플래시 메모리 파일 시스템을 위한 순수도 기반 페이지 할당 기법에 대한 연구

백 승 재[†] · 최 종 무[††]

요 약

본 논문에서는 플래시 메모리 파일 시스템을 위한 새로운 페이지 할당 기법을 제안한다. 제안된 기법은 순수도를 고려하여 페이지를 할당하며, 이때 순수도는 플래시 메모리에서 유효한 페이지와 유효하지 않은 페이지가 공존하는 블록의 비율로 정의된다. 순수도는 플래시 메모리 파일 시스템의 블록 클리닝(block cleaning) 비용, 구체적으로 블록 클리닝을 수행할 때 복사해야할 페이지와 삭제해야할 블록의 양을 결정한다. 제안된 기법은 순수도를 향상시키기 위해 빈번하게 변경되는 데이터와 그렇지 않은 데이터를 구분하고, 이들을 서로 다른 블록에 할당한다. 데이터의 구분은 데이터의 속성 등의 정적 특성과 수행 시 변경 횟수 등의 동적 특성을 모두 고려한다. 제안된 기법은 내장형 보드와 YAFFS 상에 구현되었으며, 성능 분석 결과 기존 YAFFS에 비해 최대 15.4초 (평균 7.8초) 블록 클리닝 시간을 단축시켰다. 또한 이용율이 증가함에 따라 제안된 기법이 더욱 좋은 성능을 제공하였다.

키워드 : 플래시 메모리, 파일 시스템, 블록 클리닝, 순수도, 페이지 할당, 성능 평가, 내장형 시스템

A Study of Purity-based Page Allocation Scheme for Flash Memory File Systems

Seungjae Baek[†] · Jongmoo Choi[††]

ABSTRACT

In this paper, we propose a new page allocation scheme for flash memory file system. The proposed scheme allocates pages by exploiting the concept of purity, which is defined as the fraction of blocks where valid pages and invalid pages are coexisted. The purity determines the cost of block cleaning, that is, the portion of pages to be copied and blocks to be erased for block cleaning. To enhance the purity, the scheme classifies hot-modified data and cold-modified data and allocates them into different blocks. The hot/cold classification is based on both static properties such as attribute of data and dynamic properties such as the frequency of modifications. We have implemented the proposed scheme in YAFFS and evaluated its performance on the embedded board equipped with 400MHz XScale CPU, 64MB SDRAM, and 64MB NAND flash memory. Performance measurements have shown that the proposed scheme can reduce block cleaning time by up to 15.4 seconds with an average of 7.8 seconds compared to the typical YAFFS. Also, the enhancement becomes bigger as the utilization of flash memory increases.

Key Words : Flash Memory, File System, Block Cleaning, Purity, Page Allocation, Performance Evaluation, Embedded System

1. 서 론

저장 장치의 특징은 그 저장 장치를 관리하는 파일 시스템의 설계와 성능에 큰 영향을 준다. 예를 들어 디스크는 헤드의 탐색 거리(seek distance)가 성능에 큰 영향을 주는 특징 중에 하나이며, 디스크를 위한 파일 시스템은 탐색 거리를 줄이기 위해 실린더 기반 블록 할당(cylinder based allocation), 요청 통합(request grouping), 최적 탐색 거리 우선 스케줄링 (shortest seek time first scheduling) 등의 기법을 사용한다[1]. 그러나 플래시 메모리는 디스크와는 비교되는 특징들을 가지고 있기 때문에 기존 기법을 그대로 사용하는 것은 비효율적이다. 로그 구조 파일 시스템(LFS: Log-structured File System)에서의 세그먼트 클리닝이 로그 구조 파일 시스템의 성능에 큰 영향을 끼치는 것처럼 [2-6], 블록 클리닝은 플래시 메모리 파일 시스템의 성능에 큰 영향을 끼친다[7-10]. 이는 플래시 메모리의 삭제 연산이

※ 이 연구는 2004년도 단국대학교 대학연구비의 지원으로 연구되었음.
† 정 회 원 : 단국대학교 대학원 정보컴퓨터과학과 석사과정
†† 종신회원 : 단국대학교 정보컴퓨터학부 컴퓨터과학 전공 조교수
논문접수 : 2006년 3월 24일, 심사완료 : 2006년 7월 24일

쓰기 연산에 비해 무척 느린 연산이기 때문이며, 따라서 응용의 수행 중에 요구 삭제(on-demand erase)가 발생하면 응용의 응답 시간을 크게 저하시킨다. 응용의 수행 중에 요구 삭제 횟수에 영향을 주는 것이 바로 블록 클리닝이므로, 블록 클리닝은 플래시 메모리 파일 시스템의 처리율과 응용의 대기 시간에 크게 영향을 준다고 할 수 있다.

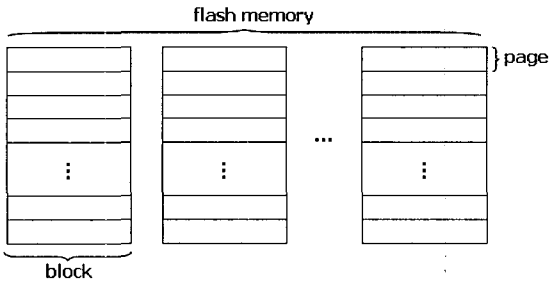
본 논문에서는 블록 클리닝 비용을 분석하기 위한 성능 모델을 제안한다. 우선 플래시 메모리의 특징들로부터 블록 클리닝에 영향을 주는 성능 인자들(performance parameters)을 추출한다. 그리고 성능 인자들이 블록 클리닝 비용에 어떤 영향을 주는지 나타내는 수식을 제안한다. 성능 모델의 분석 결과 순수도의 향상이 블록 클리닝 비용을 크게 줄일 수 있음을 알 수 있다. 또한 본 논문에서는 순수도를 향상시킬 수 있는 새로운 페이지 할당 기법을 제안한다. 제안된 기법의 기본 아이디어는 자주 수정되는 데이터(hot-modified data)들과 그렇지 않은 데이터(cold-modified data)들을 구분하고 서로 다른 블록에 할당하는 것이다.

본 논문의 구성은 다음과 같다. 2절에서는 플래시 메모리의 특징과 동작 원리를 소개한다. 3절에서는 블록 클리닝 비용 분석을 위한 모델을 설명하며, 4절에서는 순수도를 고려한 블록 할당 정책을 제안한다. 5절에서는 제안된 기법의 구현 내용을, 6절에서는 성능 분석 결과를 설명한다. 관련 연구는 7절에 기술되며, 마지막 8절에서 결론 및 향후 연구 방향을 소개한다.

2. 플래시 메모리 특징과 동작 원리

현재 일반 적으로 사용되고 있는 플래시 메모리는 크게 NOR 플래시 메모리와 NAND 플래시 메모리로 구분할 수 있다. 본 논문에서는 NAND 플래시 메모리를 주 고려 대상으로 한다. 하지만 본 논문의 연구 결과는 NOR 플래시 메모리에서도 대부분 적용될 수 있다.

(그림 1)은 NAND 플래시 메모리의 일반적인 구조를 보여준다. 플래시 메모리는 블록들로 구성되며 각 블록은 다시 페이지들로 구성된다. NAND 플래시 메모리는 소블록(small block) NAND와 대블록(large block) NAND로 세분될 수 있는데[11, 12], 소블록의 경우 블록은 32개의 페이지로 구성되며 각 페이지는 528 bytes(512 bytes for data + 16 bytes for spare area)의 크기를 갖는다. 반면 대블록의 경우 블록은 64개의 페이지로 구성되며 각 페이지는 2112 bytes(2048 bytes for data + 64 bytes for spare area)의 크기를 갖는다.



(그림 1) 플래시 메모리 구조

플래시 메모리는 읽기(read), 쓰기(write), 삭제(erase)라는 3가지 기본 연산을 제공한다. 플래시 메모리의 기본 연산과 관련되어 본 논문에서는 다음 2가지 특징에 주목한다. 첫째, 읽기/쓰기 연산과 삭제 연산의 수행 단위가 다르다[13]. 둘째, <표 1>에서 볼 수 있듯이 3가지 기본 연산의 수행 시간은 서로 다르다[14, 15]. 셋째, 플래시 메모리에는 탐색 연산이 없다. 넷째 플래시 메모리는 이미 쓰여진 공간에 덮어쓰기(overwrite)를 할 수 없는 덮어쓰기 제한(overwrite limitation)이 있다. 다섯째, 플래시 메모리의 각 블록에 수행될 수 있는 삭제 연산의 횟수가 제한되어 있다.

플래시 메모리의 덮어쓰기 제한과 수행 단위의 차이는 파일 시스템에서 쓰기 요청의 처리를 복잡하게 한다. 이 문제를 해결하기 위해 많은 플래시 메모리 파일 시스템은 수정 요청을 처리할 때 페이지를 새로 할당 받고 여기에 새로운 데이터를 쓴 뒤 기존의 불필요한 페이지들을 무효화 시키는 방법(non in-place update)을 사용한다.

