韓國컴퓨터情報學會 論文誌
第15卷 第8號, 2010. 8.

2010-15-8-1-3

플래쉬 메모리 시스템을 위한 인덱스 블록 매핑

이 정 훈*

Index block mapping for flash memory system

Jung-Hoon Lee *

요 약

플래시 메모리는 비휘발성이며 시스템에 전원이 없는 상태에서도 데이터를 유지할 수 있는 특성을 가지는 메모리이다. 게다가 빠른 접근 시간과 저전력 소비, 충격에 강하고, 작은 크기와 매우 가벼운 특성을 가진다. 가격이 점차 낮아지고 용량이 증가함에 따라 플래시 메모리의 활용도는 가전제품, 내장형 시스템, 그리고 이동 단말기 등에 널리 사용되고 있는 추세이다. 이러한 플래시 메모리를 구동함에 있어서 필수적인 소프트웨어인 FLT이 필요하다. 본 연구에서는 기존의 블록 매핑 알고리즘의 가장 큰 단점을 극복하기 위한 새로운 FTL 알고리즘을 제안한다. 핵심적인 사항은 기존의 블록 매핑 테이블에 추가하여 작은 램 메모리를 이용하여 섹터 위치를 바로 알 수 있는 인덱스 블록 매핑 테이블을 제안하고자 한다. 시뮬레이션 결과에 따르면 제안된 FTL은기존의 하이브리드 매핑과 비교했을 때 수행 시간을 평균 45%정도 줄이는 효과를 얻을 수 있었으며, 메모리 사상 요구량에 대해서 약 12% 줄이는 효과를 얻을 수 있었다.

Abstract

Flash memory is non-volatile and can retain data even after system is powered off. Besides, it has many other features such as fast access speed, low power consumption, attractive shock resistance, small size, and light-weight. As its price decreases and capacity increases, the flash memory is expected to be widely used in consumer electronics, embedded systems, and mobile devices. Flash storage systems generally adopt a software layer, called FTL. In this research, we proposed a new FTL mechanism for overcoming the major drawback of conventional block mapping algorithm. In addition to the block mapping table, a index block mapping table with a small size is used to indicate sector location. The proposed indexed block mapping algorithm by adding a small size. By the simulation result, the proposed FTL provides an enhanced speed than a conventional hybrid mapping algorithm by around 45% in average, and the requirement of mapping memory is also reduced by around 12%.

▸ Keyword : 플래시 변환 계층(flash translation layer), 플래시 메모리(flash memory), 매핑 알고리즘(mapping algorithms)

• 제1저자, 교신저자 : 이정훈
• 투고일 : 2010. 03. 11, 심사일 : 2010. 05. 07, 게재확정일 : 2010. 07. 14.
* 국립경상대학교 공과대학 제어계측공학과 (공학연구원) 부교수
※ 이 논문은 2007년 정부(교육인적자원부)의 재원으로 한국학술진흥재단의 지원을 받아 수행된 연구임(KRF-2007-611-D00028).
www.dbpia.co.kr / kci.go.kr

Ⅰ. 서 론

최근의 플래시 메모리 시스템 개발 동향을 보면 기본적인 플래시 메모리의 사상 정책에 비해서 속도를 조금이나마 향상 시킬 수 있는 버퍼 경영 방식으로 많이들 접근하는 듯하다. 또한 플래시 메모리는 추가적인 소프트웨어인 FTL을 사용하게 되는데, 이것의 기능은 장치 내에서 논리 블록을 물리페이지로 사상시켜주는 역할을 수행한다. 플래시 메모리의 랜덤쓰기 성능은 FTL알고리즘의 효율에 따라서 매우 의존적이라고 볼 수 있다. 이러한 FTL은 다양한 알고리즘이 있으며 플래시 메모리의 핵심 기술 중의 하나이다 [1]. 기존에 제안된 FTL 알고리즘에서 플래시 메모리의 쓰기 전 지우기 성질을 극복하기 위하여 주로 사용한 방법은 논리 물리 사상 방법이다. 즉, 논리 물리 사상 테이블을 이용하여 해당 물리 주소에 이미 데이터가 쓰여 있으면 비어있는 플래시 메모리의 공간에 먼저 쓴 후 논리 물리 사상 테이블을 변경하는 방법이다. 이러한 방법으로 지우기 연산을 줄일 수 있다. 또한 FTL 알고리즘을 실제 플래시 메모리 시스템에 적용할 때 또 다른 중요한 점의 하나는 사상 정보를 위한 메모리 요구량이다. 즉, 사상 정보는 성능을 위하여 값비싼 SRAM에 저장 되는데 이 사상 정보의 양이 많아서 SRAM의 요구량이 많아지면 전체 비용 면에서 FTL 알고리즘을 실제 시스템에 적용되기 힘들다. 현재의 대표적인 기본적인 매핑 기술(mapping technique)은 3가지로 요약할 수 있다[2].

1.1 섹터 매핑(Sector mapping)

이 기법은 단순하면서도 매우 직관적인 방법으로써 모든 논리 섹터(Logical sector)를 상응하는 모든 물리적 섹터(Physical sector)에 1:1로 매핑 시키는 방식이다 [3]. 만약에 파일 시스템에 m개의 논리 섹터가 있다면 논리 섹터 번호와 물리적 섹터 번호를 구성하는 매핑 테이블의 엔트리 개수는 m개가 된다.

