NAND 플래시 메모리에서 업데이트 패턴을 고려한 효율적인 페이지 할당 기법

(Efficient Page Allocation Method Considering Update Pattern in NAND Flash Memory)

김 희 태 † 한 동 윤 † 김 경 석 ††
(Hui-tae Kim) (Dongyun Han) (Kyongsok Kim)

요 약 플래시 메모리는 하드 디스크와 여러 면에서 다른데 특히 덮어쓰기가 되지 않는다는 것이 가장 큰 차이점이다. 그로 인해 대부분의 플래시 메모리 파일 시스템들은 파일을 수정할 때 not-in-place 수정 기법을 사용하고 있다. 그 과정에서 가끔 플래시 메모리 파일 시스템들은 가용 공간의 확보를 위해 무효 페이지들이 많은 블록들의 유효 페이지들을 다른 블록으로 옮기고 블록들을 쓸 수 있는 빈 페이지로 만들어 주는 작업인 블록 클리닝 작업을 수행한다. 블록 클리닝 작업은 플래시 메모리의 성능을 직접적으로 좌우하는 요소이다. 그래서 이 논문은 유효 페이지와 무효 페이지를 동시에 가진 블록의 수를 최소화하여 블록 클리닝 비용을 줄일 수 있는 효율적인 페이지 할당 기법을 제안한다. 그리고 실험 결과는 원래의 YAFFS에 비해 블록 클리닝 비용이 확연하게 줄어들었음을 보여 준다.

키워드 : 플래시 메모리, 파일 시스템, 페이지 할당 기법

Abstract Flash Memory differs from the hard disk, because it cannot be overwritten. Most of the flash memory file systems use not-in-place update mechanisms for the update. Flash memory file systems execute sometimes block cleaning process in order to make writable space while performing not-in-place update process. Block cleaning process collects the invalid pages and convert them into the free pages. Block cleaning process is a factor that affects directly on the performance of the flash memory. Thus this paper suggests the efficient page allocation method, which reduces block cleaning cost by minimizing the numbers of block that has valid and invalid pages at a time. The result of the simulation shows an increase in efficiency by reducing more block cleaning costs than the original YAFFS.

Key words : Flash memory, File system, Page allocation Method

1. 서 론

플래시 메모리는 비휘발성 메모리 반도체로서 기존 저장 장치인 하드 디스크와 비교해 크기와 소비 전력이 작고 외부 충격에 강하며, 기계적인 구성 요소가 없어 데이터의 접근 속도가 빠른 장점이 있다. 따라서 휴대폰, MP3, 디지털 카메라와 같은 휴대용 단말기기 뿐만 아니라 메모리 카드나 USB 메모리의 형태로 제작되어 이동식 저장 장치의 대부분을 차지하고 있으며, SSD (Solid State Disk)로 제작되어 차세대 대용량 저장 장치로서 광범위하게 사용하고 있다[1].

플래시 메모리는 저장 방식에 따라 낸드(NAND) 형과 노어(NOR) 형으로 나뉜다. 노어형은 바이트 단위로 주소 지정이 가능하고, 집적도가 낮고 고가이며, 읽기 연산 속도는 빠르지만 쓰기 연산 속도와 소거 연산의 속도가 느리기 때문에 코드 저장용으로 주로 사용된다. 낸드형은 페이지 또는 블록 단위로 주소 지정이 가능하며, 노어형에 비해 가격이 저렴하고 쓰기 연산과 소거

· 이 논문은 부산대학교 자유과제 학술연구비(2년)에 의해 연구되었음

† 학생회원 : 부산대학교 컴퓨터공학과
htk@asadal.pusan.ac.kr
dyhan@asadal.pusan.ac.kr
†† 종신회원 : 부산대학교 컴퓨터공학과 교수
gimgs0@asadal.pnu.kr

논문접수 : 2010년 7월 23일
심사완료 : 2010년 9월 9일

연산의 속도가 빠르기 때문에 데이터 저장용으로 많이 사용한다[2].

플래시 메모리는 단위 용량의 가격이 하드 디스크보다 높기 때문에 디스크 드라이브로 쓰기 위해서는 가격이 내려갈 필요가 있다. 현재 플래시 메모리의 용량은 급격히 증가하는 반면 가격은 하락하고 있는 추세이다. 특히 하나의 저장 단위(cell)에 2bit의 정보를 저장할 수 있는 MLC(Multi-Level Cell) 낸드 플래시 메모리가 개발되어 대용량화와 가격 하락을 촉진하고 있다. 낸드 플래시 메모리 시장은 2008년까지 연평균 성장률이 16.6%로, 앞으로 더 크게 성장할 것으로 전망된다[3].

플래시 메모리는 기존의 저장매체와 비교하여 다음과 같은 특징이 있다. 첫째, 읽기(read), 쓰기(write), 소거(Erase) 연산의 수행 단위가 다르다. 읽기와 쓰기 연산은 페이지 단위로 수행하며, 소거 연산은 페이지보다 큰 단위인 블록 단위로 수행한다. 둘째, 표 1에서와 같이 읽기, 쓰기, 소거 연산의 수행 시간이 다르다. 읽기와 쓰기 연산에 비해 소거 연산의 수행 시간이 길다. 셋째, 플래시 메모리는 덮어쓰기(overwrite) 연산이 불가능하다. 플래시 메모리의 각 비트는 단방향으로 토글링(toggling)되기 때문에 기존의 다른 저장 매체와는 달리 데이터를 수정할 때 본래의 주소에 덮어 쓰기(in-place update)가 불가능 하다. 따라서 특정 주소의 데이터를 수정하기 위해서는 초기화된 저장 공간을 미리 확보하여야 한다. 넷째, 각 블록의 소거 횟수가 제한되어 있다. 따라서 플래시 메모리의 전체 공간을 균등하게 사용하지 않을 경우, 데이터를 쓸 수 있는 공간이 줄어드는 현상이 발생한다[2,4-7].

덮어쓰기 연산의 제한은 쓰기 연산의 처리를 복잡하게 한다. 대부분의 플래시 메모리 파일 시스템은 특정 페이지에 대한 수정 요청이 오면, 새로운 페이지를 할당받아 그 곳에 새 데이터를 쓰고, 기존의 페이지는 무효화(invalid)시키는 방법(not-in-place update)을 사용한다[8].

업데이트로 인해 생성된 무효 페이지를 수집하여 다시 쓰기 가능한 상태로 초기화 하는 작업을 블록 클리닝(block cleaning)이라 한다. 블록 클리닝은 먼저 소거할 희생 블록을 선택한 후, 그 블록 안에 유효한(valid) 데이터가 존재할 경우 이를 다른 공간에 복사하고, 희생 블록을 소거하는 순서로 진행된다. 블록 클리닝은 읽기 연산과 쓰기 연산에 비해 연산 시간이 길고, 전력

표 1 플래시 메모리 기본 연산 수행 시간

연산 / 디바이스	Read	Write	Erase
NAND Flash	35.9us(512B)	226us(512B)	2ms(16KB)
NOR Flash	150ns(1B) 14.4us(512B)	211us(1B) 3.53ms(512B)	1.2s(128KB)

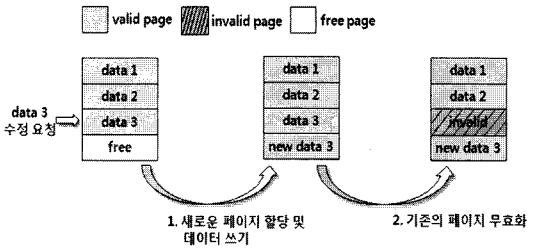

그림 1 낸드 플래시 메모리의 업데이트 (not-in-place update)

소모량도 많다[4-6]. 또한 희생 블록 내에 유효 페이지가 있을 경우 해당 페이지를 모두 다른 공간에 다시 써야 하는 추가적인 오버헤드가 발생한다. 따라서 블록 클리닝의 비용을 줄이는 것은 플래시 메모리 파일 시스템의 성능을 향상 시킬 수 있는 핵심 요소라 할 수 있다. 블록 클리닝 비용을 줄이기 위한 방법은 다음과 같다.

- 블록 클리닝 시 발생하는 유효 페이지 복사 횟수를 줄인다.
- 블록 소거 연산의 횟수를 줄인다.

본 논문에서는 페이지의 업데이트 패턴에 따라 Hot 페이지와 Cold 페이지로 나누어 서로 다른 공간에 할당함으로써 같은 블록에 유효(valid) 페이지와 무효(invalid) 페이지가 섞여 있는 것을 최소화하여 결과적으로 블록 클리닝 비용을 줄일 수 있는 효율적인 페이지 할당 기법을 제안한다. 이 때 Hot 페이지와 Cold 페이지를 판정하는 것이 중요한데 본 논문에서는 이벤트 윈도우(Event Window)를 사용하여 Hot 페이지와 Cold 페이지를 판정한다. 제안한 페이지 할당 기법은 YAFFS 를 기반으로 구현하였으며, 성능 평가 결과 기존의 YAFFS 에 비해 평균 30%의 블록 클리닝 비용을 감소시켰다.

본 논문은 다음과 같이 구성된다. 2장에서는 관련 연구를 소개한다. 3장에서는 본 논문에서 제안한 업데이트 패턴에 따른 페이지 할당 기법에 대하여 설명한다. 4장에서는 제안하는 기법의 구현과 성능 평가를 보인다. 마지막으로 5장에서는 결론과 향후 과제를 제시한다.