무효화된 페이지를 수집하여 다시 쓰기 가능한 상태로 전환하는 작업이 블록 클리닝(block cleaning)이다. 블록 클리닝은 무효화된 페이지가 존재하는 블록을 선택하고, 이 블록에 유효한 페이지가 존재하면 다른 공간으로 복사하고(복사는 매핑 정보의 수정을 동반한다), 그 블록을 삭제하여 블록내의 모든 페이지를 쓰기 가능한 상태로 전환하는 작업을 수행한다. 블록 클리닝 기법은 언제, 어떤 블록을, 얼마나 수집할 것인가를 고려하여 설계 되어야 하며 또한 삭제 연산의 균등화를 고려해야 한다[16].

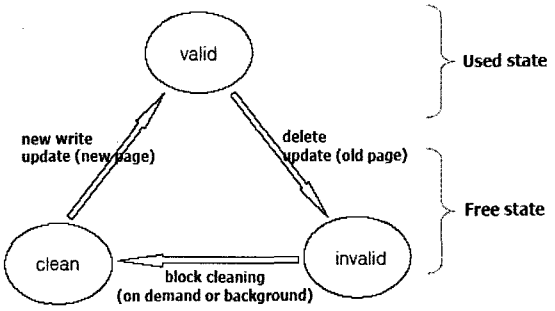
(그림 2)는 플래시 메모리 파일 시스템에서 페이지의 상태와 전이를 보여준다. 각 페이지는 유효(valid), 무효(invalid), 클린(clean)이라는 3가지 상태로 존재할 수 있다[2]. 블록이 클리닝되면 그 블록에 속해있던 모든 페이지는 클린 상태가 된다. 그 페이지에 새로운 데이터를 쓰면 그 페이지는 유효 상태가 된다. 반면 그 데이터를 삭제하면 해당 페이지는 무효 상태가 된다. 한편, 기존의 데이터를 수정하는 경우 새로운 데이터가 쓰이는 페이지는 유효 상태가 되며, 기존의 데이터가 존재하던 페이지는 무효 상태가 된다. 이

〈표 1〉 플래시 메모리 기본 연산 수행 시간

Media	Access time		
	Read	Write	Erase
DRAM	60ns (2B) 2.56us (512B)	60ns (2B) 2.56us (512B)	N/A
NOR Flash	150ns (1B) 14.4us (512B)	211us (1B) 3.53ms (512B)	1.2s (128KB)
NAND Flash	10.2us (1B) 35.9us (512B)	201us (1B) 226us (512B)	2ms (16KB)
Disk	12.4ms (512B) (average)	12.4ms (512B) (average)	N/A

2) 디스크에서 각 섹터는 사용 중(used), 사용 가능(free)라는 2가지 상태로 존재한다.

러한 페이지의 상태와 전이가 플래시 메모리 파일 시스템의 성능 특성에 어떤 영향을 주는가? 클린 상태의 페이지들이 충분히 존재할 때 쓰기 요청은 플래시 메모리에 대한 쓰기 연산과 매핑 정보 수정 연산으로 서비스 될 수 있다. 하지만 클린 상태의 페이지가 부족할 때 쓰기 요청은 삭제 연산을 필요로 한다. 이때 클린 상태의 페이지를 만드는 작업이 블록 클리닝이므로 블록 클리닝은 플래시 메모리 파일 시스템의 성능에 큰 영향을 준다고 할 수 있다. 본 논문에서는 블록 클리닝의 비용 분석과 효과적인 블록 클리닝 기법을 중점적으로 연구한다.



(그림 2) 페이지 상태와 전이

3. 블록 클리닝 비용 분석

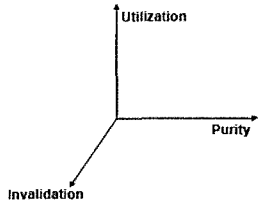
3.1 성능 인자와 비용

블록 클리닝은 무효 상태의 페이지가 존재하는 블록들을 선택하고, 만일 이 블록에 유효한 페이지들이 존재하면 이들을 복사한 후 블록을 삭제하는 작업으로 구성된다. 결국 블록 클리닝 비용은 페이지 복사 비용과 블록 삭제 비용으로 구성되는 것이다. 본 논문에서는 블록 클리닝의 비용에 영향을 주는 성능 인자(performance parameter)로 다음 3가지를 고려한다.

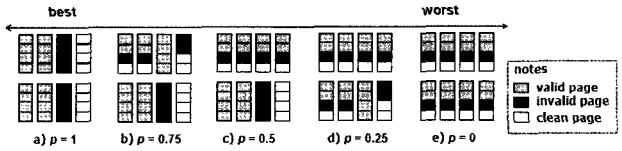
- 이용율(utilization) : 플래시 메모리에서 유효 상태의 페이지 비율
- 무효율(invalidation) : 플래시 메모리에서 무효 상태의 페이지 비율
- 순수도(purity) : 플래시 메모리에서 무효 상태의 페이지 분포

이용율은 블록 클리닝을 수행할 때 복사되는 페이지의 최대 개수를 결정한다. 반면, 무효율은 블록 클리닝을 수행할 때 삭제되는 블록의 최대 개수를 결정한다. 그리고 순수도(p)는 블록 클리닝을 수행할 때 실제로 복사되는 페이지와 삭제되는 블록 개수를 결정한다. (그림 3)은 3가지 성능 인자를 도시한 것이다.

순수도는 '무효 상태의 페이지들이 플래시 메모리 상에서 어떻게 분포하는가'로 정의된다. 예를 들어 무효 상태의 페이지들이 플래시 메모리의 일부 공간에 모여 있다면 좋은



(그림 3) 3가지 성능 인자



(그림 4) 순수도 스펙트럼

순수도이며, 만일 전체 공간에 고루 퍼져 있다면 좋지 않은 순수도이다. 순수도를 정량적으로 파악하기 위해 본 논문에서는 순수도를 "1 - 유효 상태의 페이지와 무효 상태 페이지가 함께 존재하는 블록의 비율"로 구체화하여 정의하였다. 이렇게 정의하면 우리는 플래시 메모리의 특정 상태에서 순수도를 정량적으로 측정할 수 있다.

(그림 4)는 순수도 스펙트럼을 예시한 것이다. 그림에서 a), b), c), d), e)는 각각 특정 플래시 메모리 상태이다. 각 상태에서 플래시 메모리는 8개의 블록으로 구성되며, 각 블록에는 4개의 페이지가 존재한다고 가정하자. 그리고 회색 사각형으로 표현된 페이지는 유효 상태의 페이지이며, 어두운 사각형으로 표현된 페이지는 무효 상태의 페이지이며, 흰 사각형은 클린 상태의 페이지를 의미한다. 결국 그림에서 a) ~ e) 모두 이용율은 0.5, 무효율은 0.25이다. (그림 4)의 a)는 유효 상태의 페이지와 무효 상태의 페이지가 공존하는 블록이 없다. 따라서 순수도(p)는 1이다. 반면 (그림 4)의 b)는 2 블록에서 유효 상태의 페이지와 무효 상태의 페이지가 공존하며, 따라서 순수도(p)는 0.75이다. 가장 오른쪽에 있는 (그림 4)의 e)의 경우, 유효 상태의 페이지와 무효 상태의 페이지가 모든 블록에서 공존하며, 따라서 순수도(p)는 0.0이다. 한편, (그림 4)의 a)에서 모든 무효 상태의 페이지들을 재생하기 위해 필요한 블록 클리닝 비용은 2 블록의 삭제이다. 반면, (그림 4) b)의 경우 클리닝 비용은 4 페이지의 복사와 4 블록의 삭제이며, (그림 4) e)의 경우 클리닝 비용은 16 페이지의 복사와 8 블록의 삭제이다.

지금까지 관찰한 블록 클리닝 비용과 성능 인자 간에 관계를 수식으로 표현하면 아래 수식과 같다.

블록 클리닝 비용 (block cleaning cost)

= 복사 비용 + 삭제 비용

$$= ((1-p) \times \min(B, i \times P)) \times ((r_t + w_t) \times (P / B \times u) + e_t)) + ((p \times B \times i) \times e_t) \quad \cdots\cdots 1)$$

where

- u : 이용율 ($0 \le u \le 1$)
- i : 무효율 ($0 \le u + i \le 1 - u$)
- p : 순수도 ($0 \le p \le 1$)
- P : 플래시 메모리 내의 페이지의 개수 (P = Capacity/size of page)
- B : 플래시 메모리 내의 블록 개수 (P / B : # of pages in a block)
- r_t : 읽기 연산 수행 시간
- w_t : 쓰기 연산 수행 시간
- e_t : 삭제 연산 수행 시간

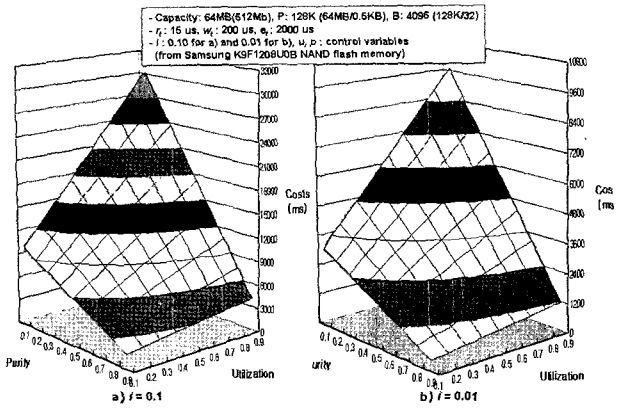
수식은 크게 2개의 항(term)으로 구성되는데, 첫 번째 항은 순수하지 않은 블록을 위한 복사 비용과 삭제 비용을 의미하며, 두 번째 항은 무효 상태의 페이지들로만 구성된 순수한 블록에서 삭제 비용을 의미한다. 수식에서 복사 비용을 읽기 연산과 쓰기 연산 시간의 합(r_t+w_t)으로 표현하였는데, 만일 플래시 메모리 내부에서 읽기와 쓰기를 통합한 재기록(copy-back) 연산을 지원하면[12], 복사 비용으로 이 연산 시간을 사용할 수도 있다. 일반적으로 재기록 연산 시간은 쓰기 연산 시간과 비슷하며, 결국 위 수식에서 r_t를 제외하면 그대로 적용된다.

