NAND 플래시 메모리 저장 장치에서 블록 재활용 기법의 비용 기반 최적화

(Cost-based Optimization of Block Recycling Scheme in NAND Flash Memory Based Storage System)

이 종 민 † (Jongmin Lee) 김 성 훈 †† (Sunghoon Kim) 안 성 준 ††† (Seongjun Ahn) 이 동 희 †††† (Donghee Lee) 노 삼 혁 ††††† (Sam H. Noh)

요 약 이동기기의 저장 장치로 사용되는 플래시 메모리는 이제 SSD(Solid State Disk) 형태로 노트북 컴퓨터까지 그 적용 범위가 확대되고 있다. 이러한 플래시 메모리는 무게, 내충격성, 전력 소비량 면에서 장점을 가지고 있지만, erase-before-write 속성과 같은 단점도 가진다. 이러한 단점을 극복하기 위하여 플래시 메모리 기반 저장 장치는 FTL(Flash-memory Translation Layer)이라는 특별한 주소 사상 소프트웨어를 필요로 하며, FTL은 종종 블록을 재활용하기 위하여 *병합* 연산을 수행해야 한다. NAND 플래시 메모리 기반 저장 장치에서 블록 재활용 비용을 줄이기 위해 본 논문에서는 *이주* 연산이라는 또 다른 블록 재활용 기법을 도입하였으며, FTL은 블록 재활용시 이주와 병합 연산 중에서 비용이 적게 드는 연산을 선택하도록 하였다. Postmark 벤치마크와 임베디드 시스템 워크로드를 사용한 실험 결과는 이러한 비용 기반 선택이 플래시 메모리 기반 저장 장치의 성능을 향상시킬 수 있음을 보여준다. 아울러 이주/병합 연산이 조합된 각 주기마다 블록 재활용 비용을 최소화하는 이주/병합 순서의 거시적 최적화의 해를 발견하였으며, 실험 결과는 거시적 최적화가 단순 비용 기반 선택보다 플래시 메모리 기반 저장 장치의 성능을 더욱 향상시킬 수 있음을 보여준다.

키워드 : 플래시 메모리 저장 장치, 병합 연산, 이주 연산, FTL(Flash-memory Translation Layer)

Abstract Flash memory based storage has been used in various mobile systems and now is to be used in Laptop computers in the name of Solid State Disk. The Flash memory has not only merits in terms of weight, shock resistance, and power consumption but also limitations like erase-before-write property. To overcome these limitations, Flash memory based storage requires special address mapping software called FTL(Flash-memory Translation Layer), which often performs *merge* operation for block recycling. In order to reduce block recycling cost in NAND Flash memory based storage, we introduce another block recycling scheme which we call *migration*. As a result, the FTL can select either merge or migration depending on their costs for each block recycling. Experimental results with Postmark benchmark and embedded system workload show that this cost-based selection of migration/merge operation improves the performance of Flash memory based storage. Also, we present a solution of macroscopic optimal migration/merge sequence that minimizes a block recycling cost for each migration/merge combination period. Experimental results show that the performance of

· 니콘 카메라 트레이스를 제공한 서울대학교 민상렬 교수님께 감사드립니다. 이 논문은 2005년도 서울시립대학교 학술연구조성비에 의하여 연구되었음

† 학생회원 : 서울시립대학교 컴퓨터과학부
jmlee@uos.ac.kr

†† 비 회 원 : 고려대학교 정보보호대학원
kimsunghoon@korea.ac.kr

††† 비 회 원 : 삼성전자 중앙연구소 선임연구원
sjahn@ssrnet.snu.ac.kr

†††† 정 회 원 : 서울시립대학교 컴퓨터과학부 교수
dhl_express@uos.ac.kr
(Corresponding author임)

††††† 종신회원 : 홍익대학교 정보컴퓨터공학부 교수
samhnoh@hongik.ac.kr

논문접수 : 2006년 12월 11일
심사완료 : 2007년 10월 18일

:개인 목적이나 교육 목적인 경우, 이 저작물의 전체 또는 일부에 대한 복사본 혹은 디지털 사본의 제작을 허가합니다. 이 때, 사본은 상업적 수단으로 사용할 수 없으며 첫 페이지에 본 문구와 출처를 반드시 명시해야 합니다. 이 외의 목적으로 복제, 배포, 출판, 전송 등 모든 유형의 사용행위를 하는 경우에 대하여는 사전에 허가를 얻고 비용을 지불해야 합니다.
정보과학회논문지: 컴퓨팅의 실제 및 레터 제13권 제7호(2007.12)
Copyright@2007 한국정보과학회

Flash memory based storage can be more improved by the macroscopic optimization than the simple cost-based selection.

Key words : Flash memory based storage device, merge operation, migration operation, FTL (Flash-memory Translation Layer)

1. 서 론

플래시 메모리는 현재 휴대전화나 PDA와 같은 이동기기 이외에도 SSD(Solid State Disk) 형태로 데스크탑 컴퓨터까지 그 적용 범위를 확장하고 있다. 이러한 플래시 메모리는 마그네틱 디스크에 비해 무게가 가볍고 충격에 강하며 전력 소모가 적다는 장점이 있다. 하지만 플래시 메모리는 데이타를 쓰기 전에 소거해야 하고 쓰기와 소거의 단위가 다른 것과 같은 고유의 특성을 가지고 있으며, 이러한 특성으로 인한 제약을 극복하기 위해 주소 사상 소프트웨어인 FTL(Flash-memory Translation Layer)[1]이 필요하다. FTL은 논리적인 섹터 주소를 물리적인 데이타 위치로 변환하여 플래시 메모리가 디스크 같은 저장 장치로 보이도록 모방하는 소프트웨어 계층이다.

플래시 메모리 셀에 기록된 데이타를 갱신하는 것이 제한되어 있기 때문에 어떤 섹터에 대하여 쓰기가 요청되면 FTL은 새로운 영역으로 섹터를 재사상한다. 전체적으로 FTL의 동작은 LFS(Log-structured File System)[2]와 유사하다[3]. LFS는 쓰기 요청마다 파일 데이타를 새로운 영역에 기록하고 사상 정보를 갱신하며, 공간 재활용 기법에 의해 디스크 공간을 재사용한다. 플래시 메모리의 경우 역시 FTL은 공간을 재활용해야 하는데, 특히 NAND 플래시 메모리의 경우 물리적인 소거 단위인 블록들을 재활용해야 한다. 따라서 앞으로 FTL의 공간 재활용 기능을 블록 재활용 기법이라 부른다. FTL의 블록 재활용 기법은 종종 *병합(merge)* 연산이라고 불리는데, 이는 두 개 이상의 블록에 흩어져 있는 섹터들을 새로운 블록에 모으고 기존 블록들을 재사용하기 때문이다. LFS와 유사하게 FTL의 블록 재활용 기법의 효율성은 플래시 메모리 저장 장치의 성능에 큰 영향을 미친다. 그런데 블록 재활용 기법의 효율성은 데이타 쓰기 패턴에 따라서 달라질 수 있다. 만약 데이타가 순차적으로 쓰여졌다면, 병합 연산은 비교적 적은 비용으로 블록들을 재활용할 수 있지만, 무작위로 또는 반복적으로 데이타가 기록되면 블록 재활용 비용은 커진다.

플래시 메모리 저장 장치가 데스크탑 컴퓨터와 서버까지 적용되면서 신뢰성과 저전력 이외에도 성능에 대한 요구가 높아지고 있다. 본 논문에서는 이주(*migration*) 연산이라는 새로운 블록 재활용 기법을 설명하는데, 이 기법은 블록 재활용 비용을 감소시키고 플래시 메모리 기반 저장 장치의 성능을 향상하기 위하여 도입되었다. 특히, 반복적인 쓰기 패턴의 경우 기존의 병합 연산을 적용하면 블록 재활용 비용이 매우 커지는데, 이주 연산은 이러한 반복적인 쓰기 패턴의 블록 재활용 비용을 줄일 수 있다. 아울러 병합 연산과 이주 연산의 비용 모델을 제시하여 FTL이 블록을 재활용할 때 병합 연산과 이주 연산 중에서 비용이 적게 드는 연산을 선택할 수 있도록 하였으며, 앞으로 이러한 기법을 *비용 기반 이주/병합 선택*이라 부른다. Postmark 벤치마크와 임베디드 시스템 워크로드를 이용한 실험은 비용 기반 이주/병합 선택이 플래시 메모리 기반 저장 장치의 성능을 향상시킴을 보여준다. 그리고 이러한 실험을 통하여 이주/병합 연산이 혼합된 주기(period)에서 블록 재활용 비용을 최소화 시키는 해가 존재함을 인식하였으며, 앞으로 이를 *이주/병합 순서의 거시적 최적화*라 부른다. 본 논문은 이주/병합 순서의 거시적 최적화의 수학적 해를 제시하며, 실험을 통하여 거시적 최적화가 플래시 메모리 기반 저장 장치의 성능을 더욱 향상시킴을 보인다.