2. 관련 연구

본 장에서는 플래시 메모리의 특징에 대하여 알아보고, 본 논문에서 제안하는 페이지 할당 기법을 구현하는데 기반이 된 낸드 플래시 메모리 전용 파일 시스템인 YAFFS(Yet Another Flash File System)에 대하여 설명한다. 마지막으로 본 논문의 기본 아이디어인 업데이트 패턴에 따른 페이지 할당 기법을 적용한 기존의 할당 기법에 대하여 설명한다.

2.1 플래시 메모리의 구조

그림 2는 1Gbit 낸드 플래시 메모리의 구조를 보여준다. 낸드 플래시 메모리는 블록과 페이지로 구성된다. 그리고 블록과 페이지의 크기에 따라 대블록(large block)과 소블록(small block)으로 나뉜다. 1Gbit 대블록 낸드 플래시 메모리의 경우 전체 1,024개의 블록으로 이루어져 있고 하나의 블록은 64개의 페이지로 구성되며, 각 페이지는 2,112Bytes(2048Bytes for data + 64 Bytes for spare area)의 크기를 갖는다. 소블록 낸드 플래시 메모리는 전체 8,192개의 블록으로 이루어져 있고 하나의 블록은 32개의 페이지로 구성되며, 각 페이지는 528Bytes(512Bytes for data + 16Bytes for spare area)로 구성된다[7,9,10]. 본 본문에서는 소블록 낸드 플래시 메모리를 사용하여 제안하는 기법을 구현하고 성능 평가를 수행하였다.

플래시 메모리는 페이지 상태에 따라 유효(valid) 페이지, 무효(invalid) 페이지, 빈(free) 페이지, 3가지로 나뉜다. 빈 페이지에 데이터를 쓰게 되면 그 페이지는 유효한 상태로 변경 된다. 그리고 유효 페이지의 데이터를 업데이트하거나 삭제하면 그 페이지는 무효한 상태가 된다. 마지막으로 블록 클리닝에 의해 무효 페이지를 포함하는 블록이 소거되면 그 블록 내에 있는 모든 페이지는 데이터를 쓸 수 있는 빈 페이지가 된다[7,8,11].

낸드 플래시 메모리는 무효(invalid) 페이지를 수집하여 데이터를 쓸 수 있는 빈(free) 페이지로 만들기 위해 블록 클리닝을 수행한다. 플래시 메모리의 블록 클리닝에는 두 가지 종류가 있다.

ㄱ) 희생 블록에 유효 페이지와 무효 페이지가 함께 섞여 있는 경우

ㄴ) 희생 블록에 무효 페이지만 존재하는 경우

ㄱ)의 경우 블록 클리닝은 유효 페이지 복사(valid page copy)와 블록 소거(block erase) 연산으로 이루어진다. 하나의 블록을 소거하는 비용은 플래시 메모리의 전체 블록에 대하여 동일하다. 따라서 이 경우 유효 페이지의 개수가 블록 클리닝 비용에 영향을 준다. 반면 ㄴ)의 경우 한 번의 블록 소거 연산만을 필요로 한다[4,5,8].

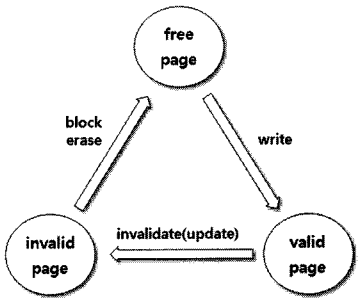

그림 3 낸드 플래시 메모리의 페이지 상태와 전이

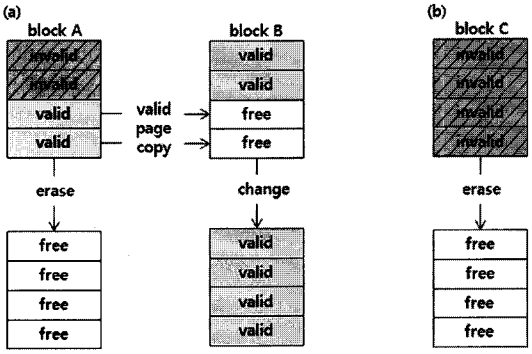

그림 4 플래시 메모리의 블록 클리닝

2.2 YAFFS(Yet Another Flash File System)

YAFFS는 2002년 5월 Aleph One 사에서 개발한 낸드(NAND) 플래시 메모리 전용 파일 시스템으로 JFFS

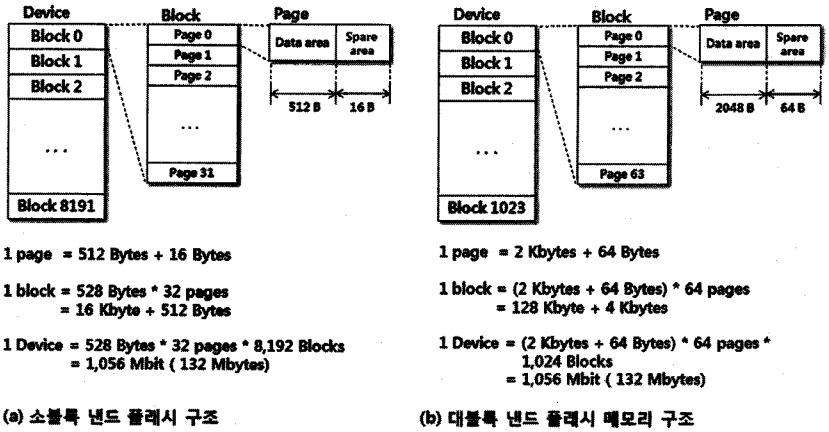

그림 2 1 Gbit 낸드 플래시 메모리의 구조

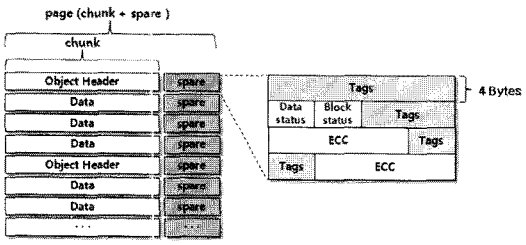

그림 5 플래시 메모리 상의 데이터 구조

(Journaling Flash File System)의 높은 램 사용률, 느린 마운팅 속도 문제 등을 보완하였다[12]. YAFFS는 입출력 연산의 기본 단위인 페이지를 청크(chunk)와 스페어(spare)라는 용어를 사용하여 관리한다[13]. 그림 5는 YAFFS의 청크와 스페어 영역을 보여준다. 청크는 파일(디렉터리, 정규 파일, 하드링크, 소켓 등)의 메타 정보를 담은 "헤더 청크"와 정규 파일의 데이터를 담은 "데이터 청크"로 나뉜다. 헤더 청크는 파일의 이름, 크기, 수정 시간, 상위 오브젝트의 아이디(id) 등으로 구성된다[12,13].

스페어 영역에는 에러 검출을 위한 ECC(Error Correcting Code) 필드와 하나의 청크에 대한 메타 정보를 저장하는 태그(tag) 영역이 있다. 태그 영역에는 해당 청크가 속한 파일의 아이디(id), 파일 내부에서 청크의 인덱스, 오류를 복구할 때 사용하는 시리얼 번호(serial number) 등을 저장한다. 낸드 플래시 메모리가 마운트되면 YAFFS는 낸드 플래시 메모리의 헤더 청크와 데이터 청크의 스페어 영역을 읽어들여 메인 메모리에 파일 시스템을 구성한다. 각 헤더 청크마다 yaffs_Object라는 객체를 생성하고, 이들 객체는 서로 연결되어 트리 구조를 형성한다. 그리고 yaffs_Object가 정규 파일 객체일 경우, 파일의 실제 데이터를 저장하고 있는 데이터 청크의 위치를 Tnode라는 트리로 관리한다.

정규 파일 객체에 Tnode를 구성하기 위해서는 해당 객체가 포함하는 모든 데이터 청크의 스페어 영역을 읽어야 한다[12,13]. 정규 파일 객체에는 top이라는 필드가 존재하는데 이것은 Tnode의 최상위 노드를 가리킨다. Tnode 는 크게 내부 노드와 레벨 0 노드로 구분하는데, 내부 노드에는 다른 내부 노드나 레벨 0 노드의 위치를 저장하고, 레벨 0 노드는 데이터 청크의 실제 물리 주소(낸드 플래시 메모리 상의 실제 물리 페이지 번호)를 저장한다.

YAFFS는 페이지 할당 기법으로 순차 할당 방식을 사용한다. 낸드 플래시 메모리의 전체 블록 중 빈(free) 페이지를 포함하는 블록을 찾고, 그 블록 내에서 순차적으로 페이지를 할당한다.

YAFFS의 블록 클리닝은 쓰기 연산을 수행하기 직전에 발생하며, 블록 이용률(utiliaztion)이 가장 낮은 하나의 블록을 선택하여 소거한다. 이용률이 낮은 블록을 선택하여 소거하는 기법을 그리디(Greedy)라고 한다. 그리디 기법은 블록 소거 연산 중 발생하는 유효(valid) 페이지의 복사 횟수를 최소화하여 소거 효율은 높으나, 특정 블록에 소거 연산이 집중될 수 있어 메모리의 균등 사용이 어렵다는 단점을 가지고 있다[13].

