# 플래시 메모리 기반의 가상 메모리 시스템을 위한 중복성을 고려한 GC 기법

## (Duplication-Aware Garbage Collection for Flash Memory-Based Virtual Memory Systems)

지 승 구 † 신 동 군 ††
(Seunggu Ji) (Dongkun Shin)

**요 약** 임베디드 시스템이 모놀리식(monolithic) 커널을 사용하면서, NAND 플래시 메모리는 가상 메모리 시스템의 스왑(swap) 공간을 위해 사용되고 있다. 플래시 메모리는 저전력 소비, 충격 내구성, 비 휘발성의 장점을 가지지만, '쓰기 전 삭제'의 특징 때문에 가비지 컬렉션(GC) 작업이 필요하다. GC 기법의 효율성은 플래시 메모리 성능에 큰 영향을 미친다. 본 논문에서는 플래시 메모리를 기반으로 하는 가상 메모리 시스템에서 메인 메모리와 플래시 메모리 사이에 중복된 데이터를 활용한 새로운 GC 기법을 제안한다. 제안된 기법은 GC 부하를 최소화하기 위해 데이터의 지역성을 고려한다. 실험 결과는 제안된 GC 기법이 이전의 기법과 비교하여 평균적으로 37%의 성능을 향상시킴을 보여준다.

**키워드** : NAND 플래시 메모리, 플래시 변환 계층, 가비지 컬렉션, 가상 메모리, 버퍼 관리

***Abstract*** As embedded systems adopt monolithic kernels, NAND flash memory is used for swap space of virtual memory systems. While flash memory has the advantages of low-power consumption, shock-resistance and non-volatility, it requires garbage collections due to its erase-before-write characteristic. The efficiency of garbage collection scheme largely affects the performance of flash memory. This paper proposes a novel garbage collection technique which exploits data redundancy between the main memory and flash memory in flash memory-based virtual memory systems. The proposed scheme takes the locality of data into consideration to minimize the garbage collection overhead. Experimental results demonstrate that the proposed garbage collection scheme improves performance by 37% on average compared to previous schemes.

**Key words** : NAND flash memory, Flash Translation Layer (FTL), Garbage Collection, Virtual Memory, Buffer Management

## 1. 서 론

NAND 플래시 메모리는 휴대전화, 디지털 카메라, MP3 플레이어 같은 임베디드 시스템의 저장 장치로 널리 사용되고 있다. NAND플래시 메모리는 저전력 소비, 높은 임의 접근 성능, 충격 내구성의 장점을 가진다. 플래시 메모리의 가격이 하락하면서 최근에는 데스크톱 PC와 엔터프라이즈 서버에서 여러 개의 플래시 칩으로 구성된 SSD(Solid State Disk)가 HDD(Hard Disk Drive)를 대체하고 있다.

플래시 메모리는 낮은 접근 비용 때문에 파일과 코드를 저장할 뿐만 아니라 가상 메모리 시스템의 스왑 공간을 위해서도 좋은 장치이다[1-3]. 하드 디스크와 비교해 볼 때, 플래시 메모리는 페이지 스와핑 비용을 줄일 수 있다. 임베디드 시스템이 임베디드 리눅스와 Windows CE 같은 모놀리식 커널을 이용하면서 플래시 메모리 기반의 가상 메모리 시스템은 많이 사용될 것이다. 그러나 플래시 메모리에 관한 대부분의 기존 연구는 플래시 파일 시스템에 집중되어 있으며 플래시 메모리 기

† 학생회원 : 성균관대학교 정보통신공학부
skyjsg@skku.edu
†† 정 회 원 : 성균관대학교 정보통신공학부 교수
dongkun@skku.edu
논문접수 : 2010년 3월 3일
심사완료 : 2010년 4월 5일


정보과학회논문지: 시스템 및 이론 제37권 제3호(2010.6)

반의 가상 메모리 시스템에 관한 연구는 부족한 실정이다.