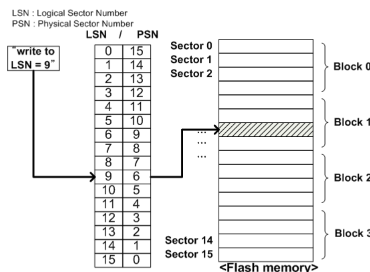


그림 1. 섹터매핑
Fig. 1. Sector mapping

1.2 블록 매핑(Block mapping)

섹터 매핑은 가장 구성이 간단하고 지우기 시점 및 쓰기 동작을 원활히 할 수 있지만 매핑 테이블을 위한 많은 양의 SRAM을 요구하게 되고 플래시 메모리 용량에 따라 기하급수적으로 증가함으로써 현실적으로 사용이 불가능함을 알 수 있다. 이러한 단점을 극복할 수 있는 블록 매핑 방식[4]은 논리 블록 내의 논리 섹터 오프셋(logical sector offset)이 물리적 블록내의 오프셋과 그 값이 동일하다는 것이다. [그림 2]는 블록 매핑의 방식을 나타내고 있다. 4개의 논리 블록(logical block)이 있고 매핑 테이블(mapping table) 또한 4개의 엔트리를 가지고 있다고 가정하자. 동일한 플래시 메모리 용량을 가진 섹터 매핑에 비해 75%의 감소 효과를 보임을 알 수 있다. 그러나 가장 큰 단점은 [그림 2]의 예처럼 다시 논리적 섹터 "9"에 쓰기 동작을 수행하게 되면 하나의 빈 블록을 찾아 오프셋이 1인 곳에 쓰기 동작을 수행해야한다. 만약 섹터 매핑의 경우에는 비어있는 섹터가 블록 내에 존재한다면 같은 블록 내에 쓰기 동작을 수행할 수 있지만 블록 매핑의 경우 오프셋의 지정으로 블록 내에 정해진 위치에만 기록이 가능하여 쓰기 동작 시마다 하나의 블록이 무효 블록이 되어 버리는 단점이 생긴다. 이러한 특성은 지우기 연산이 가장 큰 단점인 플래시 메모리 특성에서 매우 큰 취약점으로 작용한다. 최악의 경우(Worst case)로 그림의 예처럼 논리적 섹터 "9"가 연속적으로 4번 쓸 경우 블록 매핑은 최종 하나의 블록 내에 하나의 섹터만이 유효 섹터가 되지만 3개의 블록을 무효화시키는 결과를 초래한다. 그러나 섹터 매핑의 경우 하나의 블록 내에 하나의 유효 섹터만 존재하지만 3개의 섹터만 무효화 시키는 결과를 초래함으로써 메모리 낭비 및 지우기 동작을 효과적으로 줄일 수 있는 장점을 가진다.

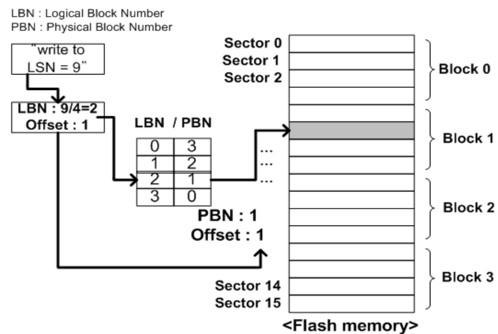


그림 2. 블록매핑
Fig. 2. Block mapping

www.kci.go.kr

1.3 혼합 매핑(Hybrid mapping)

혼합 매핑은 위에서 설명한 섹터 매핑과 블록 매핑의 단점을 보완하여 두 기법들을 활용한 것이다. [그림 3]처럼 물리적 블록을 찾아내기 위해서 블록 매핑을 사용하고 섹터를 찾아내기 위해서 오프셋 대신 섹터 매핑을 사용한다. LSN에 대한 정보를 여유 공간(spare area)에 기입을 함으로써 블록 매핑의 가장 큰 단점인 오프셋 개념을 없앨 수 있는 기법이다 [5]. 이 기법은 매핑테이블에 대한 램용량은 기존의 블록 매핑과 동일하지만 여유공간(spare area)부분에 LSN과 블록 상태에 대한 정보를 저장함으로써 플래시 메모리의 용량을 사용하고 있다. 또한 가장 큰 단점은 하나의 섹터를 찾기 위하여 한 블록내의 모든 섹터를 차례대로 검색해야 한다. 원하는 섹터가 블록내의 앞부분 섹터인 경우에는 검색 시간이 적지만 블록내의 가장 뒷부분에 있는 경우 접근 시간이 긴 플래시 메모리의 한 블록 내 모든 섹터를 다 찾아야 하는 치명적 단점을 가지고 있다. 그러므로 플래시의 지우기 동작 및 SRAM의 사이즈 단점을 극복하기 위하여 가장 기본적인 동작 특징인 읽기 동작 및 쓰기 동작의 시간을 크게 증가시키는 단점을 내포하고 있다.