논문의 구성은 다음과 같다. 다음 절에서는 플래시 메모리의 특징을 설명하고, 플래시 메모리 기반 저장 장치와 관련된 연구를 제시한다. 3절은 병합 연산이라 불리는 FTL의 블록 재활용 기법에 대한 기본 지식을 제공한다. 4절에서는 이주 연산을 소개하고, 이주 연산과 병합 연산의 비용 함수를 정의한다. 5절에서는 블록 재활용 비용이 최소가 되는 이주/병합 순서의 거시적 최적화 해법을 설명한다. 6절에서는 Postmark 벤치마크와 임베디드 시스템 워크로드를 이용한 실험 결과를 보이고, 마지막으로 7절에서 본 논문의 결론을 맺는다.

2. 플래시 메모리의 특성 및 관련 연구

현재 NOR와 NAND 두 가지 유형의 플래시 메모리가 널리 사용되고 있다. NOR 플래시 메모리는 인터페이스가 기존 메모리와 유사하고, 바이트 단위로 접근이 가능하며, 데이타 읽기 속도가 빨라 주로 프로그램 코드 저장 및 수행에 사용된다. 반면에 NAND 플래시 메모리는 인터페이스가 기존 메모리와 상이하고, 페이지 단위로만 접근이 가능하며, 읽기를 위한 셋업 시간이 크기

때문에 프로그램 코드보다는 데이타를 저장하는 용도에 적합하다. 본 연구는 대용량 플래시 메모리 저장장치에 주로 사용되는 NAND 플래시 메모리를 적용 대상으로 한다. 그렇지만 FTL의 동작이 본 논문에 기술된 것과 유사한 경우 제안하는 기법들은 다른 유형의 플래시 메모리를 사용하는 저장 장치에도 적용될 수 있다.

NAND 플래시 메모리의 구체적인 특징은 다음과 같다. NAND 플래시 메모리는 동일한 크기의 블록들로 구성되며, 각 블록은 동일한 크기의 페이지들로 구성된다. 일반적으로 블록의 크기는 16KBytes부터 256KBytes이고, 페이지의 크기는 0.5KBytes부터 4KBytes이다. NAND 플래시 메모리에서 읽기/쓰기는 페이지 단위로 수행된다. SLC(Single-Level Cell) 플래시 메모리는 일반적으로 페이지 읽기에 25 μs가 소요되고 페이지 쓰기에는 300 μs가 소요된다. 플래시 메모리에 기록된 데이타의 변경에는 제약이 존재하므로, 사실상 블록 단위로 데이타를 소거하고 재기록 하여야 한다. 일반적으로 SLC 플래시 메모리에서 블록을 소거하는 데는 2 ms가 소요되고, MLC 플래시 메모리에서는 1.5 ms가 소요된다[4,5].

플래시 메모리에 데이타를 기록하기 위해서는 먼저 블록을 소거해야 한다. 그리고 데이타 기록은 페이지 단위로 수행되며, 이렇게 페이지에 기록된 데이타는 해당 페이지를 포함한 블록이 다시 소거되지 않는 한 변경에 제약이 따른다. 이러한 제약을 극복하기 위하여 FTL은 데이타가 변경되면 새로운 위치에 데이타를 기록하고 논리 주소를 물리 주소로 변환하는 맵을 갱신한다. 사상 단위에 따라 FTL은 블록 단위로 사상하는 방식과 섹터 또는 페이지 단위로 사상하는 방식으로 구분될 수 있다. 소용량 NOR 플래시 메모리 카드에서 사용되는 FTL은 주로 섹터 단위로 사상하며 그 동작 방식은 LFS(Log-structured File System)과 유사하다. 그런데 섹터 단위 사상은 블록 단위 사상과 비교하여 맵의 크기가 매우 크며, 클리닝(cleaning)과 같은 블록 재활용 기법의 효율에 따라 성능 변화가 심하다[3]. 이러한 문제를 극복하기 위하여 [6]에서는 자주 변경되는 데이타와 그렇지 않은 데이타를 구분하여 저장하였으며, 이러한 구분을 통하여 블록 재활용 효율을 높이고 성능을 향상시킬 수 있음을 보였다.

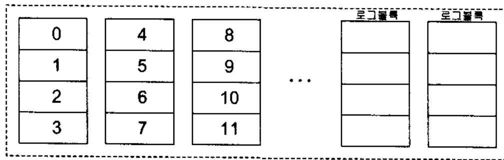
대용량 NAND 플래시 메모리를 사용하는 메모리 카드[7,8] 용 FTL 또는 임베디드 시스템 용 FTL(예, M-systems의 NFTL[9])에서는 주로 블록 단위 사상이 이용된다. 블록 단위 사상의 경우 맵의 크기가 작고, FTL의 동작 방식이 단순하여 저 사양의 내장형 시스템에서도 효율적으로 구현되며, 필요한 경우 하드웨어/소프트웨어 통합 설계를 통하여 성능 향상을 꾀할 수 있다는 장점이 있다. 현재 NAND 플래시 메모리를 사용하는 대용량 메모리 카드와 내장형 시스템의 경우 대부분 블록 사상 FTL이나 이의 변종을 사용한다. 본 연구에서 사용하는 FTL 역시 블록 사상을 사용하며 저 사양의 시스템에서도 효율적으로 동작하도록 설계되었다.

블록 사상 FTL은 순차 쓰기는 효율적으로 처리하지만 임의 쓰기(random write)와 반복 쓰기는 효율적으로 처리하지 못한다. 임의 쓰기와 반복 쓰기의 성능을 높이기 위하여 최근 블록 사상 기법을 기본으로 하면서, 로그 블록에 제한적으로 섹터 단위 사상을 도입한 FAST (Fully Associative Sector Translation) 기법[10]이 개발되었다. 이렇게 FAST 기법은 블록 사상 기법에 제한적으로 섹터 단위 사상 기능을 추가하여 임의 쓰기와 반복 쓰기의 성능을 향상시킨다. 그런데 본 논문이 제안하는 기법은 기존 블록 사상 기법의 변경을 최소화 하면서 반복 쓰기를 효율적으로 처리한다. 따라서 이 두 기법은 FTL을 설계할 때 경우에 따라 상호 보완적으로 적용될 수 있을 것이다.

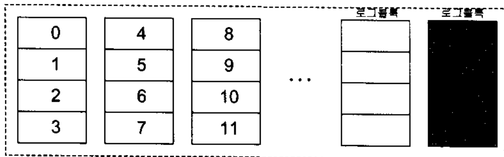
3. FTL의 동작과 병합 연산

앞에서 설명한 바와 같이 FTL은 상위에 있는 파일 시스템 계층에게 가상 블록 장치 인터페이스를 제공한다. 이를 위해 FTL은 플래시 메모리가 가상 섹터처럼 보이도록 환상을 제공하며, 이러한 가상 섹터는 언제든지 읽기와 갱신이 가능하다. 아울러 FTL은 내부적으로 블록 재활용 기법을 통하여 공간을 재사용하면서, 가상 섹터가 변경되면 새로운 위치에 기록하고 맵을 변경하여 재사상한다. 본 절에서는 새로운 블록 재활용 기법인 이주 연산을 소개하기 전에 먼저 기존 블록 재활용 기법인 병합 연산의 동작을 설명한다. FTL은 사상 기법과 블록 재활용 기법에 따라 종류가 다양한데, 본 논문에서 제시하는 최적화 기법들은 [11]과 같이 블록 단위로 사상하는 FTL을 대상으로 한다. 아울러 실험에서 사용하는 FTL은 블록 단위로 사상하는 FTL이며, 병합 연산의 동작은 [11]에서 소개된 병합 알고리즘과 유사하다. 실험에 사용된 FTL은 추가적으로 마모 평준화(wear-leveling)와 성능/신뢰성 향상 기법과 같은 복잡한 특징을 가지고 있지만, 이러한 특징들은 논문에서 소개하는 병합/이주 연산 및 그들의 비용 함수에 큰 영향을 미치지 않는다.