2.3 기존 할당 기법

참고문헌[5]에서는 새로 쓰이는 페이지와 블록 클리닝에 의해 복사되는 유효 페이지를 Not-Cold 블록과 Cold

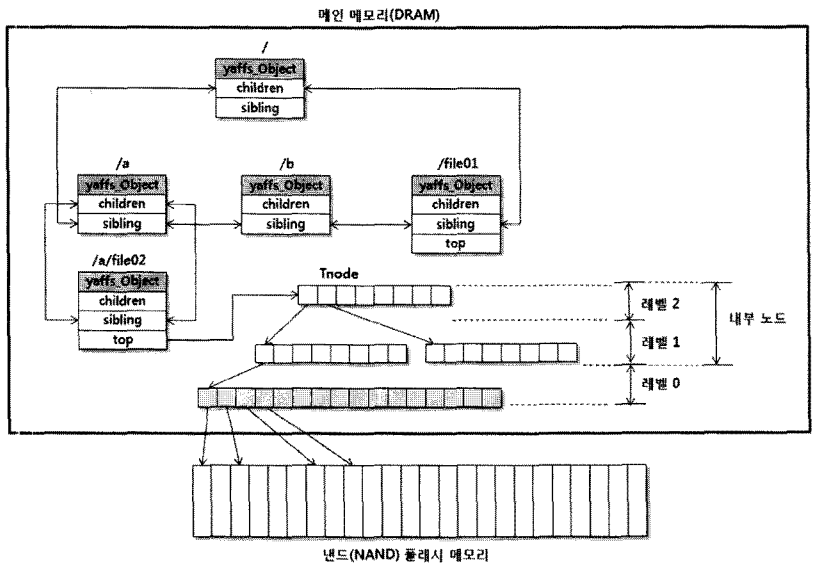

그림 6 YAFFS의 메인 메모리상의 자료구조

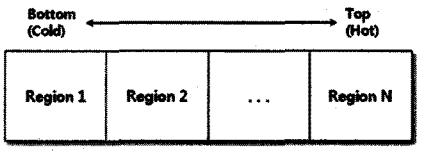

그림 7 DAC 기법의 논리적인 구조

블록에 나누어 쓰는 기법(separate segment for cleaning)을 제안하였다.

참고문헌[6]에서는 플래시 메모리를 여러 개의 구역으로 나누어 페이지가 업데이트 되면 상위 구역으로 이동시키고, 업데이트가 없는 페이지는 하위 구역으로 이동시키는 DAC(Dynamic dAta Clustering) 기법을 제안하였다. 그림 7과 같이 업데이트 빈도가 높은 페이지는 Top 지역에 위치시키고, 업데이트 빈도가 낮은 페이지는 Bottom 지역에 위치시킴으로써 Hot 페이지와 Cold 페이지를 분류하였다.

참고문헌[14]에서는 블록 클리닝을 할 때 발생하는 누적 유효 페이지 복사 횟수를 기준으로 임계치 이상의 복사 횟수를 보이는 페이지를 Cold 페이지로 판정하고 이들을 특정 블록에 할당하였다. 그리고 Cold 페이지를 모아둔 블록이 희생 블록으로 선택되는 것을 방지하여 전체 블록 소거 횟수와 유효 페이지의 복사 횟수를 줄였다.

참고문헌[8]에서는 블록 클리닝의 성능에 영향을 주는 인자로 순수도(Uniformity)를 제안하였다. 순수도는 플래시 메모리에서 무효 상태의 페이지 분포를 나타낸다. 블록이 유효 페이지나 무효 페이지로만 구성하였을 때 높은 순수도를 가지며, 순수도가 높을수록 블록 클리닝의 비용을 줄일 수 있다고 정의하였다. 그리고 높은 순수도를 유지할 수 있는 페이지 할당 기법으로 MODA (modification-aware)를 제안하였다.

3. 제안하는 페이지 할당 기법

본 장에서는 누적 블록 클리닝 비용에 대하여 알아보고, 누적 블록 클리닝 비용을 줄이기 위해 본 논문에서 제안한 페이지 할당 기법에 대하여 설명하도록 하겠다.

3.1 블록 클리닝 비용

본 논문에서는 제안하는 페이지 할당 기법의 성능을 평가하기 위해 누적 블록 클리닝 비용을 사용한다. 누적 블록 클리닝 비용이란 특정 이벤트가 발생할 동안 누적된 블록 클리닝 비용을 의미한다. 누적 블록 클리닝 비용을 수식으로 나타내면 식 (1)과 같다.

$$CC_F = (NV \times MC_V) + (NE \times EC) \quad (1)$$

CC_F : $Cleaning\ Cost_{Flash\ memory}$

NV : $Number\ of\ Validpage$

MC_V : $Migrate\ Cost_{Validpage}$

NE : $Number\ of\ Erase$

EC : $Erase\ Cost$

식 (1)의 CC_F는 플래시 메모리의 누적 블록 클리닝 비용을 나타낸다. 식은 두 개의 항으로 구성되는데 첫 번째 항은 블록 클리닝 중 발생하는 누적 유효(valid) 페이지 복사 비용을 나타내고, 두 번째 항은 누적 블록 소거(erase) 연산 비용을 나타낸다. 첫 번째 항의 MC_V는 하나의 유효(valid) 페이지를 다른 공간으로 복사하는데 드는 비용이며, 읽기 연산과 쓰기 연산으로 이루어진다. 플래시 메모리는 기존의 하드 디스크와는 달리 기계적인 탐색연산이 없다. 따라서 MC_V를 연산 수행 시간으로 식 (2)와 같이 나타낼 수 있다.

$$MC_V = t\,Read + t\,Write \quad (2)$$

MC_V는 플래시 메모리의 모든 페이지에 대하여 동일하다. NV는 블록 클리닝 중 발생한 유효 페이지의 개수이다.

두 번째 항의 EC는 단순히 하나의 블록을 소거하는 비용이고, 플래시 메모리의 모든 블록에 대하여 동일하다. EC는 식 (3)과 같이 블록 소거 연산의 수행 시간으로 나타낸다.

$$EC = t\,Erase \quad (3)$$

NE는 블록 소거 연산의 횟수이다. 결론적으로 누적 블록 클리닝 비용은 누적 유효 페이지 복사 비용과 누적 블록 소거 연산 비용을 합한 것이다. 누적 블록 클리닝의 비용을 줄이기 위한 방법은 다음과 같다.

- 블록 클리닝 중 발생하는 유효 페이지의 개수를 줄인다.
- 블록 소거 횟수를 줄인다.

하나의 유효 페이지를 다른 공간에 복사하는 비용인 MC_V와 블록 하나를 소거하는 비용인 EC는 그 값이 플래시 메모리에서 일정하다. 따라서 유효 페이지의 개수인 NV를 줄임으로서 블록 클리닝 중 발생하는 누적 유효 페이지 복사 비용을 줄일 수 있으며, 블록의 소거 횟수인 NE를 줄임으로서 누적 블록 소거 연산 비용을 줄일 수 있다.

3.2 업데이트 패턴에 따른 데이터 분류 및 할당

희생 블록의 유효 페이지 개수를 줄이고, 블록 소거 횟수를 줄이기 위해 본 논문에서는 업데이트 패턴에 따른 페이지 할당 기법을 사용한다. 즉 업데이트가 빈번한 페이지(Hot 페이지)와 그렇지 않은 페이지(Cold 페이지)를 구분하여 서로 다른 공간에 할당하는 것이다.

페이지의 상태 변이는 업데이트 패턴과 관련이 있다. 자주 업데이트되는 페이지는 빠르게 무효화가 진행되고, 그렇지 않은 페이지는 상대적으로 무효화가 느리게 진행된다. 따라서 업데이트 패턴에 따라 Hot 페이지와 Cold 페이지로 분류하고, 이들을 서로 다른 블록에 할당하면 같은 블록에 유효 페이지와 무효 페이지가 섞여

있는 것을 최소화 할 수 있다.

그림 8은 기존의 순차 할당 기법과 본 논문에서 기본 아이디어로 사용하고 있는 업데이트 패턴에 따른 페이지 할당 기법의 차이를 보여준다. 그림 8에서 하나의 블록은 4개의 페이지로 구성되고, 낸드 플래시 메모리의 블록 개수는 5개로 가정한다. 파일 A(3개의 페이지), 파일 B(2개의 페이지), 파일 C(2개의 페이지), 파일 D(1개의 페이지), 파일 E(1개의 페이지)를 생성한 후, 파일 B 업데이트, 파일 D 업데이트 순으로 이벤트가 발생하였다고 가정한다. 이때 파일 B와 파일 D는 업데이트가 빈번한 파일임을 알고 있다고 가정한다.