이처럼 위에서 설명한 기존의 매핑 테이블 기술들은 특히 대용량의 플래시 메모리 사용에 적합하지 않으며 새로운 매핑 테이블 기술이 제안되어야 할 것이다.

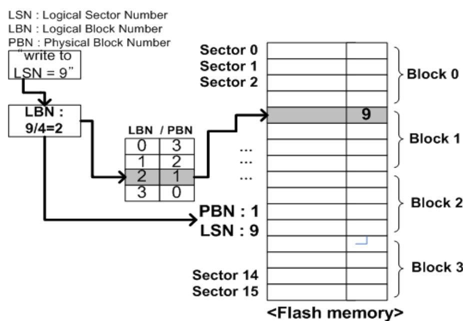


그림 3. 혼합매핑
Fig. 3. Hybrid mapping

II. 기존 연구

위에 제시한 기본적인 매핑 알고리즘에 더하여 대표적인 FTL 알고리즘을 소개한다. Misubish 사의 MITS 알고리즘 [6]은 하이브리드 알고리즘으로써 매핑 테이블은 블록 매핑의 장점을 이용하면서 블록 매핑의 가장 큰 단점인 덮어쓰기의 문제점을 해결하기 위하여 여유공간(spare space)을 이용하여 논리적 번호와 블록 상태 등의 정보를 기록하는 방식이다. 이 방식은 여유 공간을 이용하여 블록 내에서 발생하는 임의 쓰기 연산에 잘 대응하나 여유 공간의 정도에 따라 추가적인 플래시 메모리를 요구하게 되고, 한 블록은 교환하지 못하고 항상 합병해야하는 단점을 가지고 있다.

M-System의 FMAX 알고리즘[7] 역시 블록 매핑의 장점을 이용하면서 복사 블록을 이용하여 덮어쓰기의 문제점을 해결하고자하였다. 이 알고리즘은 복사 블록을 이용하여 단순 블록 상태와 복합블록 상태를 바꾸어가면서 순차적 쓰기 연산과 임의적 쓰기 연산에 대하여 잘 대응하고 있으나 복사 블록을 모든 데이터 블록마다 추가적으로 가지고 있어야하기 때문에 두 배 큰 메모리 요구량을 필요로 하며, 특정 섹터에 복사 블록의 위치를 기록해야함으로 읽기와 쓰기에 대한 추가 비용이 요구된다.

로그블록기법(Log Block Scheme)인 BAST(Block Associative Sector Translation) 알고리즘[8]은 블록 매핑 테이블과 로그블록을 동적으로 할당하기 위한 로그테이블과 로그 블록의 내용을 빠르게 접근하기위한 로그섹터테이블을 이용하는 방식이다. 이 알고리즘은 한 블록에 해당하는 덮어쓰기 연산을 로그블록에 변동섹터방식(outof-place)으로 사용하여 임의 쓰기 연산에 잘 대응하는 알고리즘이지만 로그블록의 수에 따라 성능이 좌우되며, 로그블록의 활용성에도 문제가 많은 것으로 나타나고 있다.

마지막으로 개선된 로그블록 기법인 FAST(Fully Associative Sector Translation) 알고리즘[9]은 순차 쓰기용 로그블록과 임의 쓰기용 로그블록으로 나누어 사용하는 알고리즘으로 데이터의 특성에 따라 효과적인 쓰기 대응에 장점을 가지고 있는 알고리즘이다. 그러나 순차 쓰기 블록이 하나만 존재하며, 성능 향상을 위하여 많은 양의 매핑 테이블이 사용되는 단점을 보이고 있다.

제시된 이러한 대표적인 FTL 알고리즘들은 기본적으로 처음에는 한 블록에 저장되는 섹터들이 정해진 위치(offset)에 저장되는 고정 섹터방식(in-place)이고, 덮어쓰기 동작에 따라 변동 섹터 방식(outof-place)를 이용한다. 즉 이러한 변동 섹터 방식을 이용하기 위하여 MITS 방식은 블록 내의 여유공간를 사용하고, FAMX의 경우 각 블록당 1:1로 복사블록을 사용하며, 로그블록 기법은 블록당 로그 블록 하나를 할당하여 이용하며, 이에 추가하여 FAST는 로그블록을 순차쓰기용과 임의 쓰기용으로 구분하여 이용하는 알고리즘을 사용하고 있다. 이러한 알고리즘에 대한 비교 성능 분석은 [10]에 잘 나타나 있다.

www.kci.go.kr

제시된 알고리즘과의 차별성중 가장 큰 특징 중의 하나는 제안된 구조의 데이터 저장 특성이 변동섹터 방식을 처음부터 이용하고 있다. 제안된 구조는 인덱스 블록 매핑을 이용하여 한 블록내의 비어있는 공간에 임의적으로 데이터를 저장시키는 기법을 이용함으로써 다른 구조와 뚜렷한 특징을 보이고 있다. 또한 위의 구조들은 합병연산을 수행할 때 한 개의 논리블록의 덮어쓰기에 대해 두 개의 물리적인 블록을 소거해야 하지만 제안된 구조는 한 개의 물리적 구조만 소거하므로 소거 횟수를 줄일 수 있는 장점을 가진다.