플래시 메모리는 하드 디스크와 다른 몇 가지 특징을 가지고 있다. 하나의 플래시 메모리 칩은 여러 개의 블록들로 구성되어 있으며 하나의 블록은 여러 개의 페이지로 구성된다. 예를 들어, MLC NAND 플래시 메모리에서 하나의 블록은 128개의 페이지로 구성되어 있고, 하나의 페이지 크기는 4KB이다. 플래시 메모리는 읽기, 쓰기, 삭제의 세 가지의 명령을 지원한다. 읽기와 쓰기의 명령은 페이지 단위로 실행되며, 삭제 명령의 단위는 블록이다. 플래시 메모리는 기존에 데이터가 기록된 페이지에 새로운 데이터를 덮어쓸 수가 없다. 데이터를 페이지에 쓰기 전에 해당 블록은 삭제되어야 한다. 이러한 특징을 '쓰기 전 삭제' 제약이라고 부른다. 그러므로, 대부분의 플래시 저장 시스템은 클린 페이지에 새로운 데이터를 기록하고 이전의 페이지는 무효화 시킨다. 이러한 동작은 OS에서 사용되는 논리적 주소를 플래시 메모리에서 사용되는 물리적 주소로 변경하는 매핑 기법을 필요로 한다.

이러한 특징을 처리하기 위해, 플래시 변환 계층(Flash Translation Layer, 이하 FTL)이라 불리는 소프트웨어 계층이 파일 시스템과 플래시 메모리 사이에 사용된다[4,5]. FTL은 주된 두 가지의 기능을 가지는데, 첫 번째는 매핑 단위에 따라 블록 레벨, 페이지 레벨, 하이브리드 매핑으로 나뉘는 주소 매핑 기능이다. 두 번째는 업데이트 동작에 의해 무효화된 플래시 페이지를 삭제하여 다시 쓰기가 가능하도록 만드는 가비지 컬렉션(GC: garbage collection) 동작이다. GC는 희생 블록 선택, 유효 페이지 복사, 희생 블록 삭제의 세가지 단계로 이루어진다. 희생 블록 선택은 삭제할 블록을 찾는 것으로, 일반적으로 GC비용을 가장 적게 유발시키는 블록을 선택한다. 다음으로 유효 페이지 복사는 희생 블록의 유효 페이지를 다른 클린 블록으로 복사한다. 마지막 단계에서는 희생 블록을 삭제하여 이후에 쓰기가 가능하도록 만든다.

GC는 많은 수의 페이지 복사와 블록 삭제를 필요로 하기 때문에 상당한 부하가 발생한다. 따라서, 효과적인 GC 기법은 플래시 메모리 저장 시스템의 높은 성능을 위해 매우 중요하다. 그 동안 플래시 메모리를 위한 GC에 관한 많은 연구들이 있었지만, 플래시 메모리 기반의 가상 메모리 시스템을 위한 GC에 초점을 둔 연구는 많지 않다.

플래시 메모리가 스왑 공간을 위해 사용될 때, 메인 메모리와 플래시 메모리 사이에서 중복된 데이터를 활용한다면 GC 중에 불필요한 유효 페이지 복사를 제거할 수 있다. 가상 메모리 페이지가 스왑-인(swap-in)될 때, 이 페이지는 메인 메모리와 플래시 메모리에 동시에 존재하며, 해당 페이지가 스왑-아웃(swap-out) 될 때 플래시 메모리에 다시 쓰여지게 될 것이다. 따라서, 만약 GC 중에 이러한 중복된 페이지를 다른 블록으로 복사하지 않는다면 페이지 복사를 줄일 수 있다. 기존의 DA-GC (Duplication-Aware Garbage Collection)[6]은 이러한 기법을 사용하고 있다. DA-GC는 FTL이 페이지 레벨 주소 매핑을 사용하는 것을 가정하여 고안되었으며 페이지 레벨 매핑에서는 좋은 성능을 보여주고 있다. 그러나, 하이브리드 매핑에서는 GC의 성능을 향상 시키지 못하는 문제를 가지고 있다.

본 논문에서는 플래시 메모리 기반의 가상 메모리 시스템에서 지역성과 중복 데이터를 고려한 희생 블록 선택 기법(LDA-VBS)과 지역성과 중복 데이터를 고려한 블록 병합 기법(LDA-BM)을 제안한다. 이 기법들은 하이브리드 매핑 FTL에서 중복된 데이터의 업데이트 가능성을 고려하여 DA-GC의 문제점을 해결한다. 트레이스 기반의 시뮬레이터를 사용한 실험 결과, 가상 메모리 벤치마크에서 중복성을 고려하지 않은 DU-GC(Duplication-unaware Garbage Collection)보다 전반적인 플래시 I/O성능을 평균적으로 37% 향상할 수 있다는 것을 확인했다.