3.2 비용 분석(Cost Analysis)

(그림 5)는 본 논문에서 제안한 수식을 이용하여 블록 클리닝 비용과 성능 인자 간에 관계를 분석한 결과를 보여준다. 그림에서 x 축은 이용율이고 y 축은 순수도이며, z 축은 블록 클리닝 비용이다. 그리고 (그림 5)의 a)는 무효율이 0.1일 때 결과이며, (그림 5)의 b)는 무효율이 0.01일 때 결과이다. 분석할 때 소블록 NAND인 삼성 K9F5608U0M NAND 메모리의 기본 데이터를 사용하였으며, 구체적으로 용량은 64MB, 페이지 크기는 512B (따라서 페이지 개수는 128K개), 블록 크기는 16KB (블록 개수는 4K개), 읽기/쓰기/삭제 연산은 평균적으로 15us, 200us, 2ms 걸린다[12].

(그림 5)에서 우리는 다음과 같은 것은 파악할 수 있다. 첫째, 예상대로 이용율이 증가함에 따라 비용이 증가한다. 둘째, 순수도가 나빠짐에 따라 비용이 증가한다. 셋째, 무효율이 0.01일 때에 비해 무효율이 0.1이 되면 비용이 증가한다. 이는 (그림 5)의 a)에서 z축 최대 값은 33000ms인 반면 (그림 5)의 b)에서 z축 최대 값은 10800ms임을 통해 알 수 있다. 하지만 이용율과 순수도의 변화에 따른 비용의 변화는 무효율이 0.01일 때와 0.1일 때 유사하다. 넷째, 이용율과 순수도 모두 나빠질 때 비용은 급격히 증가한다. 반면 이용율이나 순수도 중에서 하나를 좋은 상태로 유지할 수 있으면 비용이 급격히 증가하지 않는다.

다른 용량의 소블록 NAND 플래시 메모리와 대블록 NAND 플래시 메모리에 대하여 본 논문에서 제안한 수식을



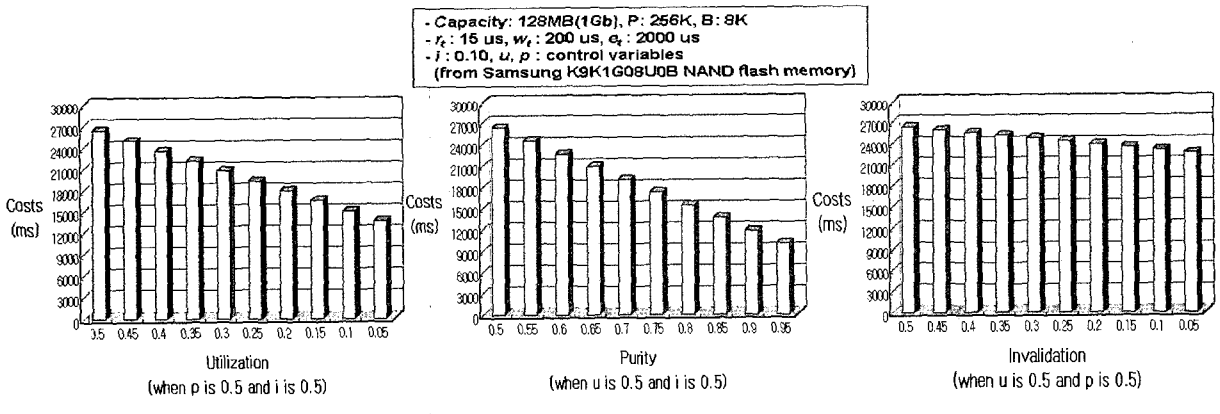
(그림 5) 클리닝 비용과 성능 인자 관계

이용하여 블록 클리닝 비용과 성능 인자 간의 관계를 분석한 결과는 (그림 5)의 경향과 유사하다. 따라서 본 논문에서 제시된 수식이 다양한 용량의 플래시 메모리와 대블록 NAND 플래시 메모리의 분석에도 사용될 수 있음을 알 수 있다.

(그림 6)은 각 성능 인자가 비용에 끼치는 영향을 분석한 것이다. 이 분석은 (그림 5)와 동일한 데이터를 사용하였다. 각 그림은 이용율과 무효율이 감소할 때, 그리고 순수도가 증가할 때 블록 클리닝 비용이 어떻게 변화하는 지를 보여준다. 무효율이 감소함에 따라 얻어지는 비용 감소는 상대적으로 적었고, 이용율 감소와 순수도의 증가는 비용을 크게 감소시켰다. 결국 플래시 메모리에서 자주 블록 클리닝을 수행하여 무효율을 줄이는 것 보다는 순수도를 좋게 하는 것이 성능 향상에 더 효과적임을 알 수 있다.

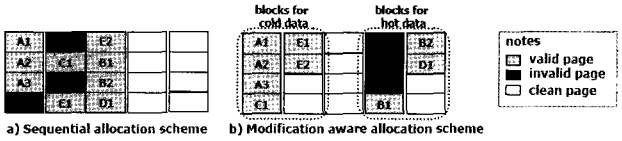
4. 순수도 기반 페이지 할당 기법

블록 클리닝 비용은 이용율, 무효율 그리고 순수도에 의해 영향을 받는다. 이때 무효율과 순수도는 파일 시스템 수준에서 제어 가능하다. 무효율의 경우 파일 시스템이 관리하는 버퍼 캐시에서 지연 쓰기(delayed write)를 이용하면



a) Effects of Utilization b) Effects of Purity c) Effects of Invalidation

(그림 6) 클리닝 비용과 성능 인자 관계: 각 성능 인자의 영향



(그림 7) 페이지 할당 기법

감소시킬 수 있다. 또한, 파일 시스템의 페이지 할당 기법과 블록 클리닝 기법을 이용하여 순수도를 제어할 수 있다. 본 논문에서는 순수도를 향상시킬 수 있는 새로운 페이지 할당 기법을 제안한다.

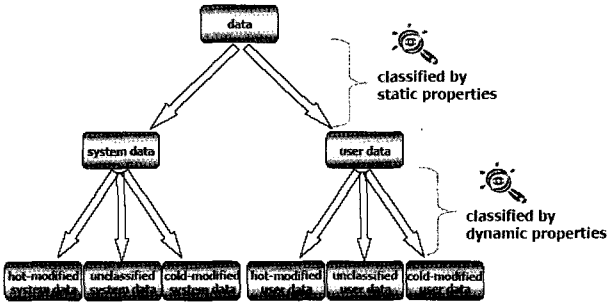
본 논문에서 제안하는 할당 기법은 데이터의 변경 특성을 고려한 할당(modification aware allocation) 기법이다. 이 기법의 기본 아이디어는 빈번하게 변경되는 데이터(hot data)와 그렇지 않은 데이터(cold data)를 구분하고, 이들을 서로 다른 블록에 할당하여 순수도를 향상시키겠다는 것이다. 빈번하게 변경되는 데이터(hot data)와 그렇지 않은 데이터(cold data)의 구분은 데이터 변경의 치우침 경향(skewness)을 이용하면 가능하다[6, 7]. 이때 치우침 경향이란 전체 데이터 중에 특정 부분이 다른 부분에 비해 더 높은 확률로 접근되는 프로그램 동작 특성을 의미한다.

(그림 7)은 본 논문에서 제안한 페이지 할당 기법의 기본 아이디어를 보여준다. 그림에서 각 블록은 4개의 페이지로 구성됨을 가정한다. 그리고 파일 A 생성 (3개의 페이지로 구성), 파일 B 생성 (2개 페이지로 구성), 파일 C 생성 (1개 페이지로 구성), 파일 D 생성 (1개 페이지로 구성), 파일 E 생성 (2개 페이지로 구성), 파일 B 수정, 파일 D 수정의 순으로 요청이 발생했다고 가정하자. 그리고 파일 B와 D는 빈번하게 변경됨을 이미 알고 있다고 가정하자. (그림 7)의 a)는 기존의 순차 할당 기법이며, (그림 7)의 b)는 제안된 변경 특성 고려 할당 기법이다. 그림에서 흰 사각형, 회색 사각형, 검은 사각형은 각각 클린 상태 페이지, 유효 상태 페이지, 무효 상태 페이지를 의미한다.