III. 새로운 인덱스 블록 매핑

위에서 제시된 기존의 매핑 기술의 단점을 극복하면서 대용량 플래시 메모리에 사용 가능한 간단하면서 효과적인 사상 테이블을 제안한다. 이 기법은 앞서 언급한 섹터 매핑, 블록 매핑, 혼합 매핑의 단점을 보완한 것으로써 섹터 매핑의 많은 사상 메모리 요구량, 블록 매핑의 재 쓰기 시의 고정된 오프셋 값에 의한 빈번한 지우기 가능성과 블록 쓰래싱(block thrashing), 낮은 공간 활용도, 그리고 혼합 매핑에서의 여유 공간에 있는 "offset"을 찾아내는데 걸리는 시간적 오버헤드를 효과적으로 줄일 수 있다. 또한 섹터 매핑에 비해 사상 메모리의 크기를 효과적으로 줄일 수 있을 뿐만 아니라 섹터 매핑의 쓰기 방식인 변동 섹터 방식을 잘 활용할 수 있다. 여기서 변동 섹터 방식이란 쓰기 방식으로서 블록 내의 고정된 섹터의 위치 값을 가지는 고정 섹터 방식인 블록 매핑과는 다르게 블록 내에서 페이지 매핑과 같이 자유로의 섹터의 위치를 결정할 수 있다. 허용성이 높은 변동 섹터 방식을 사용하면서 사상 테이블에서 하나의 엔트리 라인을 한 사이클에 검색할 수 있고 덮어쓰기가 가능한 SRAM과 함께 이용함으로써 공간 활용도 및 성능 향상에 기여한다. 다른 사상 알고리즘들과 비교했을 때 요구되어지는 사상 정보를 위한 메모리 용량은 블록 매핑을 제외하고는 제일 적다. 제안된 사상 테이블은 다음 〔그림 4〕과 같다.

제안된 사상 테이블은 기존의 블록 매핑 테이블에 인덱스 정보를 저장하기 위한 추가적인 공간을 가진다. 예로 한 블록 내에 4개의 섹터가 존재하면 첫 번째 섹터에 대한 인덱스는 "00", 두 번째 섹터에 대한 인덱스는 "01", 세 번째 섹터에 대한 인덱스는 "10", 네 번째 섹터에 대한 인덱스는 "11"이다. 제안되는 인덱스 테이블의 동작 방식은 다음과 같이 크게 네 단계로 구분할 수 있다. 첫 번째 쓰기(Write) 요청, 두 번째는 재 쓰기(Update) 요청의 경우, 세 번째로 쓰기시의 합병(Merge) 연산의 경우, 그리고 읽기(Read) 연산의 경우를 들 수 있다.

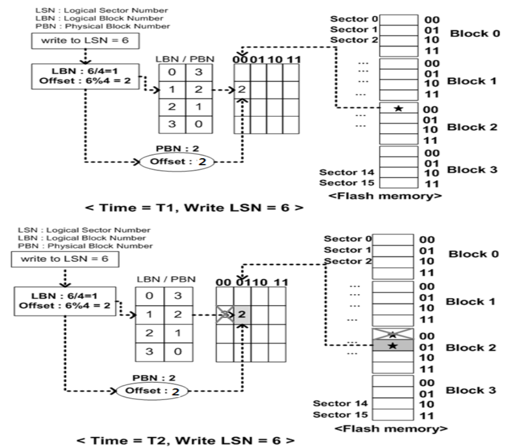


그림 4. 인덱스 블록 매핑의 쓰기 연산
Fig 4. Writing operation of the Index block mapping

첫 번째로 쓰기 요청의 경우(그림 4, Time = T1), "LSN 6"에 쓰기 명령이 들어왔을 때 블록 매핑 기술에 기준하여 "6"을 블록의 개수인 "4"만큼 나누어 주고 그 몫이 "LBN 1"이 되고 그 나머지 "2"가 오프셋(Offset)이 된다. 비어 있는 하나의 블록을 할당하여 쓰기 명령을 수행하지만 블록 매핑과 같이 고정된 위치의 섹터가 아닌 비어있는 섹터에 쓰기 동작이 일어나고 그에 해당하는 인덱스 테이블에 해당 오프셋을 기입한다. (그림 4, Time = T1)의 예는 비어 있는"블록 2"내의 첫 번째 섹터가 비어 있으므로 해당 데이터가 기록되고 해당하는 블록 매핑 테이블 "PBN 2"와 인덱스 테이블 "00"위치에 오프셋 "2"가 기입된 형태로 업데이트되어진다.