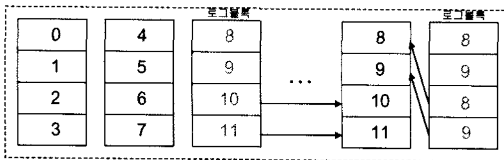
플래시 메모리의 어떤 블록에 기록된 섹터 데이타는 제자리에서 갱신이 불가능하기 때문에, FTL은 섹터에 대한 쓰기가 요청되면 먼저 임시 블록을 할당하고, 임시 블록의 미사용 페이지에 섹터 데이타를 기록한다. 앞으로 이러한 임시 블록을 로그 블록이라 부른다. 그림 1(b)에서, 8과 9번 섹터로 쓰기 요청이 반복되면, 이러



(a) 초기 상태



(b) 섹터 8과 9번이 반복적으로 기록된 후



(c) 병합 연산

그림 1 병합 연산

한 쓰기 요청은 로그 블록에 있는 페이지에 기록된다. 이후 로그 블록의 모든 페이지가 사용되면, FTL은 새로운 요청을 처리하기 전에 병합 연산을 수행하여 빈 로그 블록을 생성해야 한다. 병합 연산을 위하여 FTL은 항상 하나 이상의 미사용 로그 블록을 유지한다. 병합 연산은 먼저 미사용 로그 블록을 소거하고, 데이타가 기록된 로그 블록 또는 기존의 데이타 블록에서 새로 소거된 빈 로그 블록으로 모든 유효한 섹터들을 복사하는 작업으로 구성된다(그림 1(c)). 모든 섹터들이 복사되면, 새로운 로그 블록은 섹터 8~11을 위한 데이타 블록이 되고, 이전 데이타 블록과 이전 로그 블록은 미사용 로그 블록이 되어 다음 쓰기 요청이나 병합 연산을 위해 사용된다. 마지막으로 FTL은 관련된 맵을 변경하여 로그 블록과 데이타 블록의 변화를 반영한다.

만약 8~11 섹터가 모두 연속적으로 로그 블록에 쓰여졌으면, 교환 병합이라 불리는 최적화가 가능하다[11]. 이 경우 FTL은 8~11 섹터들이 기록된 로그 블록을 새로운 데이타 블록으로 지정하고, 이 섹터들이 있던 기존 데이타 블록을 미사용 로그 블록으로 지정하도록 사상 정보를 갱신한다. 즉 소거와 페이지 복사 없이 로그 블록과 데이타 블록의 역할을 바꾸도록 맵을 변경하는 작업만으로 블록을 재활용할 수 있다. 이러한 최적화를 적용하면 연속적으로 쓰여진 데이타의 경우 매우 적은 비용으로 블록 재활용이 가능하다.

그런데 상위 계층인 파일 시스템에서는 순차 쓰기 패턴뿐만 아니라 반복적인 쓰기 패턴도 쉽게 발견할 수

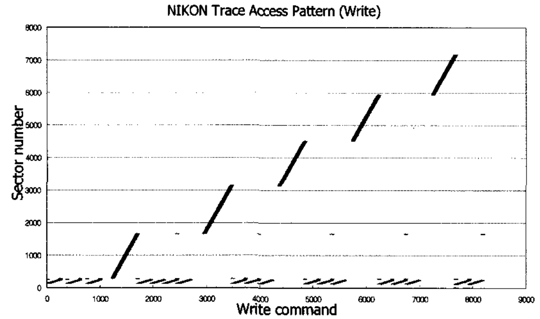


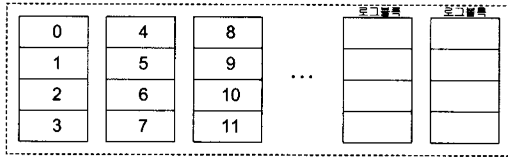
그림 2 니콘 카메라에서 동작하는 FAT 파일 시스템의 쓰기 패턴

있다. 그림 2에서 두 가지 형태의 쓰기 요청을 모두 볼 수 있는데, 첫째는 파일 데이타에 대한 순차 쓰기 요청이고, 둘째는 FAT과 디렉토리와 같은 메타데이타의 갱신을 위한 반복적인 쓰기 요청이다. 이러한 복합적인 쓰기 패턴은 많은 파일 시스템에서 일반적이라고 생각되며, 순차 쓰기 이외에도 반복적인 쓰기에 대한 최적화가 필요하다는 점을 보여준다.

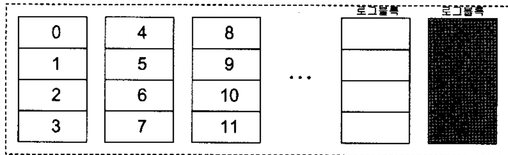
4. 이주 연산과 비용 모델

이 절에서는 반복적인 쓰기 패턴에 효율적인 블록 재활용 기법인 이주 연산을 설명한다. 또한 병합 연산과 이주 연산의 비용 모델을 제시하고, FTL이 블록을 재활용할 때 두 연산의 비용을 고려하여 비용이 작은 연산을 선택하는 기법에 대해 설명한다.

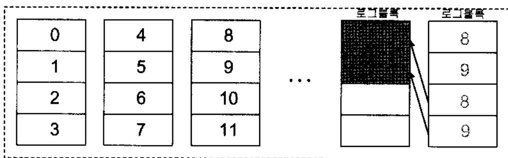
쓰기 요청이 몇몇 섹터에 집중되면, 이들 섹터들은 로그 블록에 반복적으로 기록된다. 그리고 로그 블록의 페이지가 모두 사용되면, FTL은 병합 연산을 수행하여 미사용 로그 블록들을 생성하며, 미사용 로그 블록들은 이후 새로운 쓰기 요청을 처리하기 위해 사용된다. 그런데 그림 3(b)를 보면, 로그 블록에 두 개의 섹터가 반복적으로 기록되어 있다. 이렇게 로그 블록에 상대적으로 적은 수의 섹터들이 반복적으로 기록되어 있다면, 기존의 병합 연산이 아닌 새로 소개하는 이주 연산을 적용하여 블록 재활용 비용을 줄일 수 있다. 새로운 미사용 로그 블록을 소거하고 기존 로그 블록과 데이타 블록으로부터 모든 페이지들을 복사해야 하는 병합 연산과 달리, 이주 연산은 새로운 미사용 로그 블록을 소거하고 기존 로그 블록에서 새로운 로그 블록으로 유효 페이지들만(그림 3(c)에서는 2개의 페이지)을 복사한다. 복사 후 이전 로그 블록은 미사용 로그 블록이 되고, 데이타가 복사된 로그 블록은 8~11 섹터를 위한 새로운 쓰기 버퍼 영역이 되어, 이후의 쓰기 요청은 새 로그 블록의 미사용 페이지에 기록된다.



(a) 초기 상태



(b) 섹터 8번과 9번이 반복적으로 기록된 후



(c) 이주 연산

그림 3 이주 연산

플래시 메모리의 블록에 있는 페이지의 개수를 N_p로 하고, 이주 연산에서 복사되는 유효한 페이지 개수를 p (그림 3(c)에서 p는 2)라고 하자. 이주 연산은 이전 로그 블록에서 새 로그 블록으로 p개의 페이지들을 복사하기 때문에 이주 연산 후 새 로그 블록에는 N_p-p 개의 가용 페이지가 존재한다. 일반적으로, 병합 연산은 N_p개 페이지를 복사하므로 복사 비용이 크다. 그렇지만, 연산 수행 후 새 로그 블록에 N_p 개의 가용 페이지를 얻는다. 반대로 이주 연산은 $p(<N_p)$개의 페이지를 복사하므로 복사 비용이 작다. 그렇지만 이주 연산 수행 후 새 로그 블록에서 사용 가능한 페이지는 N_p-p개로 병합 연산의 결과보다 작다. 따라서 주어진 상황에서 이주 연산과 병합 연산 중 어느 연산이 비용 대비 이득이 큰지에 대한 비용/이득 분석이 필요하며, 아울러 FTL은 상황에 따라 이러한 분석을 통하여 이주/병합 연산을 선택하여야 할 것이다.

이주 연산과 병합 연산 중 가용 페이지 생성 비용이 작은 연산을 구하기 위해서는 두 연산에 대한 비용 모델이 필요하다. 이를 위해 먼저 C_E는 블록 소거 비용(또는 시간), 그리고 C_{cp}는 한 블록에서 다른 블록으로 페이지를 복사하는 비용(또는 시간)으로 정의한다. 몇몇 NAND 플래시 메모리 칩은 카피백(copyback) 연산을 제공하는데, 이 연산은 플래시 메모리 칩 안에 있는 버퍼를 통해서 어떤 블록의 페이지를 다른 블록의 페이지로 복사한다. 카피백 연산을 지원하는 칩의 경우 페이지 복사가 단일 카피백 연산으로 가능하므로 C_{cp}는 카피백 연산 비용이며, 그렇지 않은 칩의 경우 페이지를 읽고 쓰는 두 연산을 통해 페이지를 복사해야 하므로 C_{cp}는 페이지 읽기와 쓰기 연산 비용의 합이다.