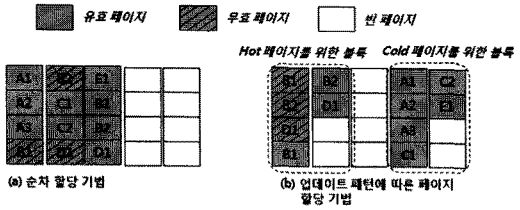

그림 8 페이지 할당 기법

그림 8의 (a)는 기존의 순차 할당 기법을 나타낸 것으로 생성한 파일 5개와 업데이트된 파일 B, 파일 D의 페이지를 순차적으로 할당한다. 그림 8의 (b)는 업데이트 패턴에 따른 페이지 할당 기법을 나타낸 것으로 업데이트가 빈번한 파일 B, 파일 D는 Hot 페이지를 위한 블록에 할당하고, 나머지 블록은 Cold 페이지를 위한 블록에 할당한다. 그리고 파일 B, 파일 D의 업데이트는 Hot 페이지를 위한 블록 내에서 모두 처리된다. 파일 B와 파일 D의 업데이트 후, 무효화(invalid) 상태의 페이지를 모두 빈(free) 상태의 페이지로 변경한다고 할 때, 기존의 순차 할당 기법은 2번의 블록 소거 연산과 5번의 유효 페이지 복사가 필요하다. 반면, 업데이트 패턴에 따른 페이지 할당 기법은 1번의 블록 소거 연산과 1번의 유효 페이지 복사만을 필요로 한다.

그림 8은 업데이트 패턴을 고려하여 Hot 페이지와 Cold 페이지를 서로 다른 블록에 할당하여 유효 페이지와 무효 페이지가 같은 블록에 섞여 있는 것을 최소화하고, 이를 통해 기존의 순차 할당 기법과 비교하여 블록 클리닝 시 유효 페이지 복사 횟수와 블록 소거 연산의 횟수가 감소하는 것을 보여준다. 자주 업데이트 되는 Hot 페이지를 한 블록에 모아두면 데이터의 공간 지역성을 높임으로써 희생 블록의 페이지가 대부분 무효화되어 블록 클리닝 시 발생하는 유효 페이지의 복사 횟수를 줄일 수 있다. 또한 자주 업데이트 되지 않는 Cold 페이지를 한 블록에 모아두면 블록의 이용률(utilization)을 높임으로써 해당 블록이 희생 블록으로 선택되는 것을 지연시켜 블록 클리닝 횟수를 감소시킬 수 있다. 또한 Cold 페이지가 플래시 메모리 전 영역에 분산되는 것을 줄임으로써 Cold 페이지가 불필요하게 여러 블록에 옮겨 다니는 것을 막을 수 있다.

3.3 이벤트 윈도우를 사용한 Hot 페이지와 Cold 페이지 분류

업데이트 패턴에 따라 페이지를 할당하기 위해 Hot 페이지와 Cold 페이지를 판정하는 것이 중요하다. 하지만 Hot 페이지와 Cold 페이지를 판정하기 위해 지나치게 많은 부하가 발생한다면 오히려 전체 성능을 저하시킬 수 있다. 그래서 본 논문에서는 Hot 페이지와 Cold 페이지를 판정하기 위해 단순하며 빠르게 판정할 수 있는 "이벤트 윈도우(Event Window)"라는 자료구조를 사용하였다.

그림 9는 본 논문에서 제안하는 페이지 할당 기법의 전체 구조이다. 그림 9의 (a)는 각 페이지의 업데이트 횟수를 기준으로 Hot 페이지와 Cold 페이지를 분류하는 이벤트 윈도우이다. 이벤트 윈도우에서 분류된 페이지는 그림 9의 (b)에 있는 Block Manager에 의해 Hot 블록, Cold 블록 그리고 Normal 블록에 할당된다.

이벤트 윈도우는 최근에 이벤트가 발생한 페이지의 정보를 저장하는 큐(queue) 형태를 가진다. 이벤트 윈도우의 단위(unit) 크기는 각 이벤트에 대응한다. 이벤트 윈도우 안에 있는 페이지는 최근에 이벤트가 발생하였다고 가정한다. 따라서 이벤트 윈도우에 포함되지 않은 페이지는 Cold 페이지로 판정한다. Hot 페이지는 이벤트 윈도우 내에 포함된 페이지들의 누적 업데이트 횟수를 기준으로 판정한다. 이는 업데이트되는 페이지가 지역성을 가진다면 윈도우 크기만큼의 이벤트 발생동안 같은 페이지가 반복적으로 업데이트 되는 경향이 있다는 것을 의미한다. 이를 위해 "Hot 페이지 임계값"을

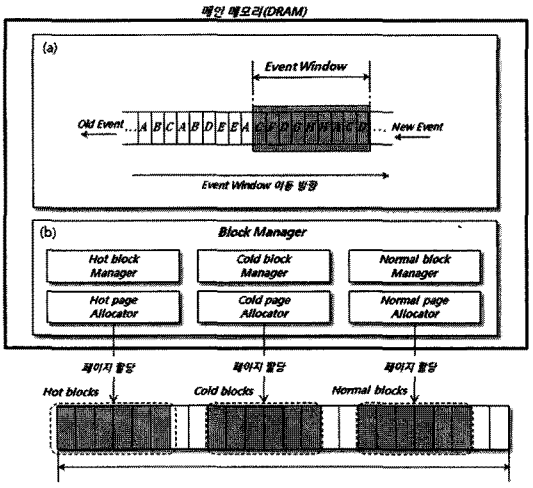

그림 9 제안하는 페이지 할당 기법의 전체 구조

사용한다. 이벤트 윈도우 안에서 각 페이지의 누적 업데이트 횟수를 기준으로 Hot 페이지 임계값 이상인 페이지는 Hot 페이지로 판정하고, 그렇지 않은 페이지는 Normal 페이지로 판정한다.

페이지에 이벤트가 발생하면 해당 페이지의 정보(페이지 번호)를 윈도우에 삽입하고, 삽입된 페이지의 누적 업데이트 횟수는 1 증가한다. 그리고 이벤트 윈도우가 꽉 찬 상태에서 새로운 이벤트가 발생하면 이벤트 윈도우에서 가장 오래된 이벤트를 삭제한다. 윈도우에서 삭제된 이벤트의 페이지는 누적 업데이트 횟수가 1 감소한다.

그림 10은 이벤트 윈도우를 사용하여 Hot 페이지와 Cold 페이지를 분류하는 과정을 보여준다. 이 알고리즘은 LRU-K 페이지 교체 알고리즘을 사용한 것이다[15]. 이벤트 윈도우의 크기는 10으로 가정하고, Hot 페이지의 임계값은 2로 가정한다. 그림 10의 이벤트 윈도우 내에 있는 알파벳은 페이지를 의미하며, 알파벳 뒤의 숫자는 페이지의 누적 업데이트 횟수를 의미한다. 그림 10의 (a)에서 이벤트는 A, B, F, B, C, D, B, G, C, C 순으로 발생하였다. 그림 10의 (a)에서 각 페이지의 누적 업데이트 횟수는 A:1, B:3, C:3, D:1, F:1, G:1이다. 따라서 페이지 B, C는 Hot 페이지로 판정하고 나머지 페이지는 Normal 페이지로 판정한다. 그림 10의 (b)에서는 페이지 D에 대한 업데이트가 발생하여 가장 오래된 이벤트인 페이지 A를 삭제하고, 페이지 D에 대한 새로운 이벤트가 윈도우에 삽입된다. 그림 8의 (b)에서는 페이지의 누적 업데이트 횟수가 A:0, B:3, C:3, D:2, F:1, G:1이 되고, 누적 업데이트 횟수가 2 이상인 페이지 B, C는 Hot 페이지가 된다. 이벤트 윈도우에서 삭제된 페이지 A는 Cold 페이지로 판정한다. 어떤 페이지가 윈도우에 삽입되고 한 번도 업데이트되지 않은 상태로 윈도우에서 삭제되면 해당 페이지의 누적 업데이트 횟수는 0이 된다. 결국, 이벤트 윈도우에 없는 페이지는 누적 업데이트 횟수가 0이 되는 것을 의미한다.

제안하는 페이지 할당 기법은 Hot 페이지와 Cold 데이터를 판정하기 위해 이벤트 윈도우의 크기와 Hot 페이지 임계값을 결정하는 것이 중요하다. 이 두 가지 값은 Postmark 벤치마크를 수행하여 구하였으며, 이는 4장 성능 평가 절에서 자세히 설명하도록 하겠다.

페이지를 분류하고 할당하는 시점은 페이지의 업데이트 패턴에 따라 다르다. 그림 11에서 보는 바와 같이 Hot 페이지의 판정 및 할당은 새로운 페이지에 대한 쓰기 연산이 발생 하였을 때, 또는 블록 클리닝 중 유효 페이지 복사가 발생하였을 때 시도한다. 반면 Cold 페이지의 판정 및 할당은 블록 클리닝 중 유효 페이지 복사가 일어날 때만 시도한다. 그리고 페이지가 최초로 할당될 때, 또는 Hot 페이지와 Cold 페이지의 조건을 만족하지 않을 경우에는 Normal 블록에 할당한다.

4. 성능 평가

본 장에서는 성능 평가에 대하여 설명하겠다. 본 논문에서는 Postmark 벤치마크와 지역성의 변화에 따른 블록 클리닝 비용을 측정하여 제안한 페이지 할당 기법을 추가한 수정된 YAFFS(Modified YAFFS)와 기존의 YAFFS (Origianl YAFFS)에 대한 비교 분석을 수행하였다.