두 번째로, 만약 다음 시간 단계(Time = T2)에서 연속적으로 요청되는 LSN이 "LSN 6"으로서 재 쓰기 요청이 발생하면 이미 앞에서 "PBN 2"로 블록의 위치가 정해져 있으며 "PBN 2"위치의 인덱스 테이블을 통해 해당 블록 내에 충분한 공간이 있음을 확인할 수 있으므로 해당 블록으로 이동하여 "블록 2"인 곳의 비어있는 섹터에 업데이트가 일어난다. 여기서는(그림 4, Time = T2) 플래시 메모리 "블록 2"의 인덱스 "01"의 공간이 비어 있으므로 해당 데이터가 "01"에 갱신되고, 해당하는 인덱스 테이블 "01"에 오프셋"2"가 기록됨과 동시에 동일 오프셋을 검색하여 검출되어지면 그 오프셋은 무효화(Invalid) 시킨다. 여기서는 해당 인덱스 테이블 "00"의 오프셋 "2"가 검출됨으로 그 오프셋 값은 무효화되고 최종적으로 "LSN 6"의 데이터는 "PBN 2"와 오프셋 "2"가 인덱스 테이블 "01"에서 검출되어지므로 "블록 2"의 "섹터 01"(인덱스 01)에 해당하는 곳에 최종 데이터가 있음을 알 수 있다.

세 번째로, 〔그림 5〕에서 "LSN 5"이고 재 쓰기 동작이 발

www.kci.go.kr

생하였으나 인덱스 테이블을 검색하면 블록 내에 충분한 공간이 없음을 알 수 있다. 이때 할당 연산을 수행하여 새로운 빈 블록을 할당받는다. 할당 연산은 빈 블록이 필요한 경우 빈 블록을 탐색, 반환하는 연산이다. 보통은 라운드로빈(round robin)방식을 사용한다. 빈 블록이 할당되게 되면 새로운 블록"3"의 인덱스"01"과 인덱스"11"로 블록 내의 위치를 유지한 채로 이전 블록의 유효한 데이터를 복사를 한다. 그 후, 새로 요청된 데이터를 갱신하기 위해서 인덱스를 검색하여 비어있는 섹터를 찾는다. 인덱스"00"과 인덱스"11"이 검색되고, 첫 번째 섹터인"00"에 데이터를 기록한다. 그 다음 인덱스 테이블 "00"에 새로운 오프셋"1"이 갱신되고"10"은 무효화 시켜준다. 마지막으로 기존의 블록 "2"는 삭제 연산이 이루어지고 빈 블록 리스트로 반환되어진다.

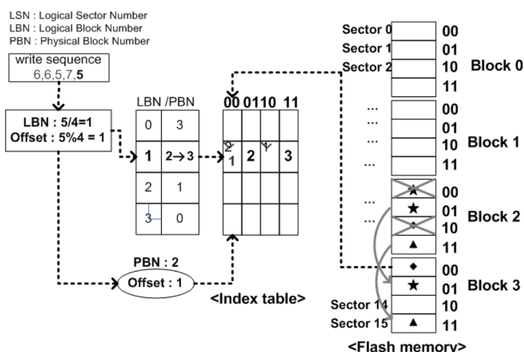


그림 5. 인덱스 블록 매핑의 합병 연산
Fig. 5. Merging operation of the Index block mapping

마지막으로 읽기 연산 요청의 경우에는 요청된 LSN 값을 통해 LBN과 오프셋 값을 계산하고 PBN 값을 통해 해당하는 인덱스 테이블 엔트리에 저장되어 있는 오프셋 값들과 LSN을 통해 구해진 오프셋 값을 비교하여 PBN 안의 세부적인 섹터의 위치를 찾아내고 데이터를 읽는다. 예를 들어 [그림 6]과 같이"LSN 6"로부터 읽기 연산이 요청되었다고 했을 때"LSN 6"은"4"로 나누어 "LBN 1"의 값과 "오프셋 2"의 값을 계산하고"LBN 1"값을 통해 PBN이 가리키는 인덱스 테이블의 해당 엔트리와 구해진 오프셋 값 "2"를 비교하여 오프셋을 검출한다. 그후 오프셋이 가지는 위치값 "01"과 PBN값 "3"을 통해 플래시 메모리 내의 정확한 섹터의 위치를 찾아낸다. 다시 말해서 데이터를 읽을 때에는 "PBN" 값과 해당 인덱스 테이블의 오프셋 비트를 검색함으로써 해당 위치로 빠르게 이동하며, 블록 매핑과 동일한 속도로 플래시 접근이 가능하다.