가용 페이지 당 병합 비용을 계산하기 위해, 그림 1(c)에서 병합 연산이 끝난 후 섹터 8에 대한 쓰기가 다시 요청되었다고 가정하자. 섹터 8에 대한 쓰기 요청을 처리하기 위해 FTL은 빈 로그 블록을 할당하고 그 로그 블록을 C_E의 비용으로 소거한다. 그리고 섹터 8~11에 대한 모든 쓰기 요청들은 새롭게 할당된 로그 블록에 있는 N_p개의 미사용 페이지를 이용하여 처리된다. 다수의 쓰기가 요청되어 로그 블록의 모든 페이지들이 사용되면, FTL은 다시 병합 연산을 수행해야 하는데, 이는 새로운 빈 로그 블록을 C_E의 비용으로 소거하고, 이전 데이타 블록이나 로그 블록에 있는 유효한 N_p 개의 페이지들을 N_pC_{cp} 의 비용으로 복사함으로써 수행된다. 그리고 연관된 사상 정보가 갱신되어, 유효 데이타가 복사된 새 로그 블록은 섹터 8~11을 위한 새로운 데이타 블록이 되고, 기존 데이타 블록과 기존 로그 블록은 모두 빈 로그 블록이 된다. 사상 정보가 갱신되면 블록 라이프 사이클의 주기(병합 연산들 사이의 간격)가 다시 시작되며, FTL은 다음에 오는 섹터 8~11에 대한 쓰기 요청들을 처리하기 위해 새 로그 블록을 할당할 것이다. 결과적으로 FTL은 병합 비용 C_{merge}를 지불하고 Np 개의 가용 페이지를 각 주기마다 얻는다. 비용과 이득을 분석하면 FTL은 각 주기 당 병합 비용 $2C_E$ + N_pC_{cp}를 지불하고, 병합의 이득으로 N_p개의 가용 페이지를 얻는다. 이제 가용 페이지 당 병합 비용은 다음과 같이 정의할 수 있다.

$$W_{merge} = \frac{C_{merge}}{N_p} = \frac{2C_E + N_pC_{cp}}{N_p}$$

가용 페이지 당 이주 비용을 계산하기 위해, 그림 3(c)에서 이주 연산이 끝난 후 섹터 8에 대하여 다시 쓰기가 요청되었다고 가정하자. 이전 이주 연산의 결과 FTL은 로그 블록에 N_p-p개의 가용 페이지를 가지고 있다. 따라서 FTL은 섹터 8~11에 대한 요청이 오면 최대 N_p-p개의 쓰기 요청을 이 로그 블록으로 처리할 수 있으며, N_p-p개의 가용 페이지가 모두 사용되면, 이주 연산이나 병합 연산을 수행해야 한다. 로그 블록에 유효한 페이지 개수는 여전히 p개이며, FTL이 이주 연산을 수행한다고 가정한자. 이주 연산은 C_E의 비용으로 새로운 빈 로그 블록을 소거하고, p개의 유효한 페이지들을 이전 로그 블록에서 새 로그 블록으로 pC_{cp}의 비용을 지불하고 복사함으로써 수행된다. 페이지를 복사한 후 FTL은 사상 정보를 갱신하여, 새로운 로그 블록을 섹터 8~11을 위한 로그 블록으로 지정하고 기존 로그 블록은 미사용 로그 블록으로 지정한다. 이와 같이 이주

연산이 수행되면, FTL은 새로운 로그 블록에 N_p-p개의 가용 페이지를 얻게 되며, 블록 라이프 사이클의 새로운 주기가 시작된다. 결과적으로, FTL은 이주 비용 C_{mig}(=C_E+pC_{cp})를 지불하고 N_p-p 개의 가용 페이지를 얻는다. 정리하면, 가용 페이지 당 이주 비용을 다음과 같이 정의할 수 있다.

$$W_{mig} = \frac{C_E + p \cdot C_{cp}}{N_p - p}$$

이주와 병합 연산에 대한 비용 모델은 사상 정보 갱신 비용을 고려하지 않는데, 그 이유는 두 연산 모두 사상 정보 갱신 비용이 거의 동일하며, 사상 정보 갱신 비용은 전체 블록 재활용 비용에서 매우 적은 부분을 차지하기 때문이다. 예를 들면, 실험에 사용한 FTL은 갱신이 필요한 사상 정보를 집중시키는 최적화가 적용되었으며, 그 결과 블록 재활용 후 한 개의 사상 정보 페이지만을 기록하면 된다.

이제 FTL은 이러한 비용 모델을 사용하여, 블록을 재활용할 때 어떤 연산을 적용할지 선택해야 한다. 비용 모델을 보면 가용 페이지 당 이주 비용(W_{mig})은 이주할 때 복사되는 페이지 개수 p에 의존적이다. 만약 p가 크다면 W_{mig}가 W_{merge}보다 크게 되며, p가 작으면 그 반대가 된다. 그리고 W_{mig}와 W_{merge}가 동일한 *equilibrium* p가 존재하며, 아래와 같이 계산된다. 모델이 제시하는 가용 페이지당 비용을 고려하면, 만약 p가 *equilibrium* p보다 작으면 이주 연산이 효과적이며, 그 반대의 경우 병합 연산이 효과적이다.

$$\frac{2C_E + N_p C_{cp}}{N_p} = \frac{C_E + p \cdot C_{cp}}{N_p - p}$$

$$equilibrium\ p = \frac{N_p}{2}$$

5. 이주/병합 순서의 거시적 최적화

가용 페이지당 비용을 기반으로 병합과 이주 연산을 선택하는 기능을 FTL에 구현하고 Postmark 벤치마크를 통해 성능을 측정하였다. 그런데 Postmark 벤치마크를 통해 측정한 비용 기반 선택 기법의 성능 향상 정도(6절 참고)는 기대보다 작았다. 그리고 실험 도중 p가 *equilibrium* p보다 훨씬 작을 때 이주 연산 대신 병합 연산을 적용하면 성능이 더 좋아지는 현상을 관찰할 수 있었으며, 이러한 관찰을 통하여 전체 블록 재활용 비용을 최소화 하는 이주/병합 순서의 거시적 최적화가 존재함을 알게 되었다.

거시적 최적화를 설명하기 위해서는 먼저 이주 연산이 복사하는 페이지 개수 p 값의 변화를 관찰해야 한다. 그리고 이주 연산이 복사하는 페이지 개수 p는 이주 연산이 반복됨에 따라 변화하기 때문에 이러한 변화를 반영하는 함수 P_n을 정의해야 한다. 이전 로그 블록에서 새 로그 블록으로 p개의 페이지를 복사하는 이주 연산이 수행되고 나면 새 주기가 시작된다. 다음 주기에서 로그 블록에 이미 존재하는 섹터들만 다시 기록되면, 다음 이주 연산은 여전히 p개의 페이지들만 복사할 것이다. 반면에, 로그 블록에 존재하지 않는 섹터에 대한 쓰기가 요청되면, 로그 블록에는 새로운 섹터가 진입하여, 결과적으로 다음 이주 연산은 $p+1$개의 페이지를 복사할 것이다. 모든 경우에, 병합 연산이 아닌 이주 연산이 계속 수행된다면 복사되는 페이지의 개수는 동일하거나 증가하며, 특히 새로운 섹터가 로그 블록으로 진입하면 복사되는 페이지의 개수가 증가한다. 이러한 분석 내용을 확인하기 위하여, 비용 기반 이주/병합 선택 기능을 가진 FTL 상단에 Postmark 벤치마크를 실행시키면서, FAT과 루트 디렉토리 섹터들이 포함된 블록에서 이주 연산을 수행할 때 복사되는 페이지 개수를 관찰하였다. 그림 4는 이주 연산이 수행될 때 복사되는 페이지 개수의 변화를 보여주는데, 관찰 결과는 이주 연산이 연속적으로 수행될 때 증가율 α를 가진 단조 증가 함수 P_n(n은 이주 연산 수행 회수)으로 복사되는 페이지 개수를 표현할 수 있음을 보여준다.