본 논문의 구성은 다음과 같다. 2장에서는 플래시 메모리 관리 기법에 관한 관련 연구를 살펴본다. 3장에서는 본 논문의 동기를 소개하며, 4장에서는 LDA-VBS와 LDA-BM 기법을 자세히 기술한다. 5장에서는 성능 평가 결과를 보여주며, 6장은 본 논문을 결론짓는다.

## 2. 관련연구

대부분의 플래시 메모리에 관한 기존 연구는 주소 매핑 기법에 초점을 맞추고 있다. 블록 레벨 매핑[7]은 논리 블록 주소와 물리 블록 주소 사이에서 변환 정보를 유지한다. 그러므로, 블록 내의 페이지의 오프셋은 논리 블록과 물리 블록에서 동일하며, 작은 매핑 테이블이 필요하다. 그러나, 데이터 업데이트 요청을 위해 블록내의 업데이트 되지 않은 페이지들이 클린 블록으로 복사되기 때문에 큰 부하를 유발시킨다.

페이지 레벨 매핑[8]에서는 논리 페이지 주소가 물리 페이지 주소에 매핑된다. 페이지 단위로 독립적인 관리가 이루어지기 때문에, 페이지 레벨 매핑은 블록 레벨 매핑 보다 더욱 효과적이다. 그러나, 매핑 테이블을 위해 큰 메모리 공간을 요구한다는 단점이 있다.

하이브리드 매핑[9-11]은 페이지 레벨 매핑과 블록 레벨 매핑을 둘 다 사용한다. 플래시 블록의 일부를 로그 버퍼로써 사용하기 때문에 로그 버퍼 기반 FTL이라고 불린다. 데이터 블록은 블록 레벨 매핑을 사용하고

로그 블록은 페이지 레벨 매핑을 사용한다. 모든 쓰기 요청은 먼저 로그 버퍼로 보내지며, 만약 로그 버퍼에 더 이상 공간이 없으면, 빈 공간을 만들기 위해 로그 블록의 유효한 페이지를 데이터 블록으로 복사하고 해당 블록을 삭제하게 된다. 하이브리드 매핑 기법은 작은 매핑 테이블의 크기로 높은 성능을 제공하므로 대부분의 시스템이 하이브리드 매핑 기법을 사용하고 있다.

로그 버퍼 기반 FTL에 대한 다양한 연구들이 있는데, BAST(Block-Associative Sector Translation) 기법[9]에서는 하나의 로그 블록이 하나의 데이터 블록에 연관되어 있다. 데이터 블록에 있는 페이지가 업데이트가 될 때 새로운 데이터는 연관된 로그 블록에만 쓰여질 수 있다. 데이터 블록에 할당 가능한 로그 블록이 없을 때는 GC가 발생하는 데, 임의 쓰기가 많을 때는 이러한 GC가 자주 발생하게 된다. GC는 로그 블록 중에 하나를 선택하고 선택된 로그 블록과 연관된 데이터 블록의 모든 유효한 페이지를 빈 블록으로 복사한다. 그리고 나서, 해당 로그 블록과 데이터 블록은 삭제되고 새로운 로그 블록으로 사용된다.

BAST 기법의 단점은 임의 쓰기 패턴에서 자주 GC가 발생하는 것이다. 그러한 문제를 해결하기 위해 FAST(Fully-Associative Sector Translation)기법[10]이 제안되었다. FAST 기법에서는 하나의 로그 블록이 여러 개의 데이터 블록과 연관될 수 있다. 그러므로 임의 쓰기 요청에 의한 빈번한 GC를 막을 수 있다. 그러나 FAST 기법은 GC가 발생할 때 희생 로그 블록에 연관된 데이터 블록들이 여러 개이기 때문에 많은 유효 페이지를 복사해야 하므로 큰 GC 비용이 발생한다.

일반적으로 플래시 메모리 저장 시스템은 쓰기 요청의 처리 시간을 줄이기 위해서 버퍼 캐시를 사용한다. 플래시 메모리에 I/O요청이 버퍼 관리 방법에 따라 변하기 때문에 버퍼 캐시 관리는 플래시 저장장치의 높은 성능을 위해 중요하다. Jo[12]는 FAB이라 불리는 플래시를 위한 버퍼 관리 방법을 제안하였다. 논리 블록의 모든 페이지를 한번에 버퍼 캐시에서 내보내는 블록 레벨 버퍼 관리 방법을 사용하여 GC의 비용을 줄이는 기법이다. FAB에서는 버퍼 캐시에 가장 많은 페이지를 가지는 논리 블록이 교체 대상으로 선택된다. Park[13]은 플래시 메모리에 쓰기 요청을 줄이기 위해 버퍼 캐시에서 더티(dirty) 페이지의 플러쉬(flush)를 지연시키는 CFLRU 교체 정책을 제안하였다. 또한, Kim[14]은 BPLRU 버퍼 관리 방법을 제안하였는데, FAB처럼 희생 블록의 모든 페이지를 플러쉬하지만 블록 레벨 LRU를 기반으로 희생 블록을 결정한다는 차이가 있다. 또한, BPLRU는 블록 채우기(block padding) 기법을 사용하여 모든 로그 블록이 유효한 페이지의 복사 없이 GC가 가능하도록 했다.