4.1 실험 환경 및 방법

본 논문에서 제안하는 페이지 할당 기법은 표 2의 임베디드 보드에 구현하였으며, 삼성 K9F1208U0B 소블록 낸드 플래시 메모리를 사용하였다. 임베디드 보드에는 리눅스 커널이 동작하며, 커널 버전은 2.4.19를 사용하였다. 그리고 제안하는 페이지 할당 기법은 낸드 플래시 메모리 전용 파일 시스템인 YAFFS(Yet Another Flash File System)를 기반으로 구현하였으며, 이를 위해

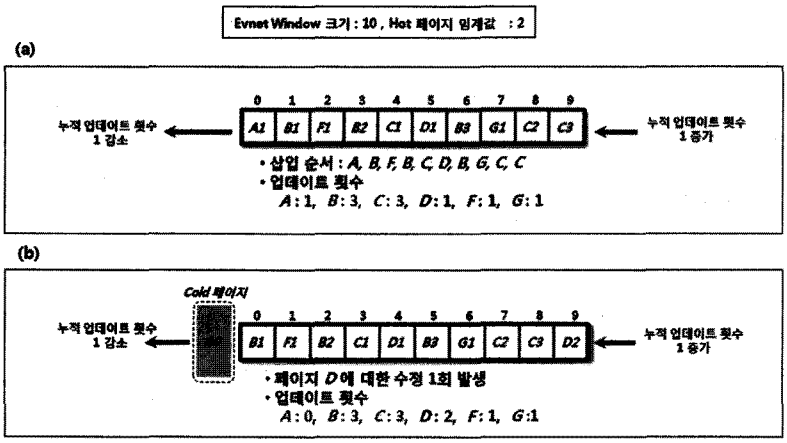

그림 10 이벤트 윈도우를 사용한 페이지의 분류

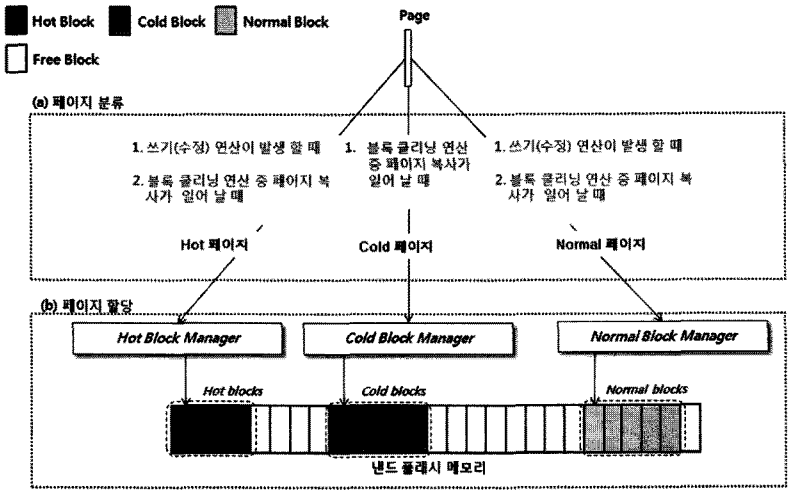

그림 11 Hot 페이지와 Cold 페이지를 분류하고 할당하는 시점

표 2 테스트 환경

테스트 수행 환경	
임베디드 보드	FALINUX EZ-X5
커널 버전	2.4.19
하드웨어	사양
CPU	PXA255 ARM RISC Chip 400MHz
RAM	32 MBytes SDRAM * 2
NAND Flash memory	K9F1208U0M (64MBytes)

yaffs_guts.c 파일에 있는 yaffs_WriteDataToFile(), yaffs_AllocateChunk(), yaffs_WriteChunkDataToObject(), yaffs_WriteNewChunkWithTagsToNAND() 등의 기존 함수를 수정하였다. 그리고 이벤트 윈도우(Event Window), 페이지 할당과 블록 관리를 위해 새로운 함수를 추가하였다.

4.2 누적 블록 클리닝 비용의 측정

YAFFS는 현재 가장 많이 사용하고 있는 낸드(NAND) 플래시 메모리 전용 파일 시스템으로 낸드 플래시 메모리의 제한 사항과 특징을 고려하여 최대한 성능을 내게끔 설계된 파일 시스템이다[16]. 따라서 본 논문에서는 Postmark 벤치마크와 지역성에 따른 블록 클리닝 횟수, 유효 페이지 복사 횟수, 누적 블록 클리닝 비용(CC_F)을 구하여 기존의 YAFFS(Original YAFFS)와 수정된 YAFFS(Modified YAFFS)를 비교 평가하였다. 수정된 YAFFS는 제안한 페이지 할당 기법을 추가한 것이다.

YAFFS의 블록 클리닝은 그리디(Greedy) 기법을 사용하며, 매 쓰기 연산 직전에 주기적으로 발생한다. 기존 YAFFS와 수정된 YAFFS는 페이지 할당 정책만 다를 뿐 다른 조건은 모두 동일한 상태에서 실험을 진행하였다.

식 (1)의 CC_F는 누적 블록 클리닝 비용을 나타낸다. 유효 페이지 하나를 다른 공간에 복사하는 비용인 식 (1)의 MC_V는 낸드 플래시 메모리의 모든 페이지에 대하여 동일하고, 그 값은 식 (2)에서 보듯이 읽기(read) 연산 수행 시간과 쓰기(write) 연산 수행 시간의 합이다. 표 1을 참고하면 읽기 연산 시간은 35.9us (512B)이고, 쓰기 연산 시간은 226us (512B)라는 것을 알 수 있다. 따라서 MC_V 의 값은 261.9us이다.

하나의 블록을 소거하는 비용인 식 (1)의 EC는 플래시 메모리의 모든 블록에 대하여 동일하고 연산 시간은 2ms이다. 결론적으로 EC가 MC_V 보다 약 7.3 배의 비용이 더 든다. 따라서 식 (1)은 EC와 MC_V를 변경하여 식 (4)와 같이 나타낼 수 있다.

$$CC_F = (NV \times 1) + (NE \times 7.3) \quad (4)$$

식 (4)는 성능 평가에서 누적 블록 클리닝 비용(CC_F)을 계산하는데 사용한다.

4.2.1 DAC 기법과의 성능 비교

이 절에서는 먼저 Postmark 벤치마크를 이용해서 제안한 방법을 적용하여 수정한 YAFFS의 성능이 얼마나 개선되었는지 평가하였다. 객관적인 평가를 위해서 기존의 YAFFS(Original YAFFS)와 참고문헌 [6]에 나오는 DAC 기법을 적용한 YAFFS(DAC YAFFS), 그리고 본 논문에서 제안한 이벤트 윈도우를 적용한 YAFFS (Modified YAFFS) 각각에 대하여 이용률(utilization)의 변화에 따른 블록 클리닝 횟수 및 유효 페이지 복사 횟수를 측정하였다. DAC의 실험 조건은 greedy한 상태에서 영역을 세 개로 구분하였는데 그것은 본 논문에서 제안한 기법 역시 영역을 세 개로 구분하기 때문에 공평한 조건으로 평가를 하기 위함이다.

그리고 일반적인 워크로드 측정을 위해 Postmark 벤치마크를 수행하였다. Postmark는 주어진 범위에서 랜덤 크기로 파일들을 생성한다. 그리고 이 파일들에 대하여 생성(create)하거나 삭제(delete)하는, 파일을 읽거나(read) 쓰는(append) 식으로 쌍을 이루어 트랜잭션을 수행한다. 파일의 개수, 수행할 트랜잭션 크기는 사용자가 설정할 수 있다[7,17,18].

Postmark의 파라미터로는 512Bytes에서 4KBytes까지의 크기로 작은 파일들을 생성하였고, 트랜잭션은 10,000회, 읽기 연산과 쓰기 연산의 단위는 512Bytes로 설정하였다. 읽기 연산과 쓰기 연산의 수행 비율은 1:1 (Postmark의 기본설정, set bias read 5)로 설정하였다.

이용률은 블록 클리닝 비용에 큰 영향을 주는 성능 인자로, 플래시 메모리에서 유효 페이지가 차지하는 비율을 의미한다. 이용률이 클수록 블록 클리닝 비용은 증가한다[7,8,11]. 이용률은 20%, 40%, 60%로 증가시키면서 실험을 수행하였다.

그림 12는 Postmark 벤치마크를 수행하였을 때, 플래시 메모리의 이용률에 따른 블록 클리닝 횟수를 보여준다. 이용률이 20%일 때, 수정된 YAFFS는 기존의 YAFFS 보다 약 12%의 블록 클리닝 횟수 감소를 보였으며 DAC를 적용한 YAFFS 보다는 약 10%의 블록 클리닝 횟수 감소를 보였다. 그리고 이용률이 60%일 때는 기존의 YAFFS에 비해서는 35.7%, 그리고 DAC를 적용한 YAFFS이 비해서는 약 29%의 블록 클리닝 횟수 감소를 보였다. 즉, 이용률이 증가할수록 기존의 YAFFS와 수정된 YAFFS의 성능 차이가 커지고 DAC를 적용한 YAFFS에 비해서도 더 좋은 성능을 보임을 알 수 있다. 이는 제안한 페이지 할당 기법이 이용률이 증가하는 상황에서도 페이지를 효율적으로 재편하여 블록 클리닝의 횟수를 감소시키는데 효과적임을 알 수 있다.