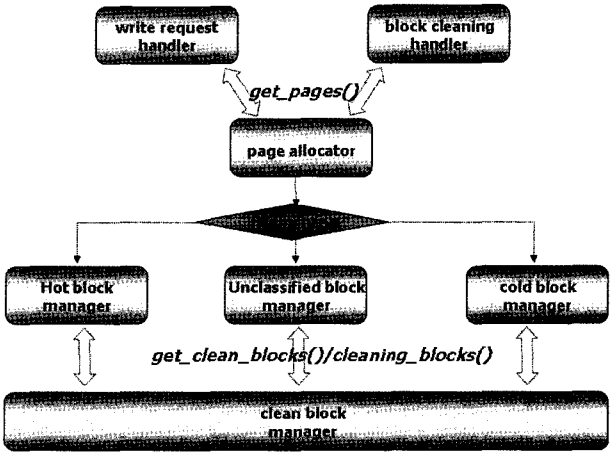
(그림 7)에서 전체 플래시 메모리의 블록 개수를 5로 가정하면, 기존의 순차 할당 기법을 적용한 상태에서 순수도는 0.6이며 제안된 기법에서는 0.8이다. 한편, 기존의 순차 할당 기법을 적용한 상태에서 모든 무효 상태의 페이지를 클리닝하려면 5번의 페이지 복사와 2번의 블록 삭제가 요구된다. 반면 제안된 기법에서는 1번의 페이지 복사와 1번의 블록 삭제가 요구된다. 결국 제안된 기법은 빈번하게 변경되는 데이터들과 그렇지 않은 데이터를 서로 다른 블록에 할당하였으며, 결국 플래시 메모리의 순수도를 향상시켜 블록 클리닝 비용을 감소시킬 수 있다.

이제 문제는 '빈번하게 변경되는 데이터(hot data)와 그렇지 않은 데이터(cold data)를 어떻게 분류하는가'이다. 본 논문에서는 데이터의 정적 속성(static property)과 동적 속성(dynamic property)을 이용하여 분류한다. (그림 8)은 변경 특성을 고려한 데이터의 분류 방법을 보여준다.

우선 데이터는 정적 속성에 의해 분류된다. 본 논문에서는 데이터가 시스템 데이터인지 아니면 사용자 데이터인지



(그림 8) 변경 특성을 고려한 데이터 분류



(그림 9) 데이터 분류 시점과 분류에 따른 페이지 할당

를 정적 속성으로 사용하였다. 예를 들어 파일 시스템의 수퍼 블록이나 inode같은 메타 데이터는 시스템 데이터이며, 사용자가 생성한 파일 내용은 사용자 데이터이다. 일반적으로 시스템 데이터와 사용자 데이터는 변경 빈도가 다르다. 예를 들어 파일의 속성을 관리하는 inode는 파일의 일부 내용이 수정될 때마다 변경되며, 결국 일반 사용자 데이터에 비해 더 큰 빈도로 변경된다. 따라서 본 논문에서는 우선적으로 데이터가 시스템 데이터인가 사용자 데이터인가에 따라 구분한다. 파일 시스템이 아닌 데이터베이스의 경우에는 인덱스인지 아니면 데이터인지를 정적 속성으로 사용할 수도 있다.

정적 구분 이후 단계는 동적 속성에 따른 분류이다. 본 논문에서는 일정 기간 동안 변경된 횟수를 동적 속성으로 사용한다. 그런데, 각 페이지 단위로 변경 횟수를 유지한다는 것은 관리 부하가 크다. 결국 본 논문에서 동적 분류는 이미 파일 시스템에서 지원하고 있는 파일별 변경 횟수를 이용하여 파일 단위로 이루어진다.

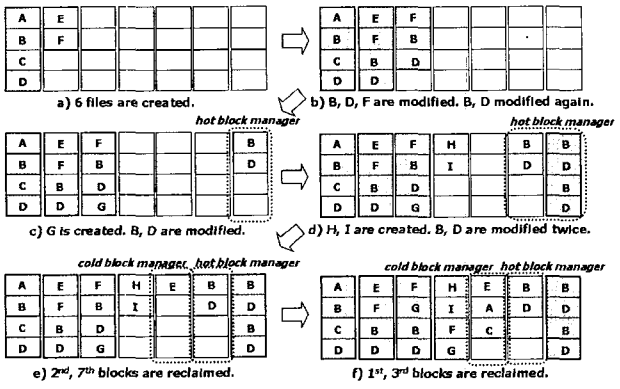
동적 분류는 구체적으로 Δt 시간 간격 내에 변경된 횟수가 h번 이상이면 자주 변경되는 데이터(hot-modified data)로 분류하며, 변경 횟수가 c번 이하이면 자주 변경되는 않는 데이터(cold-modified data)로 분류한다. 이때 Δt는 현재 시점에서 t시간 이전 시점까지 시간 간격으로, 시간은 쓰기 요청이 발생할 때마다 증가하는 논리적인 시간을 사용한다. 결국 Δt, h, c는 파일 시스템이 사용하는 제어 변수로, 실험

결과 h = 2, c = 0 그리고 Δt는 10이면 대부분의 벤치마크에서 제대로 분류하는 것으로 분석되었다 (실험 결과 절의 <표 3> 참조).

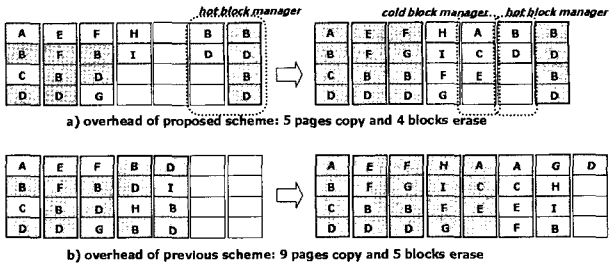
다음 문제는 분류를 시도하는 시점과 분류된 결과에 따라 페이지를 할당하는 방법이다. 자주 변경되는 데이터(hot-modified data)의 분류와 관리는 비교적 쉽다. 데이터에 대한 쓰기 요청을 서비스하기 전에 이 데이터가 속한 파일의 Δt 시간 동안 변경 횟수를 검사하여, 이 값이 h번 이상이면 자주 변경되는 데이터로 분류하면 된다. 그리고 자주 변경되는 데이터로 분류되면, 이 데이터를 위한 페이지는 자주 변경되는 데이터들을 위한 블록들에서 할당하면 된다. (그림 9)에서 hot block manager가 자주 변경되는 데이터들을 위한 블록들을 관리한다.

한편, 자주 변경되는 않는 데이터(cold-modified data)는 대부분 처음 생성된 이후로는 거의 수정되지 않기 때문에, 쓰기 요청을 처리하면서 발견하기는 어렵다. 따라서 이러한 데이터의 발견은 별도의 주기적인 디몬(daemon)을 사용하거나 다른 방법을 사용해야 한다. 본 논문에서는 추가적인 부하를 발생시키지 않기 위해 블록 클리닝 시점에 자주 변경되는 않는 데이터 발견을 시도한다. 구체적으로 클리닝을 하기로 결정한 블록에 여전히 유효한 데이터가 있으면, 이 데이터가 속한 파일의 Δt 시간 동안 변경 횟수를 검사하여 이 값이 c번 이하이면 자주 변경되지 않는 데이터로 분류한다. 그리고 이 데이터를 위한 페이지는 (그림 9)의 cold block manager가 관리하는 자주 변경되지 않는 데이터들을 위한 블록에서 할당한다. 이 방법은 실제 자주 변경되지 않는 데이터들을 모두 발견할 수 없다는 가능성이 있지만, 추가적인 부하 없이 발견 및 수집이 가능하다는 장점이 있다. 만일 약간의 부하를 소비하더라도 자주 변경되지 않는 데이터들을 모두 발견하고 따로 관리하는 것이 성능에 더 유리하다면 별도의 주기적인 디몬을 이용할 수도 있다.

(그림 10)은 제안된 페이지 할당 기법의 실제 동작 예를 보여준다. 그림에서 Δt, h, c는 각각 10, 2, 0으로 가정한다. 즉 최근 10번의 쓰기 요청이 발생하는 동안에 2번 이상 변경이 발생하면 자주 변경되는 데이터이며, 0번 이하 변경이 발생하면 자주 변경되지 않는 데이터로 분류되는 것이다.



(그림 10) 제안된 기법의 페이지 할당 예



(그림 11) 제안된 기법에서 예상되는 성능 향상

우선 A, B, C, D, E, F라는 6개의 파일이 생성되었다고 가정하자 (논의의 편의상 각 파일은 1개의 페이지로 구성된다고 가정한다). 이때 플래시 메모리의 상태가 (그림 10)의 a)에 나타나 있다.