최근에는 BA-GC(Buffer-Aware Garbage Collection) 기법[15]이 제안되었는데, 버퍼 캐시와 플래시 메모리에 쓰여진 중복된 페이지를 고려하여 GC중에 불필요한 페이지 이동을 제거하기 위해 더티상태의 중복 페이지는 플래시 메모리로 강제로 쫓아내는 기법을 사용한다.

Li[6]은 플래시 메모리 기반의 가상 메모리 시스템을 위해 DA-GC(Duplication-Aware Garbage Collection) 기법을 제안하였다. DA-GC는 플래시 메모리에서 중복된 페이지를 복사하지 않는다. 그러므로 해당 페이지는 GC가 희생 블록을 삭제할 때 플래시 메모리에서 제거되고 메인 메모리에만 남아 있다. DA-GC의 대상은 가상 메모리 시스템의 스왑 공간이기 때문에, 갑작스런 전원 차단에 의해 중복된 페이지가 메인 메모리에서 사라져도 큰 문제가 되지는 않는다. GC 후에 메인 메모리에 남아 있는 중복된 페이지는 스왑-아웃 될 때 플래시 메모리에 쓰여진다. DA-GC는 페이지 레벨 매핑 FTL에서 GC 부하를 크게 줄이지만, 하이브리드 매핑 FTL에서는 GC 부하가 오히려 증가하게 된다. 그 이유는 DA-GC가 로그 버퍼로 더욱 많은 쓰기 요청을 유발하기 때문에 하이브리드 매핑에서 빈번한 GC가 발생하게 되기 때문이다.

본 논문에서 제안하는 기법은 DA-GC를 기반으로 하고 있다. 하지만, 중복된 데이터의 지역성을 고려하여 하이브리드 매핑 FTL에서 DA-GC의 문제점을 해결한다.

Jung[16]은 GC의 부하를 줄이기 위해 OS에서 직접 플래시 스왑 공간을 관리하는 FASS(Flash-Aware Swap System)을 제안하였으며, Kwon[17]은 GC의 부하를 최소화하고 플래시 메모리의 수명을 늘리기 위해 플래시 스왑 공간을 위한 그리디(greedy) GC 방법[18]을 변형 하였다. 이 기법에서는 최근에 스왑-아웃된 페이지가 스왑-인 될 가능성이 높다는 것을 가정하였다. 이러한 플래시를 고려한 스와핑 기법은 제안하는 기법과 함께 사용될 수 있다.

## 3. 제안 기법의 동기

본 절에서는 DA-GC 기법을 소개하고 하이브리드 매핑에서의 문제점을 서술한다. 그림 1은 중복성을 고려하지 않은 DU-GC(Duplication-Unaware Garbage Collection)의 FAST 하이브리드 매핑에서의 예를 보여준다. 페이지 캐시는 여섯 개의 페이지를 가지며 접근 시간에 따라 정렬되어 있는데, 페이지 P2는 LRU(Least-Recently-Used) 페이지이며, 페이지 P9은 MRU(Most-Recently-Used) 페이지이다. 페이지 P2와 P11은 더티 상태(즉, 페이지 캐시와 플래시 메모리에 있는 페이지가

서로 다른 데이터)이고 나머지 페이지는 모두 클린상태이다. 플래시 메모리는 물리 블록 번호(PBN)가 0~6번인 일곱 개의 물리 블록으로 구성되어 있다고 가정한다. PBN 0, PBN 1, PBN 2는 데이터 블록으로 할당되어 있고, PBN 3과 PBN 4는 로그 블록으로 할당되어 있다. PBN 5과 PBN 6은 GC를 위해 남아 있는 빈 블록이다. 각 플래시 메모리는 4개의 페이지로 구성된다고 가정한다. 로그 블록은 논리 페이지 P1, P3, P4, P5, P8, P10의 데이터를 가지고 있으며, 데이터 블록 내의 해당하는 페이지는 무효화되어 있다. 예를 들어, 로그 버퍼에서 PBN 3은 데이터 블록 PBN 0와 PBN 2와 연관되어 있으며, PBN 4는 PBN 1과 연관되어 있다.