$$P_n = \alpha \cdot n$$

이제 병합 연산과 이주 연산의 거시적 주기를 살펴보자. 먼저 병합 연산이 끝난 직후, 로그 블록에는 N_p개의

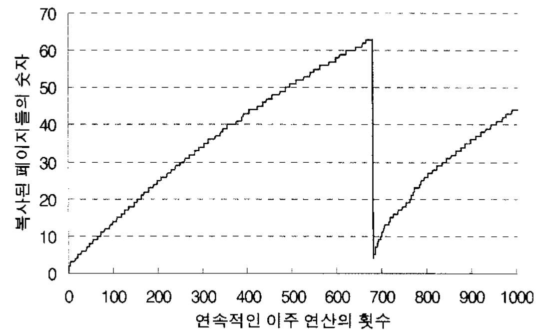


그림 4 이주 연산 중에 복사되는 페이지 수

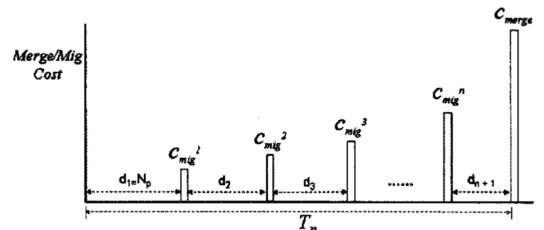


그림 5 이주/병합의 거시적 주기(T_n)

가용 페이지가 존재하며, 새로운 (미시적) 주기가 시작된다. 새로운 주기 동안 섹터에 대한 쓰기 요청은 해당 섹터를 위한 로그 블록에 기록되며, 로그 블록의 모든 페이지가 사용되면, FTL은 이주 또는 병합 연산을 수행하여 블록을 재활용해야 한다. FTL은 전체 블록 재활용 비용이 최소가 되도록 n번 이주 연산을 수행한다고 가정하자. 첫 번째 이주 연산에서 로그 블록에 유효한 페이지 개수가 P_1이면, 이주 연산은 P_1개의 페이지를 복사할 것이며, 이주 연산 수행 후 새로운 로그 블록에는 N_p-P_1개의 가용 페이지가 존재한다. 그리고 새로운 주기가 시작되며, 이 주기의 마지막에 다시 이주 연산이 실행되고, 이주 연산은 P_2개의 페이지를 복사하여 새로운 로그 블록에 N_p-P_2개의 가용 페이지를 생성한다. 이렇게 이주 연산을 n번 수행한 후 FTL은 다음에 병합 연산을 수행해야 전체 블록 재활용 비용이 최소가 되기 때문에 병합 연산을 수행할 것이다. 병합 연산이 끝난 후 로그 블록에는 N_p개의 가용 페이지가 존재하고 새로운 거시적 주기가 시작된다. 새로운 거시적 주기가 시작되면 다시 여러 번의 이주 연산을 수행한 후 마지막에 병합 연산을 수행하여 FTL은 거시적 주기를 끝낸다. 이주/병합 순서의 거시적 최적화는, P_n이 주어질 때, 이주 연산을 몇 번 수행한 후 병합 연산을 수행해야 전체 이주/병합 비용(블록 재활용 비용)이 최소가 되는지를 밝혀내는 것으로, 앞의 예에서 병합 연산을 적용하기 전에 반복된 이주 연산 회수 n을 구하는 것이다.

거시적 최적화 해를 구하기 위하여 거시적 주기에서 가용 페이지당 블록 재활용 비용($W(n)$)을 정의해야 한다. 먼저 P_n이 주어졌을 때 n 번째 이주 연산 비용인 $C_{mig}^{\ n}$ 은 다음과 같이 정의될 수 있다.

$$C_{mig}^{\ n} = P_n C_{cp} + C_E = \alpha \cdot n \cdot C_{cp} + C_E$$

$(n-1)_{th}$ 번째 이주 연산이 $C_{mig}^{\ n-1}$의 비용으로 수행된 후 N_p-P_{n-1} 개의 가용 페이지가 생성되며, d_n은 $(n-1)$번째 이주 연산 후 생성된 가용 페이지 개수를 의미한다.

$$d_n = N_P - P_{n-1} = N_P - \alpha(n-1),\ \ n >= 1$$

이제 이주/병합 순서의 거시적 주기인 T_n에서 가용 페이지 당 이주/병합 비용인 $W(n)$은 다음과 같이 정의된다.

$$W(n) = \frac{\sum_{k=1}^{n} C_{mig}^{\ k} + C_{merge}}{\sum_{k=1}^{n+1} d_k} = \frac{\frac{\alpha \cdot C_{cp}}{2} n^2 + \left(\frac{\alpha \cdot C_{cp}}{2} + C_E\right) n + C_{merge}}{-\frac{1}{2}\alpha \cdot n^2 + \left(N_P - \frac{1}{2}\alpha\right) n + N_P}$$

이주/병합 순서의 거시적 최적화는 $W(n)$을 최소로 하는 n을 찾는 것이기 때문에 $W(n)$을 미분하여 $W'(n)$을 구하고, $W'(n_0) = 0$이 되는 n_0를 구한다.

$$W'(n) = \frac{\alpha \cdot \left(\frac{1}{2}C_E + \frac{1}{2}C_{cp}N_P\right) n^2 + \alpha \cdot \left(C_{cp}N_P + C_{merge}\right) n + \frac{\alpha \cdot C_{cp}N_P + \alpha \cdot C_{merge}}{2} + N_P C_E - N_P C_{merge}}{\left(-\frac{1}{2}\alpha \cdot n^2 + \left(N_P - \frac{1}{2}\alpha\right) n + N_P\right)^2}$$

$$n_0 = \frac{-\alpha \cdot \left(C_{cp}N_P + C_{merge}\right) \pm \sqrt{\alpha^2 \cdot \left(C_{cp}N_P + C_{merge}\right)^2 - \alpha \cdot \left(C_E + C_{cp}N_P\right)\left(\alpha \cdot C_{cp}N_P + \alpha \cdot C_{merge} + 2N_P C_E - 2N_P C_{merge}\right)}}{\alpha \cdot (C_E + C_{cp}N_P)}$$

이 해법은 이주 연산을 n0 번 수행한 후 병합 연산을 적용하면 거시적 주기에서 가용 페이지 당 블록 재활용 비용이 최소가 됨을 의미한다.

6. 비용 기반 선택과 거시적 최적화의 실험 결과

MP3 플레이어와 캠코더에서 사용되는 블록 사상 FTL에 비용 기반 이주/병합 선택 기능과 거시적 최적화 기능을 구현하였다. 거시적 최적화를 위하여 FTL은 블록에 대하여 이주 연산이 수행될 때 복사되는 페이지 개수 p의 정보를 수집하고, 수집된 정보로부터 증가율 α를 계산한다. 특별히 언급하지 않으면 실험할 때 로그 블록의 개수는 8개이며, 로그 블록 개수에 따른 성능 변화는 6.3절에서 설명한다.

FTL 상단에 MS-DOS FAT 호환 파일 시스템을, 그리고 파일 시스템 상단에 테스트 프로그램을 수행시키면서 비용 기반 이주/병합 선택 기법과 거시적 최적화 기법의 성능을 측정하였다. 테스트 프로그램으로는 두 종류가 사용되었는데, 그 중 하나는 Postmark 벤치마크 프로그램이다. Postmark 벤치마크 프로그램은 인자를 조절하여 디렉토리 생성/삭제와 파일 생성/삭제 등 다양한 종류의 워크로드를 생성할 수 있으며, 트랜잭션 회수를 조절하여 장기간 성능 측정이 가능하다. 또 다른 테스트 프로그램은 [12]에서 사용된 워크로드로서, 앞으로 이를 임베디드 시스템 워크로드라 부른다. 임베디드 시스템 워크로드는 세 종류의 워크로드로 구성되며, 각각은 플래시 메모리를 저장장치로 사용하는 팩스, 휴대전화, 그리고 이벤트 레코더에서 수집된 것이다.

실험에서 FTL은 플래시 메모리 에뮬레이터 상에서 동작한다. 플래시 메모리 에뮬레이터는 플래시 메모리 데이타 시트에 제시된 각 플래시 메모리 연산들의 시간을 이용하여 전체 실행 시간을 측정하는 기능을 가지고 있다. 플래시 메모리 에뮬레이터가 측정한 실행 시간의 정확성을 확인하기 위해 [13]에서는 200Mhz S3C2410 (ARM920T 코어) 프로세서와 128MBytes NAND 플래시 메모리 칩을 가진 임베디드 보드에서 실행 시간을 측정하였고, 이 결과를 플래시 메모리 에뮬레이터와 비교하였다(그림 6)[1]. 실제 보드에서 측정한 실행 시간은 플래시 메모리 연산 수행 시간 이외에도 테스트 프로그램, 파일 시스템, 그리고 FTL 코드 실행 시간을 모두 포함하고 있기 때문에 에뮬레이터의 시간보다 클 것으로 예상되었다. 실제로 그림 6을 보면 실제 보드의 파일 읽기 시간이 에뮬레이터의 파일 읽기 시간보다 더 크다. 그렇지만 놀랍게도, 실제 보드에서의 파일 쓰기 시간은 에뮬레이터와 거의 동일한 결과를 보여주었다. 이러한 결과가 나온 이유는, 실제 플래시 메모리 칩의 페이지 읽기 시간은 데이타 시트가 제시하는 읽기 시간과 거의 같지만 실제 플래시 메모리 칩의 페이지 쓰기나 블록 소거 시간은 데이타 시트 시간보다 작기 때문이다. 즉, 쓰기의 경우 실제 보드에서는 에뮬레이터보다 코드 실행 시간이 더 걸리지만 플래시 메모리 연산 시간은 더 작다. 그리고 두 효과가 서로 상쇄되어 쓰기의 경우 실제 보드와 에뮬레이터의 결과가 거의 동일하다. 이러한 실험 결과를 바탕으로, 읽기의 경우 에뮬레이터 실행 시간은 실제 보드 실행 시간보다 작으며, 쓰기의 경우 실제 보드와 거의 동일하다는 것을 알 수 있다.

