때문에 계산상의 모순이 발생 할 수 있다.

그림 1의 경우 기본 SAST 기법으로, 입력순서가 18, 19, 20, 21, 2, 3, 4, 5, 6, 7, 8, 9, 22, 23, 24, 25인데, 실제로 18, 19, 20, 21이 있는 로그 블록이 가장 오래된 로그 블록이지만 로그 블록 그룹 LRU의 특성으로 인해 2, 3, 4, 5가 있는 로그 블록이 희생될 로그 블록으로 선택된다. 이처럼 발생하는 모순을 해결하기 위해 로그 블록 그룹 단위가 아닌 로그 블록 LRU 기법을 사용할 수 있다.



(그림 2) A-SAST 기법

로그 블록 LRU기법을 사용하면, 로그 블록 그룹 LRU 기법보다 더 효율적이기는 하지만, 두 기법 모두 희생 로

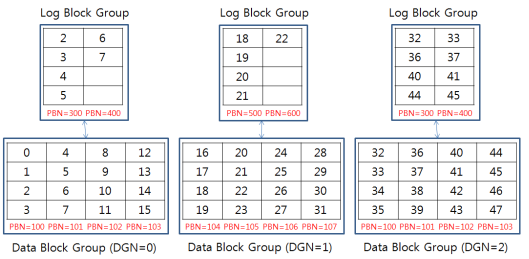
그 블록을 선택하였을 때, 머지 코스트(Merge Cost)가 높은 로그 블록을 선택 할 수도 있다. 머지 코스트는 성능에 직접적인 영향을 주므로, 머지 코스트가 낮은 희생 로그 블록을 선택하여야 하며, 머지 코스트가 낮은 희생 로그 블록을 선택하기 위해서는 로그 블록 LRU에 따라 가장 오래된 로그 블록을 몇 개 선택한 후, 머지 코스트가 가장 낮은 로그 블록을 희생 로그 블록으로 선택한다. 이때 로그 블록의 머지 코스트는 무효한(Invalid) 페이지 개수나 로그 블록과 연관된 데이터 블록 개수로 계산한다.

그림 2의 경우, 입력순서가 18, 19, 20, 21, 2, 3, 4, 5, 4, 5, 6, 7, 22, 23, 24, 25인데, 실제로 18, 19, 20, 21이 있는 로그 블록이 가장 오래된 로그 블록이고, 2, 3, 4, 5가 있는 로그 블록이 2번째로 오래된 로그 블록이다. 하지만 18, 19, 20, 21이 있는 로그 블록은 2번의 머지를 필요로 하고, 2, 3, 4, 5가 있는 로그 블록은 4와 5가 무효화 되면서 1번의 머지만 필요로 한다. 따라서 머지 코스트가 적은 2, 3, 4, 5가 있는 로그 블록을 희생된 로그 블록으로 선택하는 것이 효율적이다.

이처럼 로그 블록 LRU 기법을 사용하되, 가장 오래된 로그 블록을 몇 개 선택하여, 그중에서 머지 코스트가 가장 적은 로그 블록을 희생 로그 블록으로 선택하면 훨씬 더 좋은 성능을 얻을 수 있다.

3.2 데이터 블록 그룹 병합 및 분할

기존 SAST 논문에서는 N : N+K 매핑 기법을 사용한다. 한 데이터 블록 그룹이 최대 K개의 로그 블록을 할당받을 수 있지만, 특정 시점에서 어떤 데이터 블록 그룹은 K개 이상의 로그 블록을 필요로 할 수도 있고, 어떤 데이터 블록 그룹은 하나의 로그 블록도 사용하지 않을 수도 있다. 또는 특정 시점에서 입력 상황에 따라 최적인 N의 개수가 다를 수 있다. 이처럼 어떤 경우는 더 작은 N이 효율적일 수 있으며, 어떤 경우는 더 큰 N이 효율적일 수 있다.