로그 버퍼에 빈 공간이 없기 때문에, GC가 발생한다. 만약 PBN 3이 희생 블록으로 선택된다면, GC는 PBN 3, PBN 0, PBN 2의 모든 유효 페이지들을 빈 블록 PBN 5, PBN 6에 복사해야 하는데, 유효 페이지 P0~P3는 PBN 5에, P8~P11은 PBN 6에 복사된다. 모든 로그 블록과 이와 연관된 데이터 블록에 있는 모든 유효 페이지가 빈 블록으로 병합되기 때문에, 이러한 과정을 블록 병합이라고 한다. 블록 병합 후에, PBN 5와 PBN 6은 데이터 블록으로 변경되며, PBN 0, PBN 2그리고 PBN 3은 삭제되어 그 중에 한 블록이 로그 블록으로 할당된다.

DU-GC는 페이지 캐시에 있는 중복 페이지를 고려하지 않는다. 만약 페이지 캐시의 정보를 알고 있다면, 페이지 캐시와 플래시 메모리에 동시에 존재하는 중복된 페이지를 고려하여 더 효과적인 GC가 수행될 수 있다. 그림 2는 DA-GC 기법[6]을 보여준다. GC가 희생 블록으로 PBN 3을 선택할 때, 페이지 캐시가 가지고 있는 중복 페이지 P1, P2, P9, P11은 복사하지 않는다. 그러

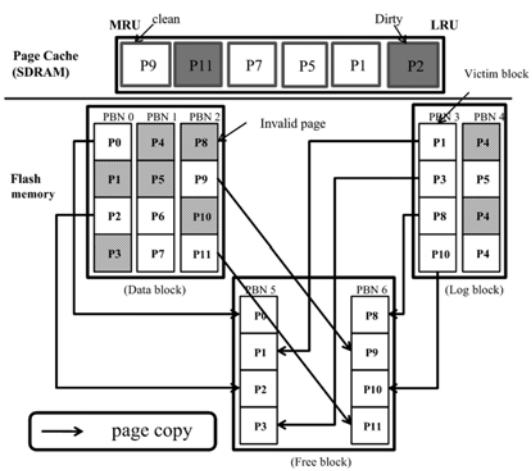


그림 1 DU-GC의 예

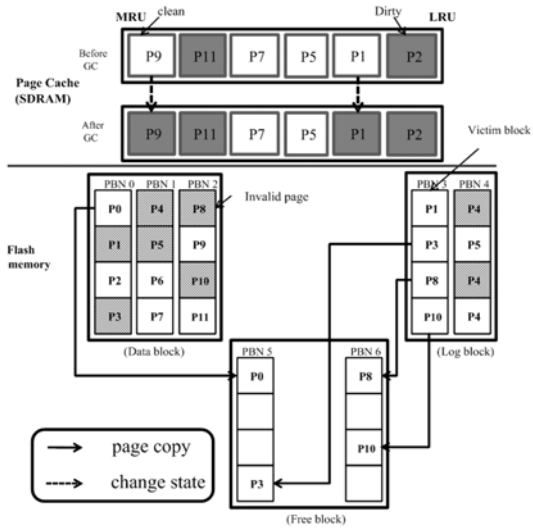


그림 2 DA-GC의 예

므로, 페이지 복사의 수는 반으로 줄어든다. 대신에 P1와 P9는 페이지 캐시에서 내보내질 때, 플래시 메모리에 쓰여져야 하기 때문에 페이지 캐시에서 더티상태로 변경된다.

DA-GC 기법은 블록 병합 비용을 상당히 줄이지만 페이지 캐시에서 클린상태의 중복 페이지가 더티상태로 변경되면서 더 많은 페이지가 플래시 메모리로 보내질 것이다. 페이지 레벨 매핑에서는 페이지들이 어느 위치에나 쓰여질 수 있기 때문에 DA-GC 기법에 의해 증가된 더티 페이지 쓰기 횟수가 GC에 부정적인 영향을 주지 않는다. 그러므로 DA-GC는 페이지 매핑에서 효과적인 방법이다. 그러나 하이브리드 매핑에서는 DA-GC가 빈번한 GC를 유발시키게 되는 데, 예를 들어, 그림 2에서 PBN 5와 PBN 6에 있는 4개의 물리 페이지는 DA-GC에서 사용되지 않고 있으며, 대신에 페이지 캐시에서 P1과 P2가 페이지 교체에 의해 플러쉬 될 때, 로그 블록에 쓰기 동작을 발생시킨다. 결과적으로, 로그 블록은 더 빠르게 빈 공간을 소비한다.