그림 13은 Postmark 벤치마크를 수행하였을 때, 이용률에 따른 유효 페이지 복사 횟수를 보여준다. 유효 페이지의 복사 횟수는 이용률이 20%일 때, 수정된 YAFFS가 기존의 YAFFS 보다 57%, DAC를 적용한 YAFFS

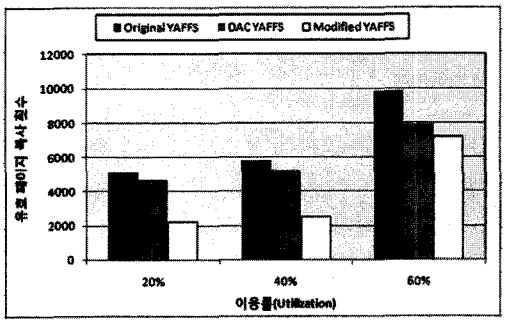

그림 13 이용률에 따른 유효 페이지 복사 횟수

보다 53% 감소하였고, 이용률이 60%일 때는 기존의 YAFFS에 비해서는 27%, DAC를 적용한 YAFFS에 비해서는 8.9% 감소하였다.

즉, 이용률이 증가하면서 기존의 YAFFS와 DAC를 적용한 YAFFS, 수정된 YAFFS 모두 유효 페이지 복사 횟수가 증가하였다. 이때 이용률이 증가하면서 기존의 YAFFS와 DAC를 적용한 YAFFS, 그리고 수정된 YAFFS와의 성능 차이가 각각 1/2, 1/6로 줄어들었고, 이는 제안한 페이지 할당 기법이 이용률이 증가할 때 유효 페이지의 복사 횟수 증가를 효과적으로 방지하지 못하는 단점이 있음을 알 수 있다.

그림 14는 이용률에 따른 누적 블록 클리닝 비용(CC_F)을 나타낸다. 이용률이 20%일 때, 수정된 YAFFS의 누적 블록 클리닝 비용은 기존의 YAFFS 보다 약 23% 감소하였고, DAC를 적용한 YAFFS에 비해서는 약 19.5% 정도 감소하였다. 이용률이 60%일 때는 기존의 YAFFS에 비해서는 약 34%가 감소하였고 DAC를 적용한 YAFFS에 비해서는 약 26.7% 감소하였다. 즉, 이용률이 증가하면서 수정된 YAFFS와 기존의 YAFFS, DAC를 적용한 YAFFS와의 성능차가 커짐을 알 수 있고, 이는 제안한 페이지 할당 기법이 이용률이 증가할수록 블록 클리닝 비용을 감소시키는데 효과적임을 알 수 있다. 그리고 본 논문에서 제안한 할당 기법은 DAC와

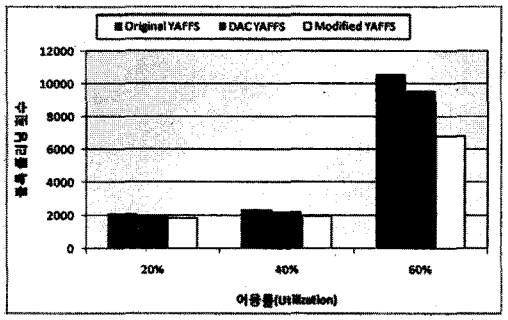

그림 12 이용률에 따른 블록 클리닝 횟수

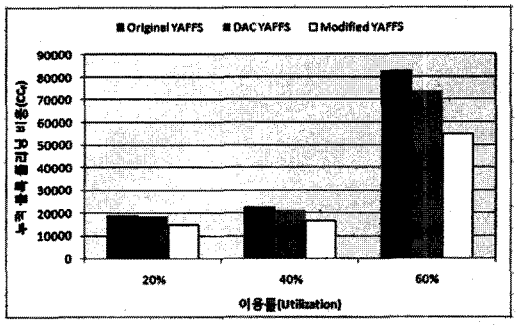

그림 14 이용률에 따른 누적 블록 클리닝 비용(CC_F)

같은 접근 방식을 사용하는데 다만 페이지 할당 기법을 다르게 적용한 것이다. 따라서 위의 실험 결과는 본 논문에서 제안한 페이지 할당 기법이 이전에 제안되었던 DAC 기법을 좀 더 효율적으로 만들어 준다는 것을 보여 준다.

4.2.2 읽기/쓰기 연산의 비율에 따른 성능 평가

이 절에서는 Postmark에서 읽기 연산과 쓰기 연산의 수행 비율에 따른 블록 클리닝 비용을 측정하였다. 읽기 연산과 쓰기 연산의 수행 비율 x/y에서 Postmark의 읽기, 쓰기 연산에서 읽기 연산이 차지하는 비율을 x%, 쓰기 연산이 차지하는 비율을 y%라고 한다. 예를 들어 10/90이면 Postmark의 읽기, 쓰기 연산에서 읽기 연산이 차지하는 비율이 10%, 쓰기 연산이 차지하는 비율이 90%임을 의미한다. 플래시 메모리의 이용률은 40%로 설정하고, 읽기 연산과 쓰기 연산의 수행 비율을 10/90, 20/80, 30/70, 40/60, 50/50, 60/40, 70/30, 80/20, 10/90으로 변경해가면서 실험을 진행하였다. 다른 설정 값들은 4.2.1 절과 같다.

그림 15는 읽기(read) 연산과 쓰기(append) 연산의 비율에 따른 블록 클리닝 횟수를 나타내고, 그림 16은 읽기 연산과 쓰기 연산의 비율에 따른 유효 페이지 복사 횟수를 나타낸다. 읽기 연산의 비율이 증가할수록 수정된 YAFFS 와 기존의 YAFFS 모두 블록 클리닝 횟수와 유효 페이지 복사 횟수가 최대 50% 이상 줄어드는 것을 확인할 수 있다. 이는 읽기 연산이 파일 메타 정보를 저장하는 헤더 청크에만 접근하고, 실제 데이터를 저장하고 있는 데이터 청크에는 접근하지 않아, 쓰기 연산에 비해 무효화 되는 페이지의 개수가 작기 때문이다.

블록 클리닝 횟수는 읽기 연산과 쓰기 연산의 비율이 10/90일 경우, 수정된 YAFFS가 기존의 YAFFS 보다 약 31%의 감소를 보였으며, 90/10일 경우 약 17%의 감소를 보였다. 즉, 읽기 연산의 비율이 커질수록 기존의 YAFFS와 수정된 YAFFS 사이의 성능 차이가 거의 반으로 줄어들었다. 반면 유효 페이지 복사 횟수는 수정된 YAFFS가 기존의 YAFFS에 비해 읽기 연산과 쓰기 연산의 비율에 관계없이 50%~60% 사이에 있음을 알 수 있다. 이것은 제안하는 페이지 할당 기법이 읽기 연산의 비중이 증가할수록 블록 클리닝 횟수 자체를 감소시키는 효율은 떨어지지만, 유효 페이지의 복사 횟수를 줄여주는 효율은 읽기 연산과 쓰기 연산의 비율에 상관없이 계속해서 유지된다는 것을 보여준다. 이는 유효 페이지의 복사 횟수가 연산의 종류보다는 지역성이나 이용률에 좌우되기 때문이다. 왜냐 하면 제안한 페이지 할당 기법은 이벤트 윈도우에 의해 cold 블록 하나당 존재하는 유효 페이지의 개수가 기존의 YAFFS에서 블록 하나당 존재하는 유효 페이지의 개수보다 적을 수밖에 없는데 그 둘 사이의 비율은 쓰기 연산의 빈도보다는 지역성과 이용률에 의해 좌우되기 때문이다.

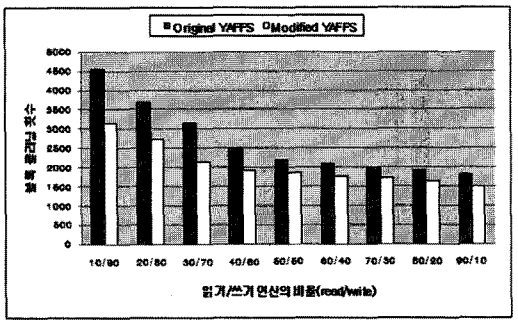

그림 15 읽기/쓰기 연산 비율에 따른 블록 클리닝 횟수

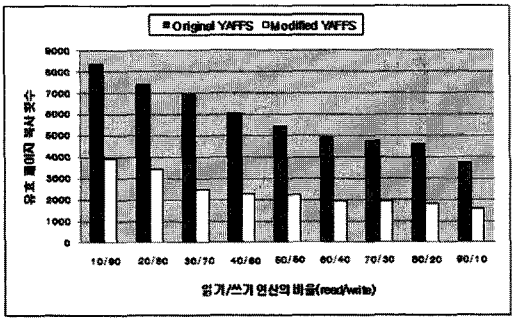

그림 16 읽기/쓰기 연산 비율에 따른 유효 페이지 복사 횟수

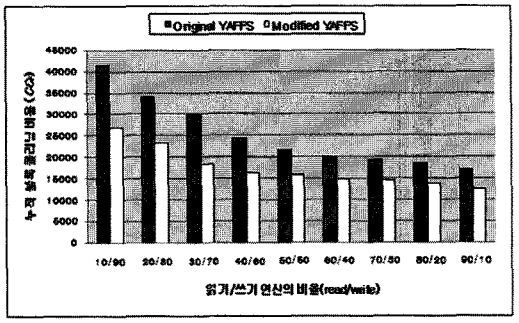

그림 17 읽기/쓰기 연산 비율에 따른 누적 블록 클리닝 비용(CC_F)

그림 17은 읽기 연산과 쓰기 연산의 비율에 따른 누적 블록 클리닝 비용(CC_F)을 나타낸다. 읽기 연산의 비율이 증가할수록 기존의 YAFFS와 수정된 YAFFS 모두 누적 블록 클리닝 비용이 감소하였으며, 두 시스템 간의 성능차이는 줄어들었다. 하지만 수정된 YAFFS는 읽기 연산과 쓰기 연산의 비율이 변하더라도 기존의 YAFFS와 비교하여 평균 31%의 누적 블록 클리닝 비용 감소를 보였으며, 읽기 연산 비율이 가장 높은 90/10인 상황에서도 기존의 YAFFS에 비해 약 26%의 누적 블록 클리닝 비용 감소를 보였다. 이는 제안한 페이지

할당 기법이 읽기 연산이 주로 이루어지는 환경에서도 기존의 YAFFS 보다 블록 클리닝 비용을 줄이는 데에 효과적임을 알 수 있다.