(그림 10)의 a)에서 우선 B, D, F 파일이 수정되었다고 가정하자. 그리고 다시 B, D가 수정되었다고 가정하자. 이때 플래시 메모리의 상태는 (그림 10)의 b)에 나타나 있다. 이 상태에서 G 파일이 생성되고, 다시 B, D가 수정된다고 가정하자 ((그림 10)의 c)). 이때 B, D는 자주 변경되는 데이터로 파악되며, hot block manager가 관리하는 블록에서 페이지를 할당한다. 이후 다시 H 파일이 생성되고 B, D가 다시 수정되며, I 파일이 생성되고 B, D가 다시 수정된다고 가정하자 ((그림 10)의 d)). B, D는 여전히 hot block manager에 의해 관리되며 따라서 새로 생성된 파일들과 분리되어 관리된다.

이 상태에서 블록 클리닝이 호출된다고 가정하자. 블록 클리닝은 유효한 페이지가 적은 블록부터 재생하며 모든 무효화된 페이지를 수집할 때까지 수행된다고 가정하자. 그럼 (그림 10)의 d)에서 7번째 블록을 우선 선택하여 삭제한다. 그리고 2번째 블록을 선택하여 유효 상태인 E를 다른 곳으로 복사한다. 이때, E는 Δt 시간동안 변경이 되지 않았으며, 따라서 자주 변경되지 않는 데이터로 분류된다. 결국 cold block manager가 관리하는 블록에서 페이지가 할당되며, 이때 플래시 메모리의 상태가 (그림 10)의 e)에 나타나 있다. 이후 클리닝 과정에서 A, C도 자주 변경되지 않는 데이터로 파악되어 cold block manager에서 관리된다. 참고로 G는 변경 횟수는 0이지만 생성 이후 시간이 아직 Δt 만큼 지나지 않았기 때문에 자주 변경되지 않는 데이터로 분류되지 않았다. 한 가지 주의할 것은 hot block manager나 cold block manager에 의해 관리되는 블록의 개수는 고정되어 있는 것이 아니라, 페이지의 할당과 클리닝이 진행됨에 따라 동적으로 변하게 된다. (그림 9)의 *get_free_blocks()*와 *cleaning_blocks()* 인터페이스가 이 역할을 담당한다.

제안된 기법은 블록 클리닝의 비용에 어떤 영향을 끼치게 될까? (그림 11)은 제안된 변경 특성을 고려한 블록 할당 기법과 기존의 순차 할당 기법을 비교한 것이다. (그림 11)의 a)는 (그림 10)의 d)와 f)이다. 이 그림에서 우리는 블록 클리닝을 위해 5번의 페이지 복사와 4번의 블록 삭제가 요구됨을 알 수 있다. 한편, (그림 11)의 b)는 (그림 10)의 시나리오를 순차 할당으로 관리했을 때 상태이다. 이 그림에

서 우리는 블록 클리닝을 위해 9번의 페이지 복사와 5번의 블록 삭제가 요구됨을 알 수 있다. 또 한 가지 주목할 것은 블록 클리닝 이후의 플래시 메모리 상태이다. 제안된 기법은 자주 변경되는 데이터와 그렇지 않은 데이터가 다른 블록에서 관리되므로, 이후 페이지 할당이 진행됨에 따라 더 큰 순수도의 향상을 얻을 수 있다.

5. 구 현

본 논문에서 제안된 순수도 기반 페이지 할당 기법을 <표 2>의 내장형 시스템 상에서 구현하였다[17]. 본 논문의 주요 대상이 되는 NAND 플래시 메모리는 (그림 6)을 분석할 때 사용했던 특성을 갖는 삼성 K9F1208U0B NAND 플래시 메모리이다[12]. 한편 <표 2>의 하드웨어 상에 리눅스 커널이 동작하며, 본 연구에서는 커널 버전 2.4.19를 사용하였다. 그리고 YAFFS (Yet Another Flash File System)를 NAND 플래시 메모리를 관리하는 소프트웨어 컴포넌트로 사용하였다[18]. YAFFS는 리눅스 커널과 VFS (Virtual File System) 및 MTD (Memory Technology Device) 층을 통해 연결된다. YAFFS는 VFS를 통해 *open(), read(), write()* 등의 사용자 인터페이스를 제공하며, MTD의 *readchunkfromnand(), writechunktonand(), eraseblockinnand()* 등의 인터페이스를 이용해 플래시 메모리를 제어한다[19].

YAFFS의 디폴트 페이지 할당 기법은 순차 할당이다. 본 논문에서는 (그림 8)과 (그림 9)에 설명된 변경 특성을 고려한 새로운 페이지 할당 기법을 YAFFS에 구현하였으며, 이를 위해 yaffs_guts.c 파일의 yaffs_WriteChunkDataToObject(), yaffs_UpdateObjectHeader(), yaffs_FlushFile() 그리고 yaffs_fs.c 파일의 init_yaffs_fs(), exit_yaffs_fs() 함수들이 수정되었다.

한편, YAFFS의 디폴트 블록 클리닝 기법은 다음과 같다. 일단 블록 클리닝은 매 쓰기 요청을 서비스할 때마다 시도된다. 그리고 블록 클리닝 방법은 normal mode와 aggressive mode라는 두 가지 모드로 구분되어 수행된다. Normal mode에서는 우선 임의 개수의 블록들 중에서 (YAFFS의 디폴트 설정은 200개) 무효 상태의 페이지가 가장 많은 블록을 하나 선택한다. 그리고 이 블록에서 유효 상태의 페이지 개수가 3보다 적으면 이 블록을 클리닝한다. 반면 aggressive mode에서는 플래시 메모리 전체 블록 중에서 무효 상태인 페이지가 존재하는 블록을 하나 선택하고, 이 블록을 무조건 클리닝 시킨다. 한편, 클린 상태의 블록이 미리 지정된 개수 (YAFFS의 디폴트 설정은 6개) 이하이면 aggressive mode가 되고, 아니면 normal mode가 된다. 결국 클린 블록이 충분하면 보수적인 방법으로 클리닝을 시도하고, 부족하면 적극적인 방법으로 클리닝을 시도하는 것이다. 본 연구진은 성능 평가를 목적으로 사용자 수준에서 블록 클리닝을 호출할 수 있는 인터페이스를 구현하였다. 이 인터페이스는 블록 클리닝 시작 시점(지금/몇 분 뒤에), 주기 여부(한번/주기적), 클리닝 양(모든 무효 상태 페이지 클리닝/일정 비율이 될 때까지 클리닝) 등을 인자로 제어할 수 있다. 본 논문에서 새로 구현한 YAFFS 소스는 [20]에서 다운받을 수 있다.

<표 2> 내장형 시스템 하드웨어 명세

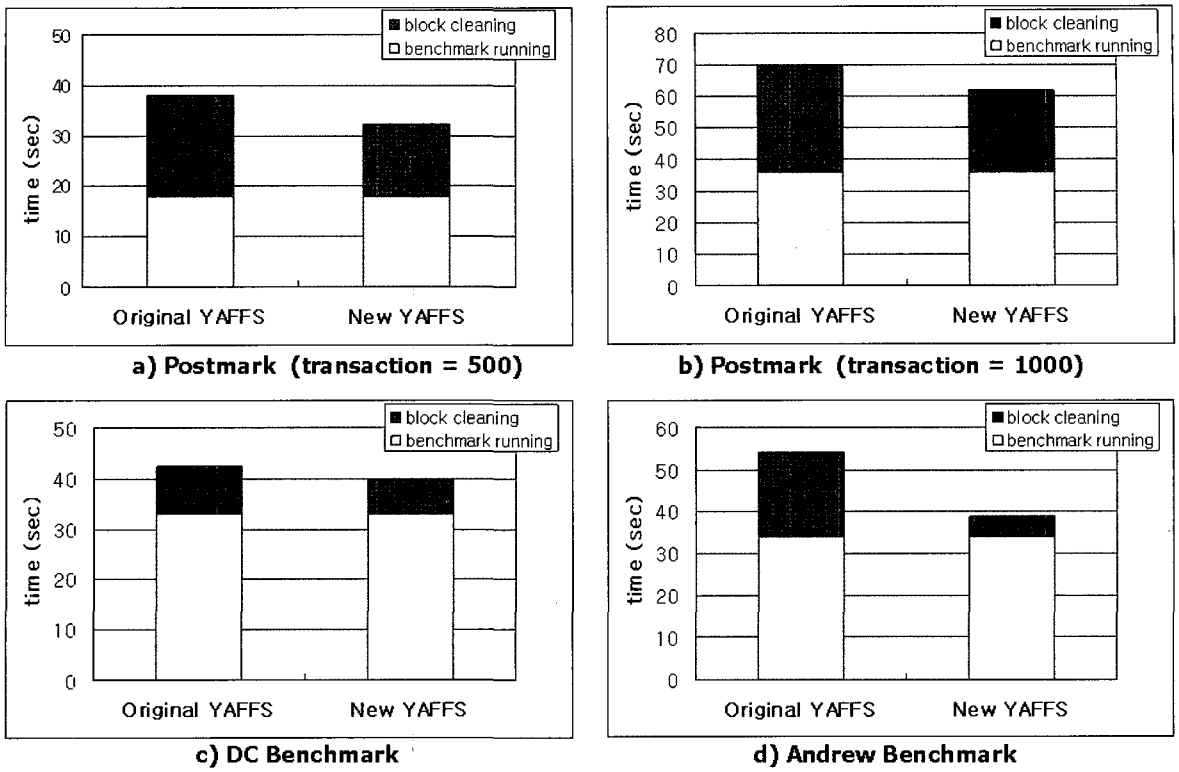
Hardware Component	Specification
CPU	400MHz XScale PXA255
RAM	64MB SDRAM
Flash memory	64MB NAND, 512KB NOR flash memory
Interface	RS232, USB, CS8900, JTAG
Others	LCD, Touch, LED