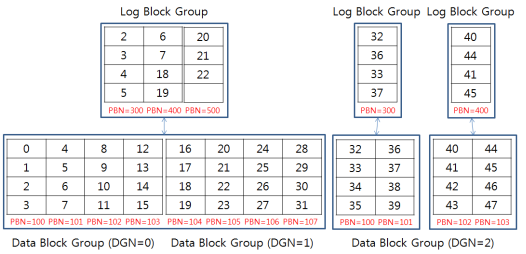


(그림 3) SAST 기법

그림 3의 왼쪽 두 데이터 블록 그룹의 경우에는 입력순서가 2, 3, 4, 5, 6, 7, 18, 19, 20, 21, 22인데, 데이터 블록 그룹이 다르기 때문에 서로 다른 로그 블록을 사용한다.

하지만 이 경우 각 로그 블록 그룹의 마지막 로그 블록에 사용하지 않은 페이지를 많이 가지고 있다. 이런 페이지가 많아지는 건 그만큼 로그 블록의 활용도가 떨어짐을 의미한다. 그림 5의 경우, 각 로그 블록에는 순차적인 페이지들이 들어있기 때문에 머지 코스트가 작다. 이렇게 머지 코스트가 작은 블록들로 이루어진 데이터 블록 그룹들을 하나로 합쳐 새로운 데이터 블록 그룹으로 만들어도 머지 코스트는 크게 변화가 없고, 로그 블록의 활용도는 높아진다.

그림 4의 왼쪽 데이터 블록 그룹의 경우에는 입력순서가 2, 3, 4, 5, 6, 7, 18, 19, 20, 21, 22인데, 2개의 데이터 블록 그룹을 1개로 합쳐, 로그 블록 그룹을 같이 사용하고 있다. 각 로그 블록들의 페이지들은 순차적이기 때문에 머지 코스트가 작고, 사용하지 않는 페이지가 적기 때문에 로그 블록을 1개 절약할 수 있어 성능을 높일 수 있다.



(그림 4) A-SAST 기법

예제 3의 오른쪽 두 데이터 블록 그룹의 경우에는 입력순서가 32, 36, 40, 44, 33, 37, 41, 45인데, 데이터 블록 그룹이 같기 때문에 서로 같은 로그 블록에 쓰인다.

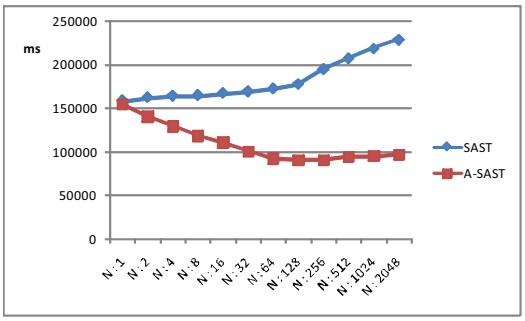
하지만 이 경우 각 로그 블록마다 머지 코스트가 너무 크다. 로그 블록의 각 페이지들은 모두 다른 데이터 블록과 머지를 하기 때문에 프리 블록으로 할당될 때마다 최대 페이지 개수만큼의 머지를 해야 한다. 이렇게 머지 코스트가 큰 블록들로 이루어진 데이터 블록 그룹들을 나누어서 관리하면 머지 코스트를 줄일 수 있다.

그림 4의 오른쪽 데이터 블록의 경우에는 입력순서가 32, 36, 40, 44, 33, 37, 41, 45인데, 데이터 블록 그룹을 2개의 데이터 블록 그룹으로 나누었기 때문에 각 로그 블록들의 최대 머지 코스트가 반으로 줄어들게 된다.

이처럼 하나의 데이터 블록 그룹의 Block 개수 N을 특정 시점이나 특정 상황에 따라 바꿔줌으로 해서 머지 코스트를 줄여 성능을 높일 수 있다.