4.2.3 지역성에 따른 성능 평가

이번 절에서는 제안한 페이지 할당 기법이 지역성을 가지는 쓰기 요청에 대하여 어떠한 성능을 보이는지 기존의 YAFFS와 비교 평가하였다. 지역성은 x/y에서 전체 파일 중 x% 파일에 전체 연산 중 y%의 연산이 집중되는 것을 나타낸다[18]. 따라서 50/50은 지역성이 낮고, 10/90은 지역성이 높다. 낸드 플래시 메모리의 이용률이 약 30%일 때, 지역성을 50/50, 40/60, 30/70, 20/80, 10/90으로 변경해가며 실험을 진행하였다.

읽기 연산과 쓰기 연산은 각 30,000 번 수행하였으며, 연산이 발생하는 파일은 의사 난수(pseudo-random number)를 사용하여 선택하였다. 쓰기 연산과 읽기 연산의 단위는 512Bytes로 설정하였다.

그림 18은 지역성에 따른 블록 클리닝 횟수를 보여준다. 기존의 YAFFS는 지역성이 50/50일 때와 비교하여 지역성이 10/90일 때 블록 클리닝 횟수가 약 4% 감소하였다. 반면 수정된 YAFFS는 지역성이 50/50일 때와 비교하여 지역성이 10/90일 때 블록 클리닝 횟수가 약 40% 감소하였다. 즉, 기존의 YAFFS는 지역성에 따른 블록 클리닝의 횟수 감소가 아주 작은데 비해, 수정된 YAFFS는 지역성이 커짐에 따라 블록 클리닝의 횟수가 큰 폭으로 감소하는 것을 알 수 있다. 기존의 YAFFS와 수정된 YAFFS와의 성능 차이 역시 지역성이 커질수록 증가하였다. 지역성이 50/50일 때 수정된 YAFFS는 블록 클리닝 횟수가 기존의 YAFFS 보다 약 5% 감소하였고, 지역성이 10/90일 때에는 약 40%가 감소하였다.

그림 19는 지역성에 따른 유효 페이지 복사 횟수를 나타낸다. 기존의 YAFFS는 지역성이 50/50일 때와 비교하여 지역성이 10/90일 때 유효 페이지 복사 횟수가 약 4% 감소하였고, 수정된 YAFFS는 지역성이 50/50일 때와 비교하여 지역성이 10/90일 때 약 34%가 감소하였다. 그리고 수정된 YAFFS는 지역성이 50/50일 때

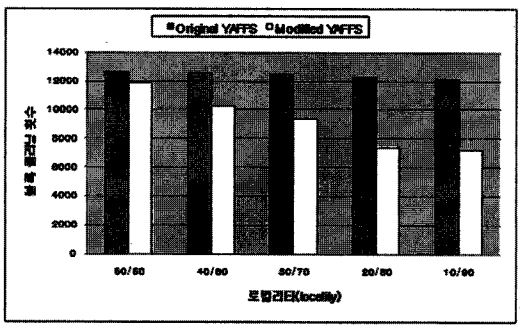

그림 18 지역성에 따른 블록 클리닝 횟수

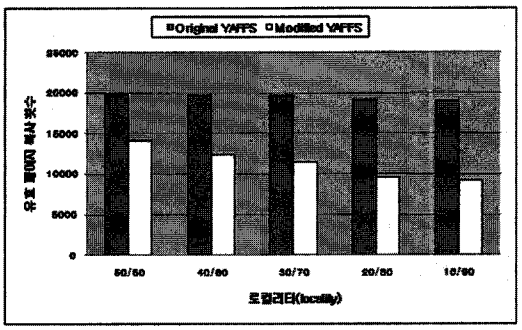

그림 19 지역성에 따른 유효 페이지 복사 횟수

로 기존의 YAFFS와 비교하여 유효 페이지 복사 횟수가 약 28% 감소하였고, 지역성이 10/90일 때는 기존의 YAFFS 보다 약 51% 감소하였다. 이용률이 커질수록 제안 하는 페이지 할당 기법이 블록 클리닝 횟수와 유효 페이지 복사 횟수를 효과적으로 줄이고 있음을 알 수 있다. 이는 지역성이 커질수록 좁은 영역에 대부분의 연산이 집중되어 특정 파일에서 업데이트가 발생할 확률이 높아지기 때문이다. 즉, 지역성이 10/90일 때 전체 파일 중 10%의 파일에 전체 연산 중 90%가 집중되기 때문에 10%의 파일에서 업데이트가 계속 발생할 확률이 높고, 업데이트가 계속 발생하다는 것은 그만큼 무효화가 빨리 진행된다는 것을 의미한다. 10%의 파일들은 Hot 페이지를 위한 블록에 할당될 확률이 높다. 따라서 희생블록으로 선택되었을 경우, 그 블록 안의 페이지가 대부분 무효화 상태에 있기 때문에 이를 통해 유효 페이지 복사 횟수를 줄일 수 있다. 나머지 90% 파일에는 전체 연산 중 10%의 연산만 발생하기 때문에 대부분의 파일들이 거의 수정이 되지 않거나 수정이 되더라도 그 횟수가 많지 않다. 따라서 이 페이지들은 Cold 블록에 할당될 것이며, 이를 통해 희생 블록으로 선택되는 것을 지연시켜 블록 클리닝 횟수를 감소시키고, 불필요하게 Cold 페이지가 재복사 되는 것을 막을 수 있다.

그림 20은 지역성을 적용한 성능 평가 중, 플래시 메모리에서 할당한 블록의 개수이다. 지역성 50/50에서 수정된 YAFFS는 기존의 YAFFS 보다 할당된 블록 개수가 약 5%가 감소하였으며, 지역성이 10/90에서 약 43%가 감소하였다. 할당된 블록의 개수는 그림 18의 블록 클리닝 횟수와 비슷한 비율로 감소한다. 이는 제안한 페이지 할당 기법이 Cold 페이지와 Hot 페이지를 적절히 분류하여 각 블록의 이용률을 높임으로써 블록 클리닝 횟수를 줄이는데 효과적임을 의미한다.

그림 21은 누적 블록 클리닝 비용을 나타낸다. 기존의 YAFFS의 경우 지역성에 관계없이 누적 블록 클리닝 비용이 거의 일정한 반면 수정된 YAFFS는 지역성이

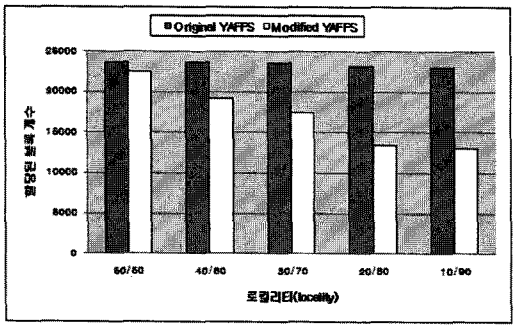

그림 20 할당된 블록 개수

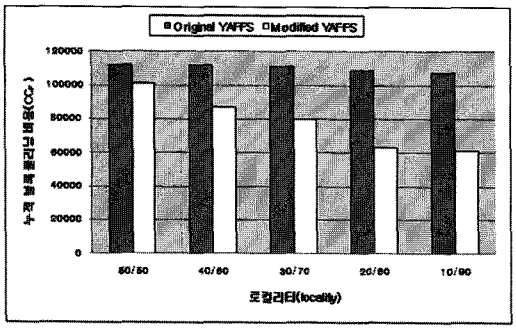

그림 21 지역성에 따른 누적 블록 클리닝 비용(CC_F)

증가할수록 누적 블록 클리닝 비용이 감소하는 것을 볼 수 있다. 수정된 YAFFS는 지역성이 50/50에서 기존의 YAFFS에 비해 약 9%의 성능 향상을 보이며, 지역성이 10/90에서 약 42%의 성능 향상을 보인다. 따라서 제안한 페이지 할당 기법이 지역성을 가지는 쓰기 패턴에서 블록 클리닝 비용을 줄이는데 효과적임을 확인할 수 있다.

4.2.4 이벤트 윈도우 크기와 Hot 페이지의 임계값

본 논문에서 제안하는 페이지 할당 기법은 적절한 이벤트 윈도우 크기와 Hot 페이지 임계값에 따라 성능이 달라진다. 따라서 이벤트 윈도우의 크기와 Hot 페이지 임계값을 결정하는 것이 중요하다. 이 두 값은 Postmark 수행 결과, 누적 블록 클리닝 비용(CC_F)이 가장 작은 것으로 결정하였다.