만약 GC 과정에서 중복 페이지를 더티상태로 변경하지 않고, 또한 해당 페이지들이 플러쉬될 때까지 호스트의 요청에 의해 더티상태로 변경되지 않는다면, 이 페이지들은 플래시 메모리에 기록되지 않을 것이다. 특히, 페이지 P1은 최근에 사용되지 않았기 때문에, 페이지 캐시에서 플러쉬 되기 전에 업데이트 되어 더티상태로 변경 될 가능성이 매우 낮다. 그러므로 GC 동안에 페이지 P1은 중복 페이지이지만 복사하는 것이 더 유리하다. 그러나 페이지 P9은 MRU 페이지이기 때문에 GC에서 상태를 변경하지 않더라도 호스트 요청에 의해서

더티상태로 변경될 가능성이 높기 때문에 GC 동작에서 P9의 복사를 제외시키는 것이 GC의 횟수를 증가시키지는 않을 것이다. 결론적으로, DA-GC 기법은 중복 페이지의 지역성을 고려하여 적용되어야 한다.

하이브리드 매핑에서 이러한 DA-GC의 문제를 해결하기 위해, 본 논문에서는 지역성을 고려한 희생 블록 선택 방법(LDA-VBS)과 지역성을 고려한 블록 병합 방법(LDA-BM)을 제안한다. 이 방법들은 페이지 캐시를 LRU 영역과 MRU 영역으로 나누고 각 영역에 다른 정책을 사용한다. LDA-VBS방법은 페이지 캐시의 LRU 영역에 있는 클린 페이지를 중복 페이지로 포함하는 로그 블록을 가능한 선택하지 않는다. LDA-BM 방법은 만약 LRU 영역의 클린 페이지가 희생 블록과의 중복 페이지라면, 병합 동안에 그 페이지를 복사하고 해당 페이지를 클린상태로 유지한다. 또한, GC 동안에 불필요한 페이지 복사를 줄이기 위해 LRU 영역의 더티 페이지를 강제로 쫓아내는 방법(LDE)도 제안한다. 이 방법은 GC 동안에 불필요한 페이지 복사를 줄이는 DA-GC의 장점을 활용하면서 하이브리드 매핑에서 DA-GC의 빈번한 GC 발생을 막을 수 있다.

## 4. 지역성을 고려한 GC

### 4.1 지역성과 중복을 고려한 희생 블록 선택(LDA-VBS)

일반적인 희생 블록 선택 알고리즘은 희생 블록을 선택하기 위해 블록 병합 비용만을 고려한다. 그러나 페이지 캐시의 LRU 영역에 있는 클린상태의 중복 페이지가 더티상태로 변경되는 것을 막기 위해, 병합 비용뿐만 아니라 DA-GC의 쓰기 요청의 증가로 발생할 수 있는 잠재적인 손실도 고려해야 한다. 제안된 LDA-VBS방법은 GC 부하와 잠재적인 손실을 모두 고려하여 최적화하는 기법이다.

DA-GC 기법에서, 희생 로그 블록 $L_i$에 대해 GC 부하 $C_{GC}(L_i)$를 다음과 같이 나타낼 수 있다.

$$C_{GC}(L_i) = (A(L_i)+1)\cdot C_e + \delta(L_i)\cdot(C_r + C_w) \quad (1)$$

$A(L_i)$는 $L_i$에 연관된 데이터 블록 개수를 나타내며 $\delta(L_i)$는 로그 블록 $L_i$와 로그 블록 $L_i$에 연관된 데이터 블록의 페이지 중에서 페이지 캐쉬와 중복되지 않는 유효 페이지의 수(플래시 메모리에만 존재하는 페이지)를 나타낸다. 예를 들어, 그림 2에서 $A$(PBN 3)는 2이며, $\delta$(PBN 3)는 4이다. $C_e$, $C_w$, $C_r$는 플래시 메모리에서 각각 블록 삭제, 페이지 쓰기, 페이지 읽기를 위한 비용을 나타낸다. DA-GC는 블록 병합 동안에 중복 페이지 복사를 하지 않기 때문에 단지 $\delta(L_i)$의 플래시 페이지 읽기와 쓰기의 수만 요구된다. 블록 병합 후에, 데이터 블록 $A(L_i)$개와 하나의 로그 블록이 삭제되므로 $A(L_i)+1$의 블록 삭제 비용이 요구된다.