실험에 사용된 FTL은 가격이 싸고 용량이 커서 최근 MP3 플레이어에 널리 사용되는 MLC(Multi-Level Cell) 플래시 메모리 지원을 목표로 개발되었다. 따라서

그림 6 실제 보드와 에뮬레이터의 실행 시간 비교[13]

1) 실험에 사용된 임베디드 보드는 플래시 메모리 칩을 연결하는 버스와 시스템버스 사이에 브리지를 거쳐 데이타를 전송하므로 버스 전송 시간이 매우 크다. 따라서 에뮬레이터와 실제 보드의 비교 실험에서 버스 전송 시간은 실제 보드에서 측정한 값을 적용하였으며, 그 이외에 플래시 메모리 연산 시간은 데이타 시트에서 제시된 값을 사용하였다.

표 1 MLC 플래시 메모리의 연산 시간[5]

플래시 메모리 연산	실행 시간 (ms)
페이지 읽기 (2112 bytes)	0.113
페이지 쓰기 (2112 bytes)	1.013
페이지 복사 (2112 bytes)	1.128
블록 소거	1.5

본 논문에서는 플래시 메모리 에뮬레이터를 MLC 플래시 메모리와 동일하게 동작하도록 설정하였다. 설정 대상 MLC 플래시 메모리 칩의 연산 실행 시간은 표 1과 같으며, 이 시간들은 해당 칩의 데이타 시트가 제시하는 것이다.[2]

테스트 프로그램을 이용하여 1) 병합 연산만을 사용하는 기법, 2) 비용 기반 이주/병합 선택 기법, 3) 비용 기반 이주/병합 선택 기법을 바탕으로 연속적인 이주 연산 회수에 상한을 두는 기법, 그리고 4) 거시적 최적화 기법을 실험하였다. 이 네 가지 기법을 각각 "병합", "병합+이주", "병합+이주(LIMIT)", "병합+이주(거시적 최적화)"로 표시한다. "병합+이주(LIMIT)" 기법에서 연속적인 이주 연산 회수 상한은 *Np/2*로 설정하였으며, 연속적인 이주 연산 회수가 상한에 도달하면 병합 연산을 수행하도록 하였다. 실험 결과에 따르면 이렇게 연속적인 이주 연산 회수에 상한을 설정하는 기법은 구현이 간단하면서도 경우에 따라 거시적 최적화에 근접한 성능을 보여준다.

6.1 Postmark 벤치마크 실험

Postmark 벤치마크는 설정에 따라 다양한 크기의 파일/디렉토리 생성/삭제 트랜잭션을 실행한다. 실험에서는 설정을 조절하여 "디렉토리/작은 파일 워크로드"와 "큰 파일 워크로드"에 대한 성능을 측정하였다. "디렉토리/작은 파일 워크로드"의 경우 Postmark 벤치마크는 디렉토리와 512Bytes 크기의 파일 50개를 생성하고 삭제하는 트랜잭션을 수행한다. 그리고 "큰 파일 워크로드"는 1MBytes 크기의 파일을 생성하고 삭제하는 트랜잭션으로 구성된다.

그림 7을 보면 "디렉토리/작은 파일 워크로드"를 실행할 때, "병합" 기법에 비해 "병합+이주" 기법이 플래시 메모리 저장 장치의 성능을 5% 향상시킴을 알 수 있다. 그리고 "병합+이주(LIMIT)" 기법과 "병합+이주(거시적 최적화)" 기법은 거의 유사한 성능을 보이며, 이 두 기법은 "병합" 기법에 비해 26~27% 정도 성능을 향상시킨다. 실험에 적용된 MLC NAND 플래시 메모리의 경우 N_p가 128이고 *equilibrium p*는 64이며,

2) 버스 전송 시간은 개발 보드에 따라 다를 수 있고, 실험 초점을 플래시 메모리 연산 실행 시간에 맞추기 위하여, 에뮬레이터를 이용한 실험에서는 버스 전송 시간을 측정 대상에서 제외하였다.

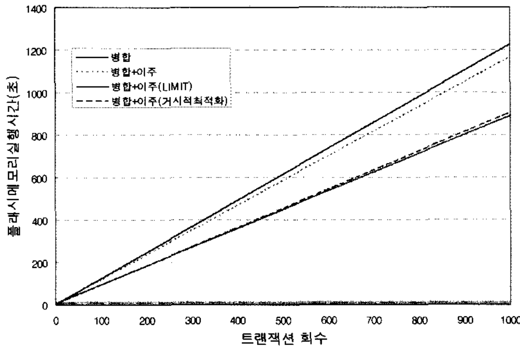


그림 7 Postmark 벤치마크를 이용한 디렉토리와 작은 파일 생성/삭제 트랜잭션 실험 결과

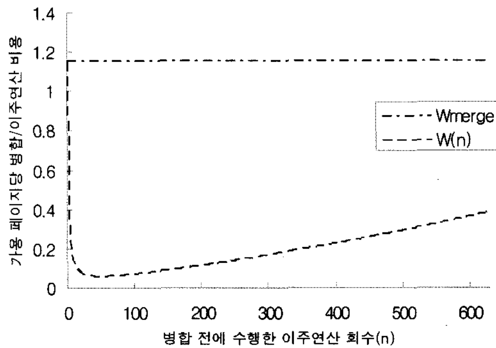


그림 8 이주/병합 순서에 따른 가용 페이지 당 블록 재활용 비용(a=0.1)

따라서 "병합+이주" 기법은 p가 64보다 작으면 이주 연산을 적용하고 그렇지 않으면 병합 연산을 적용한다. 그렇지만 "병합+이주(거시적최적화)" 기법은 복사 페이지 개수 p가 64보다 훨씬 작을 때 병합 연산을 적용함을 관찰할 수 있다. 예를 들어 Postmark 벤치마크를 실행시킬 때 FAT과 디렉토리 섹터가 존재하는 블록을 살펴보면, p의 증가율 a는 0.1로 관찰되며, 이 경우 n_0는 49로 계산된다. 따라서 거시적 최적화에 따르면 FTL이 49번 이주 연산을 적용한 후 (복사되는 페이지 개수가 약 5개 정도일 때) 병합 연산을 적용하면 블록 재활용 비용이 최소가 된다.

이주/병합 순서의 거시적 최적화 기법의 이점을 구체적으로 알아보기 위하여, 병합만을 적용할 때 가용 페이지당 블록 재활용 비용(W_{merge})과 이주/병합 순서에 따른 가용 페이지당 블록 재활용 비용($W(n)$)을 비교하였다(그림 8). $W(n)$은 n번 이주 연산을 적용한 후 병합 연산을 적용할 때 가용 페이지 당블록 재활용 비용을 의미한다(a=0.1). 그림 8을 보면, 병합만을 적용하는 경우 당연히 n 값과 무관하게 동일한 가용 페이지 당 블록 재활용 비용(W_{merge})을 보인다. 그렇지만 이주/병합 순서의 경우 n이 증가함에 따라 $W(n)$은 감소하여, 최소가 되는 지점을 통과한 후, 다시 완만하게 증가한다. 특히 n이 49(=n_0)가 되었을 때 $W(n)$은 최소가 되는데, 이는 FTL이 49번 이주 연산을 실행한 후 병합 연산을 수행하면 가용 페이지 당 블록 재활용 비용이 최소가 됨을 의미한다. 그리고 이 경우 병합 만을 적용할 때 비용의 5%만을 지불하면서 블록 재활용이 가능하다. 이러한 결과를 통하여 거시적 최적화가 블록 재활용 비용을 상당히 감소시킴을 확인할 수 있다.

W_{merge}는 병합만을 적용한 기법의 가용 페이지 당 블록 재활용 비용을 나타내고 $W(n)$은 n번 이주 연산을 적용한 후 병합 연산을 적용했을 때 가용 페이지 당 블록 재활용 비용을 나타낸다.