Postmark의 파라미터로 파일 크기는 512Bytes에서 4KBytes까지로 하였고, 트랜잭션은 10,000회, 읽기 연산과 쓰기 연산의 비율은 1:1, 읽기 연산과 쓰기 연산의 단위는 512 Bytes로 설정하였다.

그림 22는 이벤트 윈도우 크기에 따른 누적 클리닝 비용(CC_F)을 보여준다. 이벤트 윈도우의 크기가 150 전후 일 때, 누적 블록 클리닝 비용(CC_F)이 가장 작게 나타난다. 그림 23은 Hot 페이지 임계값에 따른 누적 블록 클리닝 비용(CC_F)을 나타낸다. Hot 페이지의 임

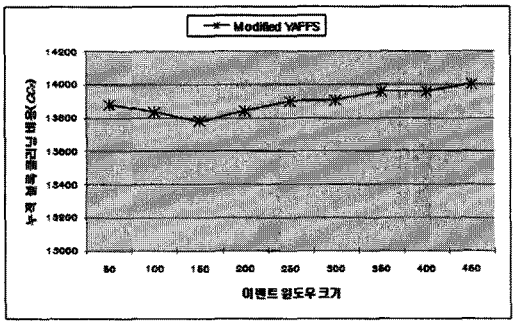

그림 22 이벤트 윈도우 크기에 따른 누적 블록 클리닝 비용(CC_F)

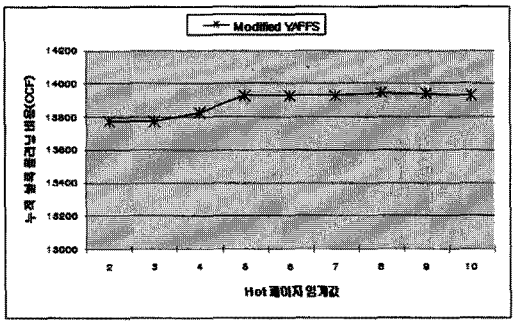

그림 23 Hot 페이지 임계값에 따른 누적 블록 클리닝 비용(CC_F)

계값이 2일 때, 누적 블록 클리닝 비용이 가장 작게 나타낸다. 그리고 그림에서 제시된 값보다 큰 값들을 가질 때는 일정하게 전체 성능에 큰 변화를 주지 않았다. 따라서 본 논문에서는 이벤트 윈도우의 크기는 150, Hot 페이지 임계값은 2로 설정하여 실험을 진행하였다.

5. 결론 및 향후 과제

본 논문에서는 업데이트 패턴에 따라 페이지를 Hot 페이지와 Cold 페이지로 구분하고 이들을 서로 다른 공간에 할당함으로써, 유효 페이지와 무효 페이지가 같은 블록에서 섞이는 것을 최소화하여 블록 클리닝 비용을 줄일 수 있는 페이지 할당 기법을 제안하였다. 또한 현재 가장 많이 사용하고 있는 낸드 플래시 메모리 전용 파일 시스템인 YAFFS에 직접 구현하여 기존의 YAFFS와 비교 평가를 수행하였다.

성능 평가 결과 본 논문에서 제안하는 페이지 할당 기법은 Postmark 벤치 마크에서 기존의 YAFFS에 비해 누적 블록 클리닝 비용을 평균 31% 이상의 감소시켰으며, 플래시 메모리의 이용률이 증가할수록 기존의 YAFFS 보다 더 좋은 성능 향상을 보였다. 또한 지역성을 적용한 성능 평가에서는 기존의 YAFFS 보다 평

균 28% 이상의 성능 향상을 보였으며, 지역성이 커질수록 제안하는 페이지 할당 기법이 블록 클리닝 비용을 줄이는데 효과적임을 알 수 있었다.

본 논문에서는 쓰기 패턴이 지역성을 가진다는 가정하에 이벤트 윈도우(Event Window)라는 자료 구조를 사용하여 Hot 페이지와 Cold 페이지를 분류하였다. 하지만, 분류된 페이지들이 정말로 Hot 페이지 인지, 또는 Cold 페이지 인지에 대한 검증은 이루어지지 않았다. 따라서 향후 Hot 페이지와 Cold 페이지를 검증 할 수 있는 모델 제안 및 실험 환경을 추가할 계획이다. 또한 본 논문에서는 플래시 메모리의 균등 사용(wear-leveling) 정책을 고려하지 않았다. 블록의 소거 횟수가 제한되어 있는 플래시 메모리에서 블록의 균등 사용은 많은 연구에서 다루고 있는 중요한 이슈 중 하나이다. 따라서 향후 균등 사용 정책을 추가하여 제안한 페이지 할당 기법을 확장 시키는 방향으로 나아갈 것이다.

참 고 문 헌

[1] 배영현, "고성능 플래시 메모리 SSD(Solid State Disk) 설계 기술", *정보과학회지 제25권 제6호 통권 제217호*, pp.18-28, 2007.6.

[2] 김성관, "플래시 파일 시스템 기술 소개", *정보과학회지 제25권 제6호 통권 제217호*, pp.11-17, 2007.6.

[3] 정보통신연구진흥원, "PC 플래시 메모리 시장 전망", 정보통신연구진흥원 학술정보 - IT World Newsletter 2권 7호.

[4] M. L. Chiang and R. C. Chang, "Cleaning policies in mobile computers using flash memory," *The Journal of Systems and Software*, vol.48, no.3, pp.213-231, 1999.

[5] Atsuo Kawaguchi, Shingo Nishioka, and Hiroshi Motoda, "A Flash-Memory Based File System," *Proceedings of 1995 USENIX Technical Conference*, pp.155-164, 1995.

[6] M-L. Chiang, P. C. H. Lee and R-C. Chang, "Using data clustering to improve cleaning performance for flash memory," *Software: Practice and Experience*, vol.29, no.3, pp.267-290, 1999.

[7] 백승재, 최종무, "플래시 메모리 파일 시스템을 위한 순수도 기반 페이지 할당 기법에 대한 연구", *한국 정보처리 학회 논문지A*, pp.387-398, 2006.

[8] Seungjae Baek, Jongmoo Choi, Donghee Lee, Sam H. Noh, "Performance Characteristics of Flash Memory: Model and Implications," *ICESS 2007, LNCS 4523*, pp.162-173, 2007.

[9] TOSHIBA AMERICA ELECTORONIC COMPONENTS, INC., "NAND vs. NOR Flash Memory: Technology Overview," www.chips.toshiba.com

[10] Micron, "TN-29-07:Small-Block vs. Large-Block NAND Flash Devices"

[11] Seungjae Baek, Seongjun Ahn, Jongmoo Choi, "Uniformity Improving Page Allocation for Flash Memory File Systems," *EMSOFT'07*, pp.154-163, 2007.

[12] YAFFS Spec, http://www.yaffs.net/yaffs-spec

[13] 이태훈, 정기동, "임베디드 시스템을 위한 신뢰성 있는 NAND 플래시 파일 시스템의 설계", *한국정보처리학회논문지 A 제12-A권 제7호*, pp.571-582, 2005.12

[14] 김기영, 손성훈, 신동하, "플래시 메모리 파일 시스템을 위한 가비지 콜렉터 설계 및 구현", *정보처리학회논문지 A 제14-A권 제1호*, pp.39-46, 2007.2

[15] Elizabeth J. O'Neill, Patrick E. O'Neill, Gerhard Weikum2, "The LRU-K Page Replacement Algorithm For Database Disk Buffering," *ACM SIGMOD* /5/93/Washington, DC, USA, 1993.

[16] 장승수, "플래시 메모리 파일 시스템 기술 동향", *정보산업진흥원 주간기술동향 통권 1425호*, 2009.12.

[17] J. Katcher, "PostMark: A new File System Benchmark," *Technical Report TR3022*, Network Appliance Inc., 1997.

[18] 이승환, 옥동석, 윤창배, 이태훈, 정기동, "페이지 비율 분석 기반의 NAND 플래시 메모리를 위한 가비지 컬렉션 기법", *정보과학회논문지:컴퓨팅의 실체 및 레터 제15권 제9호*, 2009.9.

김 희 태

2008년 경성대학교 컴퓨터공학과(학사) 2010년 부산대학교 대학원 컴퓨터공학과(석사). 2010년~현재 (주) 유노믹 부산연구소 연구원. 관심분야는 플래시 메모리, 파일 시스템

한 동 윤

2004년 부산대학교 정보컴퓨터공학부(학사). 2006년 부산대학교 대학원 컴퓨터공학과(석사). 2006년~현재 부산대학교 대학원 컴퓨터공학과 박사과정. 관심분야는 데이터베이스, 플래시 메모리, 파일 시스템

김 경 석

1977년 서울대학교 무역학과(경제학사) 1979년 서울대학교 전자계산학과(이학석사). 1988년 일리노이 주립대(어바나-샴페인) 전자계산학 박사. 1988년~1992년 미국 노스다코타 주립대학교 전자계학학과 조교수. 1992년~현재 부산대학교 전자전기정보컴퓨터공학부 교수. 관심분야는 데이터베이스, 멀티미디어, 한글/한말 정보처리, 인터넷 컴퓨팅 등