그러나, 3장에서 설명한 것처럼 DA-GC를 사용한다면, 클린상태의 중복 페이지가 더티상태로 변경되어 페이지 캐시에서 플래시 메모리로 더 많은 쓰기 요청을 유발시키게 된다. 그러므로, DA-GC는 다음과 같은 잠재적인 손실을 유발한다.

$$C_{loss}(L_i) = \gamma(L_i)\cdot(C_w + \alpha) \quad (2)$$

$\gamma(L_i)$는 로그 블록 $L_i$의 중복 페이지 중에 DA-GC에 의해 해당 페이지가 페이지 캐시에서 클린상태에서 더티상태로 변경되면서 플래시 메모리로 밀려날 때까지 호스트에 의해 업데이트되지 않는 페이지의 개수를 나타낸다. 그림 2에서, 2개의 더티 페이지가 DA-GC에 의해 발생되지만, 페이지 P9는 호스트 요청에 의해 더티상태로 변경될 가능성이 높기 때문에, $\gamma(L_i)$의 값은 2보다 작을 것이다. 플래시 메모리로 더티 페이지를 기록하기 위한 비용이 $\gamma(L_i)\cdot C_w$이다. 게다가 쓰기 요청은 더 많은 GC를 발생시키므로 증가하는 부하를 고려하여 하나의 더티 페이지 쓰기가 유발하는 평균 블록 병합 비용인 $\alpha$를 추가하였다. 플래시 메모리에 쓰여진 더티 페이지는 블록 병합 시에 한번의 페이지 읽기와 쓰기를 유발하고 플래시 한 블록에 해당하는 페이지의 수마다 한번의 삭제가 발생하기 때문에, $\alpha$의 근사값은 $(C_r+C_w+C_e)/(\text{블록의 페이지 개수})$로 나타낼 수 있다.

그러나, 미래의 호스트 요청에 대한 정보 없이 GC 동안에 $\gamma(L_i)$의 정확한 값을 아는 것은 불가능 하다. 해당 값을 예측하기 위해서, 본 기법에서는 지역성 정보를 기반으로 페이지 캐시에서 각 페이지의 업데이트 확률을 예측하는 3-region LRU 버퍼[15] 기법을 사용하였다. 3-region LRU 버퍼 기법은 initial-영역과 evict-영역, update-영역으로 구성되며 업데이트에 따라 각 영역의 크기가 변경된다. 본 기법에서는 LRU 영역을 evict-영역으로 할당하였으며 MRU 영역을 initial-영역과 update-영역으로 사용하였다.

GC 부하와 잠재적인 손실을 모두 고려하면 전체 GC 비용은 다음과 같이 나타낼 수 있다.

$$C_{total} = C_{GC}(L_i) + C_{loss}(L_i) \quad (3)$$

LDA-VBS 방법은 가장 작은 $C_{total}$ 값을 갖는 희생블록을 선택한다. 이를 통해 큰 잠재적인 손실을 발생시키는 희생블록의 선택을 막을 수 있다.

그러나 특정 로그 블록이 업데이트가 자주 발생하지 않거나 페이지 캐시에서 참조하는 페이지의 수가 적을 때, 제안한 희생 로그 블록 선택 방법은 특정 로그 블록이 희생 로그 블록으로 오랫동안 선택되지 않을 수 있다. 하이브리드 매핑 FTL에서 특정 로그 블록의 병합이 발생하지 않고 지속적으로 로그블록의 영역으로 할당되어 있으면, 로그 영역을 효과적으로 사용할 수 없으

면 마모도 관리(wear leveling) 측면에서도 좋지 않다. 그러므로, 로그 블록에서 오랜 시간 블록 병합이 발생하지 않은 블록은 우선적으로 희생 블록으로 선택되도록 하였다.