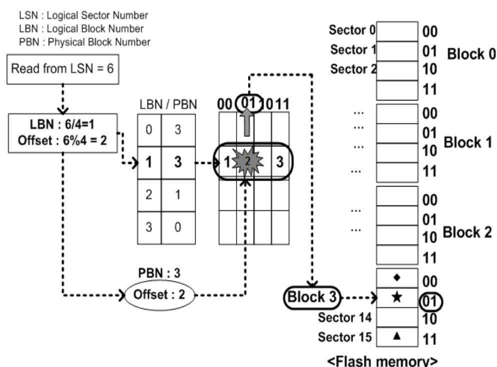


그림 6. 인덱스 블록 매핑의 읽기 연산
Fig. 6. Reading operation of the Index block mapping

제안되는 인덱스 블록 매핑은 다음과 같은 특징을 가진다. 첫째, 쓰기 시에 블록 매핑과 같이 고정된 섹터 값이 아닌 비어 있는 임의의 위치에 데이터를 기록함으로써 비사용 공간을 줄여 공간 활용도를 높여준다. 둘째, 업데이트 발생 시에 블록 매핑과 같이 새로운 블록을 할당하지 않고 비어 있는 공간에 데이터를 기록하고 원래의 데이터는 무효화함으로서 합병 연산을 줄일 수 있으며 그에 따른 추가적인 비용이 발생하는 유효한 데이터의 복사와 블록 삭제를 줄일 수 있다. 이는 임의 쓰기 방식을 통한 공간 활용도가 삭제 연산에 충분히 기인하기 때문이다. 셋째, 혼합 매핑에서의 여유 공간에 저장되어 있는 오프셋을 검출하는 추가적인 오버헤드 없이 인덱스 테이블에서 빠르게 검색하여 플래시 메모리의 해당 데이터에 빠른 속도로 접근이 가능하다. 넷째, 플래시 메모리보다 연산 속도가 상당히 빠른 SRAM을 활용한 인덱스 테이블을 도입함으로써 플래시 메모리에서의 수행시간을 (Access time) 획기적으로 줄여줄 뿐만 아니라 섹터 매핑과 혼합 매핑에 비해서 약 77%, 12%의 적은 양의 사상 메모리 사용량으로도 더 나은 성능 향상을 이루었다.

플래시 메모리에서 페이지는 읽거나 기록되는 가장 작은 단위의 데이터이고 섹터는 파일시스템에서 읽거나 기록되는 가장 작은 단위의 데이터이다. 실제로 사용되어지는 대용량 플래시 메모리의 경우 큰 블록 구조로 기본 페이지 단위가 2KB, 또는 4KB로서 대용량을 지원한다. 2KB인 경우에는 512Byte 섹터 4개를, 4KB인 경우에는 512Byte 섹터 8개를 한 번에 인출한다. 따라서 기존의 작은 블록의 경우인 섹터가 기본 512Byte로서 페이지 단위와 동일할 때와 비교 할 때 연산 수행시간에 있어서 훨씬 효과적이다.

www.kci.go.kr

IV. 인덱스 블록매핑의 성능평가

이장에서는 인덱스 블록 매핑과 사상 정보에 요구되는 메모리 요구량, 그리고 그에 따른 성능 평가에 대해서 논한다. 시뮬레이터는 범용으로 사용되는 운영체제인 윈도우즈 XP 파일 시스템에서 디스크 I/O 패턴을 추출하기 위해 윈도우즈필터 드라이버를 작성하여 파일 시스템에서 디스크로 보내지는 I/O 패턴을 얻어 실험에 사용했다. 또한 실험을 위한 시뮬레이션 도구로 실제 플래시 메모리처럼 동작하는 가상 플래시 메모리와 각 FTL 알고리즘을 직접 구현하였다. 트레이스들은 일반적인 PC의 사용 정보를 보여주는 PCmark05 benchmark로부터 추출하였으며, 트레이스 데이터들은 Windows XP NTFS 파일 시스템 기반에서 8기가바이트의 하드디스크 드라이브로부터 발생되는 입출력 정보를 수집하였다. 트레이스들은 표 1과 같이 5가지로 구성되어있다.

표 1. 입·출력 트레이스
Table 1. Input·output trace

1. Windows Update
2. Install and Setup
3. Clean manager
4. Compress and Decompress
5. General usage

시뮬레이션은 8기가바이트의 플래시 메모리를 기준으로 수행되었다. 기본적으로 제안되는 사상 구조는 큰 블록 구조로서 하나의 페이지는 512Byte sector 4개로 구성된 2048Byte로 구성되어 있으며 한 블록은 128개의 페이지로 구성된다. 자세한 시뮬레이션 사양은 표 2에 잘 나타나있다.

표 2. 시뮬레이션 사양
Table 2. Simulation spec

Flash Memory Features	
Size	8 GBytes
Max Sector Address	4,194,304
Max Page Address	1,048,576
Page Size	2,048 Bytes
Block Size	256 KBytes
Number of Blocks	32,768개
Pages per Block	128개

제안되는 FTL 알고리즘은 기존의 알고리즘과 두 가지 관점에서 비교되었다. 첫째는 플래시 메모리에서의 읽기, 쓰기, 지우기, 그리고 복사를 포함한 전체적인 연산 수행 시간이고 두 번째는 FTL에서의 사상 정보를 위한 메모리 요구량이다. 자세한 설명은 다음의 하위 두 절에 잘 나타나있다.

4.1 수행시간(Time execution)

성능 지표로서 전체적인 연산 수행시간을 이용한다. 전체적인 연산에 있어서 SRAM의 연산 속도는 플래시 메모리에 비해서 훨씬 더 빠르기 때문에 제안되는 FTL의 SRAM 내에서의 연산 시간은 무시할 수 있다고 가정한다. 플래시 메모리에서의 전체적인 읽기와 쓰기 연산에 의한 총 수행시간은 Cread, Cwrite1, 그리고 Cwrite2로 각각 계산되며 수식은 다음과 같다.