6. 성능 평가

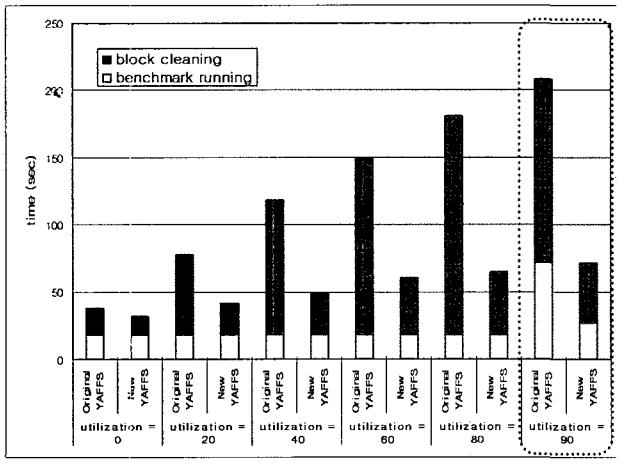
본 절에서는 기존 YAFFS의 페이지 할당 기법과 본 논문에서 제안된 새로운 페이지 할당 기법을 비교한다. 성능 비교를 위해 3가지 벤치마크를 사용하였으며, 각 벤치마크의 특성은 다음과 같다.

- Postmark: Postmark는 작은 크기의 파일들을 생성하고, 읽기, 변경, 삭제 등의 트랜잭션을 수행한다[21]. 파일의 개수와 수행할 트랜잭션 개수는 사용자에 의해 설정될 수 있다(기본 설정은 500 트랜잭션).
- DC(Digital Camera) Benchmark: 이 벤치마크는 본 연구진이 디지털 카메라의 동작을 에뮬레이션할 수 있도록 만든 인위적(synthetic) 워크로드이다. 각 트랜잭션은 일반 JPEG 사진 크기 (200~1000KB)의 파일들을 여러 개 생성하고 이 중에 임의적으로 몇 개를 읽고 삭제하는 작업으로 구성되며, 이러한 트랜잭션들을 정해진 횟수만큼 반복한다.
- Andrew Benchmark: 이 벤치마크는 디렉터리 생성, 파일 복사, 파일 상태 검사, 파일 내용 검사, 컴파일이라는 5단계로 구성된다[22]. 본 논문에서는 디렉터리 생성, 파일 복사라는 2단계만 수행하였으며, 벤치마크의 부하를 증가시키기 위해 생성 및 복사 횟수를 디폴트 설정보다 크게 하였다.

(그림 13)은 기존 YAFFS의 순차적인 페이지 할당 기법과 본 논문에서 제안한 변경 특성을 고려한 페이지 할당 기법의 성능 차이를 보여준다. 기존 YAFFS(original YAFFS)와 새로운 YAFFS(new YAFFS)간에는 페이지 할당 정책만 다를 뿐 다른 모든 조건은 동일하다. 그림에서 벤치마크 실행 시간(benchmark running time)은 각 벤치마크가 수행 시작부터 완료까지 걸린 시간으로, 모든 벤치마크는 이용율이 0% 상태에서 수행되었다. 블록 클리닝 시간(block cleaning time)은 벤치마크 수행 완료 후 모든 무효 상태의 페이지들을 클리닝하는데 걸리는 시간으로, 새로 추가한 블록 클리닝 인터페이스를 호출하여 측정하였다. 실험에서 데이터 분류를 위한 제어변수들인 Δt, h , c 는 각각 10, 2, 0으로 설정되어 있다.



(그림 13) 페이지 할당 기법과 블록 클리닝 비용



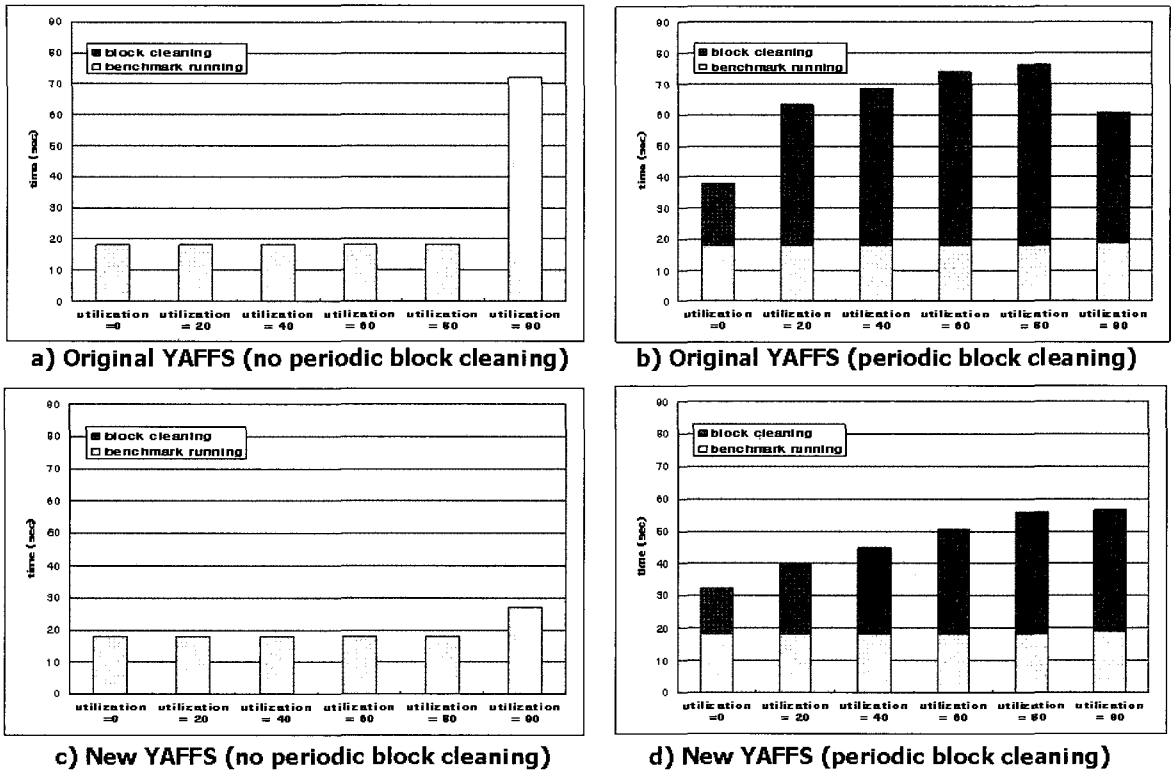
(그림 14) 이용율의 변화에 따른 성능 변화

(그림 13)에서 우리는 다음의 것들을 파악할 수 있다. 첫째, 기존 YAFFS와 새로운 YAFFS 상에서 벤치마크 수행 시간이 유사하다. 이를 통해 본 논문에서 제안된 페이지 할당 기법의 추가적인 부하가 거의 없음을 알 수 있다. 둘째, 새로운 YAFFS 상에서 블록 클리닝 비용이 기존 YAFFS에서 보다 적다. 구체적으로 Postmark(트랜잭션 500개), Postmark (트랜잭션 1000개), DC Benchmark, Andrew Benchmark에서 각각 5.6, 8.0, 2.6, 15.4초 향상되었으며, 평균적으로 7.8초 향상되었다. 이는 제안된 페이지 할당 기법이 실제로 순수도를 향상시키며 결국 블록 클리닝 부하를 감소시킴을 의미한다.