그림 7을 보면 연속된 이주 연산 횟수 상한을 임의의 값($N_p/2$)으로 고정한 "병합+이주(LIMIT)" 기법의 성능이 "병합+이주(거시적최적화)" 기법의 성능보다 미세하게 좋음을 알 수 있다. 이 실험에서 이주 연산 횟수 상한을 임의로 $N_p/2$로 정하였는데, 이 상한 값을 변경하면서 실험한 결과 $N_p/2$ 값은 "디렉토리/작은 파일 워크로드"에서 가장 좋은 성능을 보이는 오프라인 최적 값임이 밝혀졌다. 그리고 오프라인 최적 값으로 이주 연산 횟수 상한을 설정한 "병합+이주(LIMIT)" 기법은 이주 연산 횟수를 온라인으로 결정하는 "병합+이주(거시적최적화)" 기법과 비교하여 거의 같거나 미세하게 좋은 성능을 보인다. 그러나 "디렉토리/작은 파일 워크로드"의 경우 임의로 설정한 이주 연산 횟수 상한 값($N_p/2$)이 우연히 오프라인 최적 값과 일치하지만, 워크로드가 달라지면 이러한 오프라인 최적 값은 달라진다. 따라서 워크로드의 특성을 무시한 채 연속된 이주 연산 횟수 상한을 임의의 값으로 고정하는 "병합+이주(LIMIT)" 기법은 워크로드의 특성을 파악하여 동작하는 "병합+이주(거시적최적화)" 기법보다 좋지 않은 성능을 보일 수 있으며, 다음 절에서 설명하는 임베디드 시스템 워크로드에서 이러한 차이를 확인할 수 있다.

그림 9는 Postmark 벤치마크가 "큰 파일 워크로드"를 수행할 때 각 기법들의 성능을 보여준다. 본 논문에서 제시한 이주 연산은 메타 데이타나 작은 파일이 반복적으로 기록되는 경우 성능 향상을 목표로 한다. 그런데 순차적으로 파일 데이타가 기록되면 데이타를 기록할 때 이주 연산이 실행될 가능성이 거의 없다. 그렇지만, 파일을 순차적으로 기록하는 경우에도 FAT이나 디렉토리와 같은 메타 데이타의 갱신이 수반되며, 이러한 메타 데이타 갱신은 이주 연산으로 성능이 향상될 수 있다. 실제로 그림 9에서 보듯이 전체적인 성능 향상 정도는 "디렉토리/작은 파일 워크로드"에 비해 작지만,

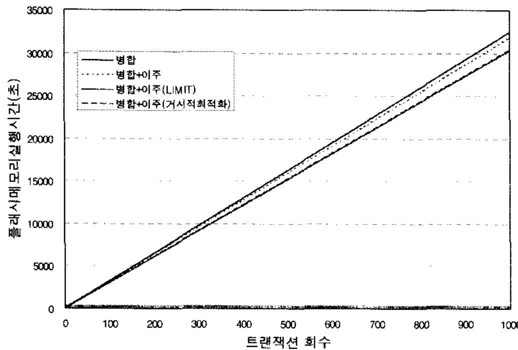


그림 9 Postmark 벤치마크를 이용한 큰 파일 생성/삭제 트랜잭션 실험 결과

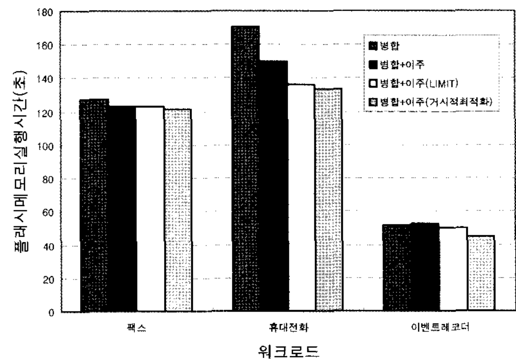


그림 10 임베디드 시스템 워크로드를 이용한 실험 결과

"병합", "병합+이주", "병합+이주(LIMIT)", 그리고 "병합+이주(거시적최적화)" 기법 순으로 성능이 좋아짐을 알 수 있다.

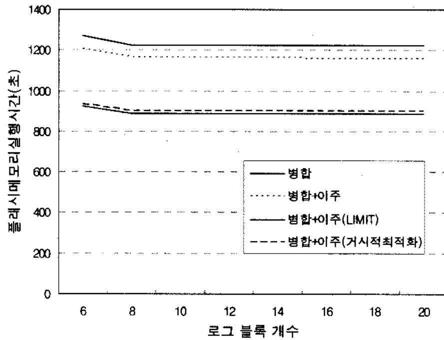
6.2 임베디드 시스템 워크로드 실험

그림 10은 팩스, 휴대전화, 그리고 이벤트 레코더 워크로드를 실행할 때 네 가지 기법의 성능을 표시한 것이다. 이벤트 레코더에서 "병합"과 "병합+이주" 기법 간의 성능 순서를 제외하고 모든 경우에 "병합" 기법이 가장 성능이 나쁘며, "병합+이주", "병합+이주(LIMIT)", 그리고 "병합+이주(거시적최적화)" 순으로 성능이 향상된다. 각 워크로드 별로 성능 향상 정도를 살펴보면 "병합+이주(거시적최적화)"는 "병합" 기법에 비하여 휴대전화 워크로드의 경우 22%, 이벤트 레코더의 경우 14%, 팩스의 경우 5% 정도 성능을 향상시킨다.

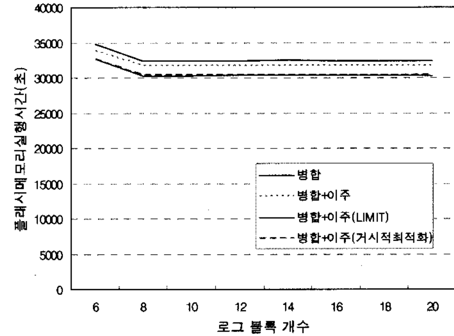
이벤트 레코더 워크로드의 경우 "병합+이주" 기법이 "병합" 기법에 비하여 미세하게 나쁜 성능을 보이는데, 그 이유는 "병합+이주" 기법이 워크로드 변화에 적응하지 못하기 때문이다. 실험에서 이벤트 레코더 워크로드를 실행하기 전에 파일 시스템을 포맷하는데, 포맷을 수행하면서 모든 FAT 섹터들이 갱신되고, 그 결과 FAT 테이블이 존재하는 블록을 위한 로그 블록에 이미 $N_p/2$에 가까운 유일한 페이지들이 기록되어 있었다. 그리고 다음에 수행되는 이벤트 레코더 워크로드는 소수의 파일을 반복적으로 변경하며, 이 때 로그 블록에 존재하는 페이지 중 소수의 페이지만 계속적으로 변경된다. 결과적으로 이벤트 레코더 워크로드는 소수의 페이지만 반복적으로 변경하지만, 이주 연산시 복사되는 페이지 개수를 관찰해 보면, $N_p/2$에 근접한 개수의 페이지들이 계속해서 복사된다. 결국 매 주기마다 한 번도 변경되지 않은 페이지들이 매 이주 연산마다 복사될 뿐 아니라 새로운 로그 블록에서 가용 공간을 줄여 블록 재활용 작업 수행 빈도를 증가시키고 맵 변경과 같은 오버헤드를 누적시키다. 만약 해당 블록이 병합 연산을 수행하였다면, 이렇게 더 이상 변경되지 않는 페이지들은 데이타 블록으로 이동하고, 로그 블록에는 그 이후 변경된 페이지들만 존재하여, 이주 연산시 복사되는 페이지 개수는 매우 적을 것이다. 이러한 관찰 결과는 결국 워크로드가 변하면 이주 연산들 사이에 병합 연산이 수행될 필요가 있음을 보여주며, 이를 수행하는 "병합+이주(LIMIT)" 기법과 "병합+이주(거시적최적화)" 기법의 성능이 "병합+이주" 기법보다 좋은 이유를 설명한다. 그리고 워크로드의 특성에 무관하게 상수로 정의된 회수 만큼 이주 연산이 수행되면 무조건 병합 연산을 수행하는 "병합+이주(LIMIT)" 기법보다 워크로드의 특성을 파악하여 최적의 이주 연산 수행 회수를 결정하는 "병합+이주(거시적최적화)" 기법의 성능이 좋다. 그렇지만, 구현의 용이성을 고려하면, 워크로드의 특성이 알려진 경우 "병합+이주(LIMIT)" 기법도 실용성이 높은 기법이라 할 수 있다.