$$Cread = Tread * Length \quad (1)$$

$$Cwrite1 = Length * Twrite \quad (2)$$

$$Cwrite2 = (Length * Twrite) + xTwrite + Terase \quad (3)$$

Cread는 읽기 비용, Cwrite는 쓰기 비용이다. Tread, Twrite, and Terase는 각각 플래시 메모리에서의 읽는 비용, 쓰기 비용, 그리고 삭제 비용이다. "x"는 합병 연산시의 유효한 섹터의 개수이고, "Length"는 요청된 섹터의 길이이다. 쓰기 비용을 식 (2) 와 (3) 과 같이 Cwrite1과 Cwrite2로 구분한 이유는 일반적인 쓰기와 합병 연산시의 쓰기의 구별을 위해서이다. 합병연산은 복사와 삭제 연산을 추가적으로 필요로 한다. 기본적으로 섹터 매핑, 블록 매핑, 인덱스 매핑 테이블, 빈 블록 리스트는 SRAM 기반으로 운영되며 이는 SRAM 보다 하위 계층인 플래시 메모리로의 접근을 줄여줌으로서 다른 플래시 변환 계층에 비해서 나은 성능을 기대할 수 있다. [그림 7]은 제안되는 인덱스 블록 매핑과 기존의 블록 매핑을 5가지 패턴을 통한 시뮬레이션 결과를 보여준다. X축은 각각의 패턴을, Y축은 수행 시간을 나타낸다. 인덱스 블록 매핑은 패턴 1. Windows update에서 최대 65%의 성능향상을 보이고 있으며, 대부분의 벤치마크에서 우수한 결과를 보이고 있다. 결론적으로 평균 45% 정도 향상을 보이고 있다.

www.kci.go.kr

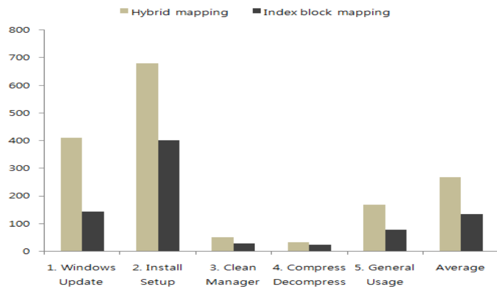


그림 7. 인덱스 블록 매핑과 혼합 매핑의 전체 수행시간 비교
Fig. 7. Timing comparison of the index block mapping and the hybrid mapping

〔그림 8〕는 FTL의 중요 고려 사항인 삭제 횟수를 제안된 인덱스 블록 매핑과 기존의 블록 매핑과 비교하였다. 패턴 4. Compress/Decompress의 경우 약 96% 삭제 횟수를 줄여 주었으며, 평균적으로 약 90% 정도의 성능 향상을 보이고 있다. 이는 블록 내의 자유로운 쓰기 방식을 통해 합병 연산 및 그에 따르는 삭제 연산을 획기적으로 줄임으로서 나타난 결과라고 볼 수 있다.

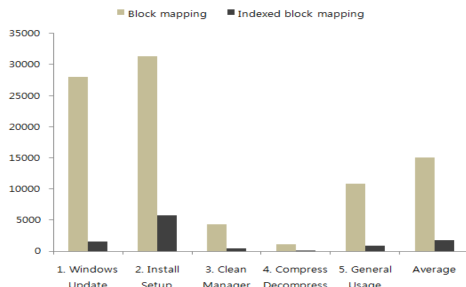
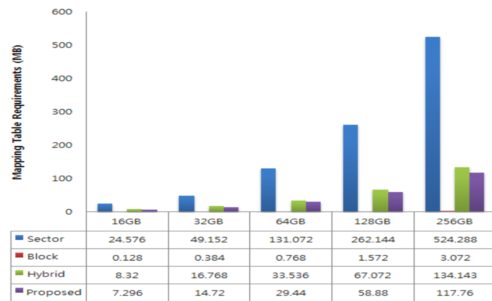


그림 8. 인덱스 블록 매핑과 블록매핑의 삭제회수 비교
Fig. 8. Erasing comparison of the index block mapping and the block mapping

4.2 사상메모리 요구량(Mapping Memory Requirements)

제안된 인덱스 블록 매핑 이용 시 반드시 고려해야 할 대상은 사상 메모리 요구량이다. 사상 메모리 요구량은 매핑테이블을 위한 RAM 메모리와 하이브리드 매핑을 위하여 추가적으로 사용되는 메모리 요구량으로써 섹터매핑, 블록매핑, 제안된 인덱스 블록 매핑의 경우 RAM 메모리만을 사용하고 있으나 하이브리드 매핑의 경우 여유공간에서 사용된 메모리 요구량을 추가적으로 더하여 시뮬레이션을 수행하였다. 제안되는 인덱스 블록 매핑에서의 요구되는 사상 메모리 요구량은 다음과 같이 계산된다. 〔그림 6〕과 같이 한 블록이 4개의 섹터로 구성되어 있다면 요구되는 오프셋 비트는 00, 01, 10, 그리고 11로서 “2bit”이다. 따라서 2bit * 4 = 8bit로 다시 말해서 한 블록별 추가적인 1Byte, 전체 4Byte가 실제로 추가된다. 이와 같이 대용량의 플래시 메모리 저장장치로 이용할 경우 다음과 같이 구성된다. 한 블록내의 섹터가 128개로 오프셋 비트는 0~127로 7비트에 해당한다. 그러므로 한 엔트리 당 7bit * 128 index = 112Byte의 용량으로, 16GB NAND의 경우 65,536개의 블록 엔트리를 가지므로 7.34Mbyte만 추가적으로 사용된다.