(그림 14)는 Postmark(트랜잭션 500개)를 서로 다른 이용율에서 수행시켰을 때 성능 결과이다. 그림에서 우리는 다음 2가지를 발견할 수 있다. 첫째, 이용율이 증가함에 따라

〈표 3〉 페이지 할당 정책과 순수도

Utilization	Scheme	Block Status				block cleaning		Classification	
		valid_only	invalid_only	both	Purity	erase	copy	hot	cold
0%	Orignal YAFFS	4	0	357	0.90	357	10147	N/A	N/A
	Modified YAFFS	116	0	246	0.93	246	6608	137	280
20%	Orignal YAFFS	97	0	1045	0.72	1045	30203	N/A	N/A
	Modified YAFFS	775	0	369	0.90	369	8527	291	1141
40%	Orignal YAFFS	192	0	1732	0.53	1732	50227	N/A	N/A
	Modified YAFFS	1430	0	496	0.86	496	10650	373	1633
60%	Orignal YAFFS	282	0	2423	0.35	2423	70397	N/A	N/A
	Modified YAFFS	2090	0	617	0.83	617	12552	460	2241
80%	Orignal YAFFS	376	0	3111	0.16	3111	90435	N/A	N/A
	Modified YAFFS	2750	0	738	0.80	738	14463	545	2973



(그림 15) 주기적인 블록 클리닝의 기대 효과

블록 클리닝의 비용이 증가한다. 둘째, 이용율이 증가함에 따라 새로운 YAFFS와 기존 YAFFS에서 블록 클리닝의 성능 차가 점점 커진다. 이는 이용율이 증가함에도 제안된 페이지 할당 기법이 순수도를 좋은 상태로 유지하여 블록 클리닝 비용의 급격한 증가를 효과적으로 방지하고 있음을 보여준다. 이것은 <표 3>에서 더욱 명확히 알 수 있다.

<표 3>은 (그림 14)의 실험을 수행하면서 각 이용율에서 벤치마크를 수행 시킨 후 플래시 메모리의 블록 상태, 모든 무효 상태의 페이지들을 재생(reclaim)하도록 블록 클리닝을 할 때 발생한 블록 삭제와 페이지 복사 횟수, 그리고 블록 클리닝 이후 분류된 자주 변경되는 데이터(hot-modified data)와 자주 변경되지 않는 데이터(cold-modified data) 개수 정보를 요약한 것이다. 표에서 valid_only, invalid_only, both는 각각 유효 상태의 페이지들만 존재하는 블록, 무효 상태의 페이지들만 존재하는 블록, 유효/무효 상태의 페이지가 동시에 존재하는 블록 개수이다. 현재 실험 환경에서 플래시 메모리의 가용 블록 개수는 3710개이며, 결국 순수도는 "1 - both/3710"의 식으로 계산된다.

<표 3>에서 우리는 다음의 3가지를 발견할 수 있다. 첫째, 제안된 페이지 할당 기법이 순수도를 향상시키며, 그 결과 블록 클리닝할 때 블록 삭제 개수와 페이지 복사 개수를 줄여 블록 클리닝 시간을 단축시킨다. 둘째, 각 기법에서 invalid_only 즉 무효 상태인 페이지들만이 존재하는 블록이 없다. 이는 실제로 그런 블록들이 드물기 때문이며, 또한 YAFFS의 디폴트 블록 클리닝 기법 때문이다. 셋째, 제안된 기법은 자주 변경되는 데이터(hot-modified data)와 자주 변경되지 않는 데이터(cold-modified data)를 100여개 이상 발견하여 구분하며, 이 때문에 순수도를 향상시킬 수 있었다. 상세 실험 결과 h가 커질수록 자주 변경되는 데이터로 발견되는 횟수가 감소하고, c 가 커질수록 자주 변경되지 않는 데이터로 발견되는 횟수가 증가하였다. 한편, Δt가 커질수록 자주 변경되는 데이터로 발견되는 횟수가 증가하고 자주 변경되지 않는 데이터로 발견되는 횟수가 감소하는 것을 측정할 수 있었다. 사실, 자주 변경되는 데이터와 그렇지 않은 데이터를 효과적으로 구분하는 것 자체만으로 큰 연구 주제이며, 본 논문의 주요 내용인 플래시 메모리 파일 시스템에서 순수도와 성능 특성 분석과는 독립적인 연구 주제라 본 연구에서 자세한 분석을 수행하지는 않았다.

한 가지 흥미로운 것은 이용율이 거의 100%가 되도록 벤치마크를 수행 시켰을 때 성능결과인데 이는 (그림 14)에서 점선 사각형으로 둘러싸여 있는 부분이다. 이 부분은 이용율 90%의 상태에서 Postmark(트랜잭션 500개)를 수행한 결과이며, 이때에는 벤치마크 수행 중에 클린 블록이 부족하게 되어 YAFFS의 디폴트 블록 클리닝이 aggressive mode로 동작한다. 따라서 응용 수행 중에 클리닝이 수행되는 요구 블록 클리닝(on-demand block cleaning)이 많이 발생한다. 그 결과 벤치마크의 수행 시간이 크게 증가한다. 구체적으로 기존 YAFFS의 경우 80%의 이용율에서는 수행 시간이 18초인데, 90%의 이용율에서는 수행 시간이 72초로 증가하였다. 반면 새로운 YAFFS의 경우 수행 시간이 27초로 증가하였다. 요구 블록 클리닝 횟수를 측정해본 결과 기존 YAFFS에서는 1257회, 새로운 YAFFS에서는 349회 발생한

것으로 파악되었다. 결국 이 결과는 제안된 블록 클리닝 기법이 순수도를 증가시켜 블록 클리닝 비용 자체를 감소시킬 수 있을 뿐만 아니라 요구 블록 클리닝 횟수를 감소시켜 응용의 수행 시간도 단축시킬 수 있음을 보여준다.

다음으로 주기적인 블록 클리닝이 성능에 어떤 영향을 주는지 실험해 보았다. (그림 15)의 a)는 기존 YAFFS, 즉 주기적인 블록 클리닝이 없는 경우 실험 결과이다. 클린 블록이 충분하면(위 실험에서 이용율 0~80%의 경우) Postmark(트랜잭션 500개)의 수행시간이 18초이지만, 클린 블록이 부족하면 (이 실험에서 이용율 90%의 경우) 수행시간이 72초로 증가함을 알 수 있다. (그림 15)의 b)는 기존 YAFFS에서 주기적으로 블록 클리닝을 수행할 경우 실험 결과이다. 구체적으로 이용율 0, 20, 40, 60, 80, 90%에서 Postmark를 수행하고 그 이후에 블록 클리닝을 수행하였다. 이 경우 Postmark의 수행 시간은 모든 이용율에서 18초이다. 결국 (그림 15)의 a)에 비해 이용율 90%일 경우 수행시간의 단축 효과를 얻는다. 하지만 (그림 15)의 b)는 주기적인 블록 클리닝을 필요로 한다.

(그림 15)의 c)와 d)는 제안된 페이지 할당 기법에서 주기적인 블록 클리닝을 사용하지 않을 경우와 사용할 경우 실험 결과이다. (그림 15)의 a), b)와 마찬가지로 주기적인 블록 클리닝을 사용할 경우 90% 이용율에서 벤치마크의 응답 시간을 단축할 수 있다는 장점이 있지만, 주기적 블록 클리닝 중에 사용자의 요청이 발생할 경우 그 요청의 응답 시간이 길어질 수 있다는 tradeoff가 존재함을 알 수 있다. 또한 새로운 YAFFS는 주기적인 블록 클리닝 사용 여부에 관계없이 항상 기존 YAFFS에 비해 좋은 성능을 제공함을 알 수 있다.

7. 관련 연구

로그 구조 파일 시스템은 디스크를 추가 전용(append only) 저장 장치로 간주한다. 따라서 수정 요청을 덮어 쓰기가 아닌 새로운 위치에 쓰기와 기존 데이터 무효화로 서비스하며, 무효화된 데이터는 세그먼트 클리닝을 통해 재생한다[2]. 세그먼트 클리닝 성능이 로그 구조 파일 시스템의 성능에 중요한 영향을 끼침에 따라 이를 개선하려는 많은 기법이 제안되었다[3, 4, 5, 6]. 참고문헌 [3]에서는 요구 클리닝과 백그라운드 클리닝을 구분하고, 휴리스틱을 이용한 유휴 시간 발견 및 백그라운드 클리닝을 통해 사용자가 느끼는 클리닝 비용의 감소를 시도하였다. 참고문헌 [4]에서는 디스크의 이용율에 따라 클리닝과 홀 플러깅(hole plugging)을 적응력있게 적용하는 방법과 캐시를 이용한 클리닝 성능 향상 기법을 제안하였다. 참고문헌 [5]와 [6]은 본 연구와 유사하게 변경 횟수를 고려하여 클리닝의 성능 향상을 시도한 연구들이다. 참고문헌 [5]에서는 데이터의 변경 빈도가 높은 데이터들과 그렇지 않은 데이터를 모아 각각 서로 다른 세그먼트에 한 번에 기록함으로써 클리닝 비용을 줄였고, 참고문헌 [6]에서는 변경 빈도가 높은 데이터는 non in-place update를 수행하고 그렇지 않은 데이터는 in-place update를 수행하여 클리닝 비용을 줄였다. 하지만 참고문헌 [5]와 참고문헌 [6]은 디스크를 대상으로 하기 때문에 플래시 메모리를 대상으로 하는 본 연구와는 다르다. 우선 참고문헌 [6]은 덮어 쓰기 제한이 있는 플래시 메모리에서는 적용할 수 없다. 그리고 참고문헌 [5]는 디스크의 탐색 거리를 고려하기 위해 2개의 세그먼트를 선택하여 한 곳에는 변경 빈도가 낮은 데이터를 다른 곳에는 변경 빈도가 높은 데이터를 저장하였으며, 저장할 때에는 세그먼트 크기만큼 데이터를 모아 한 번에 기록하였다. 하지만 플래시 메모리의 경우에는 탐색 거리를 고려할 필요가 없어 여러 블록들을 동시에 사용할 수 있으며, 특정 블록에 기록할 때에도 블록 크기만큼 데이터를 모아 한 번에 기록하는 것이 아닌 일부분만 점차적으로 기록할 수 있다.