6.3 로그 블록 개수에 따른 성능 변화

블록 사상을 기본으로 하면서 로그 블록에 제한적으로 섹터 단위 사상을 적용한 FAST 기법의 경우 성능은 로그 블록 개수에 따라 변화한다. 그렇지만 일반적으로 블록 사상 FTL의 경우 로그 블록의 개수가 일정 수준 이상이면 로그 블록의 개수에 따른 성능 변화는 크지 않다. 로그 블록 개수가 변할 때 이주 연산이 추가된 FTL의 성능을 측정하기 위하여, 로그 블록 개수를 변화시키면서 Postmark 벤치마크와 임베디드 시스템 워크로드의 성능을 측정하였다(그림 11과 12). 성능 측정 결과에 따르면, 임베디드 시스템 워크로드에서 이벤트 레코더를 제외한 모든 경우, 로그 블록의 수가 8개 미만으로 작아지면 데이타 블록들 간에 로그 블록을 차지하기 위한 경쟁이 심해져 전체적으로 성능이 상당히 나빠진다. 그렇지만 로그 블록 개수가 8개 이상인 경우 Postmark 벤치마크와 임베디드 시스템 워크로드 모두 안정된 성능을 보인다. 로그 블록의 개수가 많은

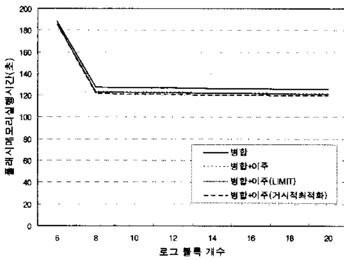


(a) 디렉토리/작은 파일 워크로드

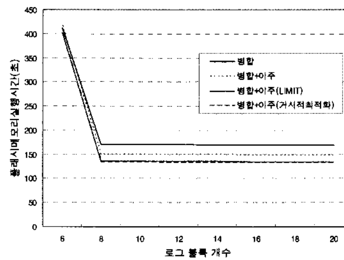


(b) 큰 파일 워크로드

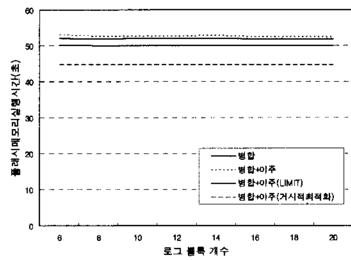
그림 11 Postmark 벤치마크에서 로그 블록 개수가 변할 때 성능 변화



(a) 팩스



(b) 휴대 전화



(c) 이벤트 레코더

그림 12 임베디드 시스템 워크로드에서 로그 블록 개수가 변할 때 성능 변화

경우 얻을 수 있는 이점은 데이타 블록이 로그 블록을 차지하기 위한 경쟁이 줄어들어, 경쟁으로 인하여 발생하는 블록 재활용 회수가 감소한다는 점이다. 즉, 로그 블록의 페이지를 모두 사용하지 않았음에도 해당 로그 블록을 다른 데이타 블록에게 할당하기 위하여 발생하는 병합 연산 회수를 줄일 수 있다. 그렇지만 실험에서 사용한 워크로드들은 소수의 메타 데이타용 로그블록과 파일 데이타용 로그 블록만을 필요로 하므로, 로그 블록의 개수가 8개 이상이면 경쟁으로 인한 병합은 거의 발생하지 않는다. 그렇지만 워크로드가 접근하는 섹터들이 다수의 블록에 퍼져 있는 경우 안정적인 성능을 유지하기 위한 로그 블록의 개수는 증가할 것이다.

7. 결 론

플래시 메모리를 저장장치로 사용하기 위해 필요한 소프트웨어 계층인 FTL은 데이타가 변경되면 이를 새로운 위치에 기록하고, 사상 정보를 변경하며, 병합 연산을 사용하여 블록을 재활용한다. 본 논문에서는 블록 재활용 비용을 줄이기 위하여, 이주 연산이라는 새로운 블록 재활용 기법을 제안하고, 이주 연산과 병합 연산의 비용 모델을 제시하였다. 그리고 블록을 재활용 할 때 두 연산 중 비용이 저렴한 연산을 선택적으로 적용하는 비용 기반 선택 기법을 제안하고 실험을 통하여 성능 향상 정도를 측정하였다. 또한 거시적 관점에서 가용 페이지 당 블록 재활용 비용을 최소화하는 이주/병합 순서가 존재함을 보이고, 이 비용을 최소화하는 거시적 최적화 해를 제시하였으며, 실험을 통하여 거시적 최적화가 블록 재활용 비용을 더욱 감소시킴을 보였다. 특히 워크로드의 변화에 적응하기 위해서는 이주와 병합 두 블록 재활용 기법이 적절한 순서로 혼합되어 적용되어야 함을 실험을 통해 증명하였다.

참 고 문 헌

[1] "Understanding the Flash Translation Layer (FTL) Specification," Intel Corporation, 1998.

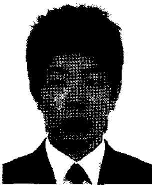
[2] M. Rosenblum and J. K. Ousterhout, "The Design and Implementation of a Log-Structured File System," *ACM Transactions on Computer Systems*, vol. 10, pp. 26-52, 1992.

[3] A. Kawaguchi, S. Nishioka, and H. Motoda, "A Flash-Memory Based File System," in *Proceedings of the Winter1995 USENIX Technical Conference*, 1995, pp. 155-164.

[4] "512M×8Bit/256M×16Bit NAND Flash Memory (K9K4G xxx0M) Data Sheets," Samsung Electronics, Co., 2003.

[5] "1G×8Bit/2G×8Bit NAND Flash memory (K9L8G

08U0M) Data Sheets," Samsung Electronics, Co., 2005.
[6] M.-L. Chiang, P. C. H. Lee, and R. C. Chang, "Managing Flash Memory in Personal Communication Devices," in *Proceedings of the 1997 International Symposium on Consumer Electronics (ISCE'97)*, 1997, pp. 177-182.
[7] "CF+ and CompactFlash Specification Revision 3.0," CompactFlash Association, 2004.
[8] "The MultiMediaCard System Summary," MMCA Technical Committee, 2005.
[9] "Flash-Memory translation layer for NAND flash (NFTL)," M-Systems.
[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 Transactions on Embedded Computing Systems*, vol. 6, Feb. 2007.
[11] J. Kim, J. M. Kim, S. H. Noh, S. L. Min, and Y. Cho, "A Space-efficient Flash Translation Layer for CompactFlash Systems," *IEEE Transactions on Consumer Electronics*, vol. 28, pp. 366-375, 2002.
[12] E. Gal and S. Toledo, "A Transactional Flash File System for Microcontrollers," in *USENIX Annual Technical Conference*, 2005, pp. 89-104.
[13] 김성관, 이동희, 노삼혁, 민상렬, "플래시 메모리를 위한 신뢰성 높은 소프트웨어 개발 환경," *SK Telecom Telecommunication Review*, Vol. 15, pp. 638-646, 2005.



이 종 민
2007년 서울시립대학교 컴퓨터과학부(학사). 2007년~현재 서울시립대학교 컴퓨터통계학과 석사과정. 관심분야는 운영체제, 플래시메모리소프트웨어

김 성 훈
2006년 서울시립대학교 수학과 졸업(학사). 2006년~현재 고려대학교 정보경영공학전문대학원 석사과정. 관심분야는 VANETs(Vehicular Ad-hoc Networks), IPTV(Internet Protocol Television) 보안

안 성 준
1997년 서울대학교 컴퓨터공학과(학사) 1999년 서울대학교 전기・컴퓨터공학부(석사). 2006년 서울대학교 전기・컴퓨터공학부(박사). 2006년~현재 삼성전자 소프트웨어연구소 책임연구원. 관심분야는 운영체제, 내장형시스템, 저장장치

이 동 희
1989년 서울대학교 컴퓨터공학과(학사) 1991년 서울대학교 컴퓨터공학과(석사) 1998년 서울대학교 컴퓨터공학과(박사) 1998년~1999년 삼성전자 중앙연구소 선임연구원. 1999년~2001년 제주대학교 통신컴퓨터과학부 조교수. 2002년~현재 서울시립대학교 컴퓨터과학부 부교수. 관심분야는 운영체제, 내장형시스템, 플래시메모리스토리지

노 삼 혁
1986년 서울대학교 컴퓨터공학과(학사) 1993년 메릴랜드대학교 컴퓨터과학과(박사). 1993년~1994년 조지워싱턴대학교 객원 조교수. 1994년~현재 홍익대학교 정보컴퓨터공학부 교수. 관심분야는 운영체제, 플래시메모리소프트웨어, 내장형시스템, 차세대저장장치