사상 메모리 용량에 대한 시뮬레이션 결과를 살펴보면 〔그림 9〕과 같다. 그림 9처럼 블록 매핑에 비해서는 추가적인 용량이 증가하지만 섹터 매핑에 비해서는 대략 77% 감소 효과가 있으며, 혼합구조 보다 사상메모리 사용량에 있어서 12% 차이를 보이는 것은 하이브리드의 경우 블록 매핑 테이블에 더하여 섹터 정보를 저장하기 위하여 플래시 메모리 여유공간 16바이트 내에 논리적 번호와 블록 상태의 정보를 저장하기 위한 메모리 사용에 기인한다.



	16GB	32GB	64GB	128GB	256GB
Sector	24.576	49.152	131.072	262.144	524.288
Block	0.128	0.384	0.768	1.572	3.072
Hybrid	8.32	16.768	33.536	67.072	134.143
Proposed	7.296	14.72	29.44	58.88	117.76

그림 9. 사상메모리 요구량
Fig. 9. Memory mapping requirements

V. 결론

이 연구에서 우리는 인덱스 정보를 위한 추가적인 SRAM 테이블을 도입하여 플래시 메모리로의 빠른 접근과 블록 삭제를 줄일 수 있는 인덱스 블록 매핑을 설계하고 평가하였다. 성능평가 결과 연산 수행시간에 있어서 성능에 결정적인 영향을 미치는 요소는 인덱스 테이블의 유무에 있다고 볼 수 있다. SRAM 기반의 인덱스 테이블를 이용하여 플래시 메모리 내의 접근과 검출을 줄일 수 있으며 인덱스 테이블 내에서 빠른 속도로 오프셋을 검색함으로써 플래시 메모리 내에서 오프셋을 순차적으로 검색해 들어가야 하는 하이브리드 매핑의 시간

www.kci.go.kr

적 오버헤드를 획기적으로 줄일 수 있다.

기존의 하이브리드 매핑과 비교했을 때 전체 수행 시간은 각각의 벤치마크에서 65%, 40%, 44%, 23%, 그리고 53% 향상된 결과를 보였다. 이것은 SRAM 기반의 인덱스 테이블이 플래시 시스템에서 효과적으로 작용했다는 것을 뜻한다. 게다가 블록 매핑과 비교했을 때 우리의 인덱스 블록 매핑은 최대 97%감소된 삭제 연산을 보였다. 블록 매핑의 삭제 연산 횟수는 많은 장점을 가지고 있는 알고리즘에도 불구하고 블록 내의 섹터의 효용성을 감소시키는 원인이 된다. 또한 하이브리드 매핑과 섹터 매핑을 비교했을 때 각각 12%와 78% 감소된 사상 테이블 요구량을 보이고 있다. 즉, 제안되는 인덱스 블록 매핑은 기존의 매핑 알고리즘의 단점들을 극복하면서 저비용으로 고성능의 효과를 얻을 수 있었다.

참고문헌

〔1〕 Intel Corp, "Understanding the flash translation layer(FTL) specification," Application Note 648, 1998

〔2〕 T. S. Chung, "An Efficient Algorithm for Flash Memory,"Journal of KISS :Computer Systems and Theory. 2005

〔3〕 Fujitsu Microelectronics Europe, "COMPARISON OF FLASH PROGRAMMING +ERASE,MB91F467 D&MB91 F362G," Application Note MCU-AN-300 019-E-V10, 2006

〔4〕 T. Shinohara, "Flash memory card with block memory address arrangement," US Patent, No. 5,905,993, 1999

〔5〕 B. S. Kim and G. Y. Lee, "Method of driving remapping in flash memory and flash memory architecture suitable therefor," US Patent, No. 6,381,176, 2002

〔6〕 Takayuki Shinohara. Flash memory card with block memory address arrangement, US Patent, No. 5, 905,993, 1999

〔7〕 Amir Ban. Flash file system optimized for pagemode flash technologies, US Patent, No.5, 937,425, 1999

〔8〕 Jesung Kim, Jong Min Kim, Sam H. Noh, Sang Lyul Min, and Yookun Cho. A space-efficient flash translation layer for compact flash systems. IEEE Transactions on Consumer Electronics, 48(2), 2002

〔9〕 Sang-Won Lee and Dong-Joo Park. FAST:An efficient flash translation layer for flash memory, Submitted for publication, 2005

〔10〕 박원주, 박성환, 박상원, "윈도우즈 기반 플래시 메모리의 플래시 변환 계층 알고리즘 성능 분석,"정보과학회논문지: 컴퓨팅의 실제, 제 13권, 제 4호, 213-225쪽, 2007년 8월

저 자 소 개

이 정 훈

1999: 연세대학교 공학석사.
2004: 연세대학교 공학박사.
2004 - 현재: 국립 경상대학교 제어계측공학과 부교수
관심분야: 내장형시스템, 마이크로프로세서, 플래시메모리 시스템, 고성능/저전력 기술 응용

www.kci.go.kr