플래시 메모리에서 효과적인 블록 클리닝에 대한 연구도 많이 수행되었다[7-9, 23]. 참고문헌 [23]에서는 새로 쓰이는 데이터와 클리닝에 의해 복사되는 데이터를 서로 다른 블록에 쓰는 기법(separate segment for cleaning)을 제안하였다. 참고문헌 [8]에서는 쓰기 분포가 균등할 때 FIFO 알고리즘과 치우쳐 있을 때 locality gathering 알고리즘을 함께 사용하는 혼성 클리닝(hybrid cleaning) 기법을 제안하였다. 이때 locality gathering은 클리닝할 때 유효한 데이터를 높은 번호의 블록으로 이동시키고, 새로 쓰여진 데이터를 낮은 번호의 블록으로 이동시킴으로써 궁극적으로 자주 변경되는 데이터와 그렇지 않는 데이터를 다른 블록으로 모으는 방법이다. 이 두 연구는 블록 클리닝 시점에서만 데이터를 분류하고 관리한다는 측면과 실제 분류하는 방법에서 본 연구와 차이가 있다. 참고문헌 [7, 24]에서는 CAT(Cost Age Time)과 DAC(Dynamic dAta Clustering) 기법을 제안하였다. CAT는 블록 클리닝 비용, 데이터가 쓰여진 이후 지난 시간, 마지막으로 삭제를 수행한 시간을 모두 고려하여 클리닝할 블록을 선택한다. 한편, DAC는 CAT를 개선한 것으로, 플래시 메모리를 몇 개의 구역(region)으로 구분하고 변경 빈도에 따라 데이터를 특정 구역에 할당한다. 결과적으로 이 방법은 데이터를 쓸 때 변경 빈도에 따라 서로 다른 블록에 할당하는 것으로 본 논문의 접근 방법과 유사하다. 하지만 본 연구는 실제 플래시 메모리 상에서 구현하고 성능을 평가하였으며, 순수도(purity)라는 측정 가능한 성능 인자를 제안하고 이를 기반으로 블록 클리닝 비용을 예측하였다는 점에서 참고문헌 [7, 24] 연구와 차이가 있다. 한편, 참고문헌 [9]에서는 실시간 환경에서 시간 제약 조건을 만족하는 블록 클리닝 기법을 제안하였다.

8. 결 론

본 논문에서는 플래시 메모리의 특징을 기술하고, 이 특징들로부터 블록 클리닝 기법이 필요한 이유를 설명하였다. 그리고 블록 클리닝의 성능에 영향을 주는 성능 인자 3가지, 구체적으로 이용율, 무효율, 순수도를 정의하였으며, 순수도

를 향상시키는 새로운 페이지 할당 정책을 제안하였다. 제안된 기법은 YAFFS 상에 구현되었으며 내장형 보드를 이용한 실험 결과 기존 YAFFS보다 블록 클리닝 시간을 최대 15.4초(평균 7.8초) 단축 시켰으며, 이용율이 100%가 되도록 벤치마크를 수행시키면 기존 YAFFS보다 벤치마크 수행 시간이 최대 45초 단축되어 플래시 메모리 파일시스템의 성능을 향상시킴을 알 수 있었다.

향후 본 연구는 다음 3가지 방향으로 확장될 것이다. 첫 번째 방향은 제안된 모델을 요청의 집중도(burstness)를 반영하여 확장하는 것이다. 두 번째 방향은 순수도를 향상시키는 새로운 블록 클리닝 기법의 연구이다. 성능 평가 절에서도 언급했듯이 순수도는 페이지 할당 기법 뿐만 아니라 블록 클리닝 기법과도 밀접히 연관되어 있으며, 이 두 기법의 유기적인 결합은 더욱 큰 성능 향상을 얻을 수 있다. 세 번째 방향은 마모도 평준화 고려이다. 본 논문에서 제안된 기법은 변경 빈도에 따라 데이터를 분류하였으며, 이 정보는 블록의 마모도 평준화에 효과적으로 이용될 수 있다.

참 고 문 헌

[1] William Stalling, "Operating Systems: Internals and Design Principles", 5th Edition, Pearson Prentice Hall, 2004.

[2] M. Rosenblum and J. K. Ousterhout, "The design and implementation of a log-structured file system", ACM Transactions on Computer Systems, Vol.10, No.1, pp.26-52, 1992.

[3] T. Blackwell, J. Harris, and M. Selzer, "Heuristic cleaning algorithms in log-structured file systems", Proceedings of the 1995 Annual Technical Conference, pp.277-288, 1993.

[4] J. Matthews, D. Roselli, A. Costello, R. Wang, and T. anderson, "Improving the performance of log-structured file system with adaptive methods", ACM Symposiums on Operating System Principles (SOSP), pp.238-251, 1997.

[5] J. Wang and Y. Hu, "WOLF - a novel reordering write buffer to boost the performance of log-structured file system", Proceedings of the USENIX Conference on File and Storage Technologies (FAST), pp.46-60, 2002.

[6] W. Wang, Y. Zhao, and R. Bunt, "HyLog: A High Performance Approach to Managing Disk Layout", Proceedings of the USENIX Conference on File and Storage Technologies (FAST), pp.145-158, 2004.

[7] 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.

[8] M. Wu and W. Zwaenepoel, "eNVy: a non-volatile, main memory storage system", Proceeding of the 6th international conference on Architectural Support for Programming languages and operation systems, pp.86-97, 1994.

[9] L. P. Chang, T. W. Kuo and S. W. Lo, "Real-time garbage collection for flash memory storage systems of real time embedded systems", ACM Transactions on Embedded Computing Systems, Vol.3, No.4, pp.837-863, 2004.

[10] E. Gal and S. Toledo, "Algorithms and Data Structures for Flash Memories", ACM Computing Surveys, Vol.37, No.2, pp 138-163, 2005.

[11] Micron, "Technical note: small block vs. large block NAND device", http://download.micron.com/pdf/technotes/nand/tn2907.pdf

[12] Samsung Inc., http://www.samsung.com/Products/Semiconductor/NANDFlash

[13] Ashok K. Sharma, "Advanced Semiconductor Memories: Architectures, Designs, and Applications", WILEY Interscience, 2003.

[14] 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. 48, No.2, pp.366-375, 2002.

[15] C. Park, J. Seo, D. Seo, S. Kim and B. Kim, "Cost-Efficient Memory Architecture Design of NAND Flash Memory Embedded System", IEEE International Conference on Computer Design, 2003.

[16] 배영현, 최종무, 이동희, 노삼혁, 민상렬, "플래시 메모리 파일 시스템을 위한 효율적인 소거 평준화 기법", 한국정보과학회 가을 학술발표논문집, pp.580-582, 2004.

[17] EZ-X5, FALINUX Inc., http://www.falinux.com/zproducts/ez-x5.php

[18] Aleph One, "YAFFS: Yet another flash file system", http://www.aleph1.co.uk/yaffs/

[19] MTD subsystem for Linux, http://www.linux-mtd.infradead.org/archive/index.html

[20] YAFFS_purity, http://embedded.dankook.ac.kr/~choijm/yaffs_purity.tar.gz

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

[22] J. H. Howard, M. L. Kazar, S. G. Menees, D. A. Nichols, M. Satyanarayanan, R. N. Sidebotham, M. J. West, "Scale and Performance in a distributed file system", ACM Transactions on Computer Systems, Vol.6, No.1, Feb., 1988.

[23] Kawaguchi, S. Nishioka and H. Motoda, "A flash-memory based file system", Proceedings of the 1995 USENIX Annual Technical Conference, pp.155-164, 1995.

[24] 정하용, 김진수, 한환수, 최기선, "JFFS2를 위한 효율적인 Garbage Collector의 설계 및 구현", 한국정보과학회 가을 학술발표논문집, pp.523-525, 2004.

백 승 재

e-mail : ibanez1383@dankook.ac.kr

2005년 단국대학교 컴퓨터공학과(공학사)

2004년~현재 비트 컴퓨터 강사

2005년~현재 단국대학교 정보컴퓨터과학과 석사과정

관심분야 : 운영체제, 내장형 시스템

최 종 무

e-mail : choijm@dankook.ac.kr

1993년 서울대학교 해양학과(이학사)

1995년 서울대학교 컴퓨터공학과 (공학석사)

2001년 서울대학교 컴퓨터공학과 (공학박사)

2001년~2003년 유비쿼스 주식회사 책임연구원

2003년~현재 단국대학교 정보컴퓨터학부 컴퓨터과학 전공 조교수

2005년~2006년 UC Santa Cruz 방문교수

관심분야 : 운영체제, 내장형 시스템, 고성능 저장장치 등