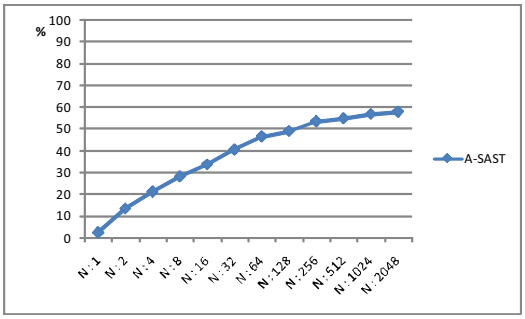
4. 실험 결과

실험 결과에서 보듯이, 매개변수 N과 K에 따라 성능 향상이 차이가 있기는 하지만 기존 SAST보다 최대 60% 까지 성능을 높일 수 있으며, 평균적으로 약 38%정도의 성능을 높일 수 있었다.



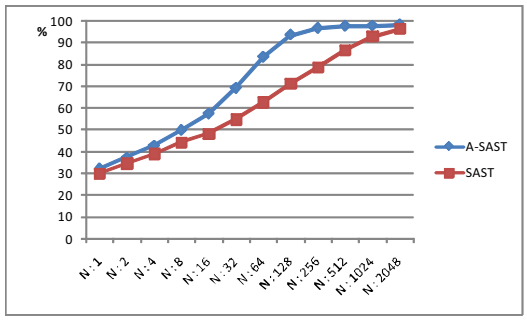
(그림 5) Explorer - 전체 실행 시간
(SAST VS A-SAST)

그림 5는 기존 SAST 기법과 새로운 A-SAST 기법의 성능을 비교해 보여준 것이다. 그림 5의 그래프는 K가 32일 때 결과 값을 나타낸 그래프인데, 기존 SAST의 결과는 기존 연구에서 알려진 대로 결과 그래프가 오목한 모양으로 나타나고 A-SAST 역시 SAST와 마찬가지로 결과 그래프가 오목한 모양으로 나타난다. 위 결과에서 알 수 있듯이, A-SAST는 N의 크기가 커질수록 더 큰 성능 향상을 나타낸다. SAST의 경우 N이 16일 때 가장 좋은 성능을 나타내지만, A-SAST의 경우 N의 크기에 따른 성능향상으로 인해 N이 64일 때 가장 좋은 성능을 나타내는 결과를 나타내었다.



(그림 6) Explorer - 성능 향상률

그림 6은 N값에 따른 성능 향상률로써 그림 5의 성능을 비교한 값을 정리한 결과이다. 그림 6의 그래프에서 보여주는 바와 같이, A-SAST는 N값에 따라 큰 성능 변화를 보여준다. 즉, A-SAST는 N이 클수록 더 좋은 성능 향상률을 얻을 수 있다.



(그림 7) Explorer - 로그 블록 활용률
(SAST VS A-SAST)

그림 7은 로그 블록의 활용도를 보여준다. 로그 블록 활용도는 로그 블록이 머지 될 때의 로그 블록의 페이지 사용률이다. SAST의 경우 기존 연구에서 알려진 것과 비슷한 결과 그래프를 나타내었고, A-SAST의 경우 기존 SAST보다 더 좋은 로그 블록 활용률을 보여주고 있다.

5. 결론

이 논문은 PC의 MP3에서 SSDs에 이르는 다양한 낸드 플래시 어플리케이션을 효율적으로 다루기 위한 새로운 FTL 설계를 소개하였다. 이 기법은 데이터 블록 그룹의 데이터 블록 개수 N과, 한 데이터 블록 그룹에 속해있는 로그 블록 그룹의 최대 로그 블록 개수 K에 따라 결과를 나타내고, 그 매개변수 값을 유연하게 조절하여 더 좋은 성능을 나타낼 수 있다. 이 기법은 BAST와 FAST 기법 사이에서 최적의 기법인 SAST 기법의 성능을 더 향상시킨 기법이다.

참고문헌

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

[2] S. W. Lee, D. J. Park, T. S. Chung, W. K. Choi, D. H. Lee, S. W. Park, and 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, 2007

[3] C. H. Park, W. M. Cheon, Y. S. Lee, M. S. Jung, W. H. Cho, and H. B. Yoon, "A Re-configurable FTL (Flash Translation Layer) Architecture for NAND Flash based Applications", 8th IEEE/IFIP International Workshop on Rapid System Prototyping, IEEE, pp. 202-208, 2007.