### 4.2 지역성과 중복을 고려한 블록 병합(LDA-BM)

페이지 캐시에 있는 여러 페이지들은 업데이트될 가능성이 다르기 때문에 LDA-BM 방법은 각 페이지에 대하여 미래의 접근 가능성을 고려하여 다른 정책을 적용한다. 페이지 캐시의 MRU 영역에 있는 클린 페이지는 DA-GC에 의해 상태가 변경되지 않을 지라도 페이지 캐시에서 밀려나기 전에 더티상태로 변경될 가능성이 크다. 반대로 페이지 캐시의 LRU 영역의 클린 페이지는 호스트 요청에 의해 더티상태로 변경될 가능성이 낮다. 따라서, LRU 영역의 클린 페이지에 대해서는 클린 데이터가 더티 데이터로 변경되는 것을 막기 위해서 DA-GC 기법을 적용하지 않는 것이 좋다. 비록 GC에서 유효 페이지 복사 비용은 증가하겠지만 큰 비용을 발생시키는 GC의 빈도를 줄일 수 있게 된다.

그림 3은 LDA-BM 방법을 나타낸다. 페이지 캐시의 LRU 영역에 있는 클린 페이지 P1은 플래시 메모리에서 블록 병합과정 동안에 복사된다. 그리고 페이지 캐시에서 클린상태를 유지한다. 그러나, 페이지 캐시의 MRU 영역에 있는 클린 페이지 P9과 더티 페이지 P2와 P11에 대해서는 DA-BM 방법이 적용된다. 페이지 P9가 더티상태로 변경되어도 어차피 호스트 요청에 의해 더티상태로 변경될 가능성이 높기 때문에 해당 페이지의 상태 변경으로 인한 잠재적인 손실은 작을 것이다.

LDA-BM 방법을 사용할 때는 희생 블록 선택 방법

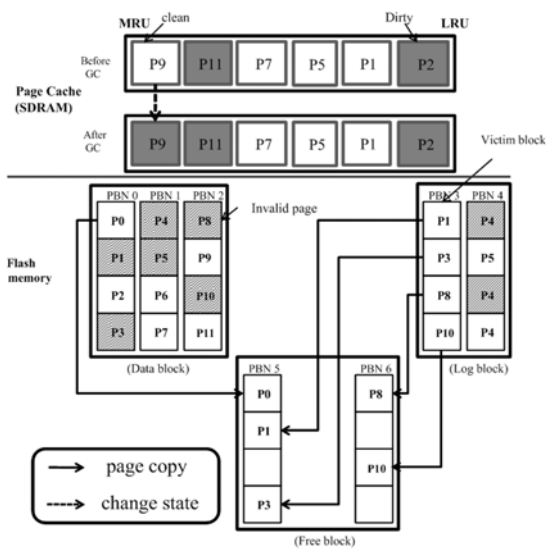


그림 3 LDA-BM의 예

이 변경되어야 하는데, 희생 로그 블록 $L_i$에 대해 GC 부하 $C_{GC}(L_i)$를 다음과 같이 나타낼 수 있다.

$$C_{GC}(L_i) = (A(L_i)+1)\cdot C_e + (\delta(L_i)+\pi(L_i))\cdot(C_r + C_w) \quad (4)$$

$\pi(L_i)$는 페이지 캐시의 LRU영역에 있는 클린상태의 중복 페이지 개수를 나타낸다. 식 (1)의 DA-GC 비용과 비교해 보면, LDA-BM은 플래시 메모리에서 더 많은 페이지를 복사하기 때문에 더 큰 GC 비용을 가진다. 하지만 LRU영역의 클린 페이지가 더티 페이지로 변경되지 않기 때문에 잠재적인 손실을 줄여주는 효과가 있다. DA-GC가 LDA-BM을 사용할 때 잠재적인 손실은 다음과 같다.

$$C_{loss}(L_i) = \gamma_{MRU}(L_i)\cdot(C_w + \alpha) \quad (5)$$

$\gamma_{MRU}(L_i)$은 페이지 캐시의 MRU영역의 페이지 중에서 DA-GC에 의해 클린상태에서 더티상태로 변경되면서 호스트 요청에 의해 추가로 업데이트되지 않는 페이지의 개수를 나타낸다. $\gamma(L_i)$이 $\gamma_{MRU}(L_i)$보다 더 크기 때문에 잠재적인 손실은 식 (2)의 값보다 작다.

블록 병합에서 희생 로그 블록의 페이지는 표 1과 같이 중복성과 페이지 캐시에서의 상태에 따라 다섯 가지로 구분할 수 있다. 표 1은 블록 병합과정에서 LDA-BM기법을 사용할 때 각 페이지에 대하여 페이지 캐시에서의 상태 변화와 플래시 메모리 페이지의 복사 여부를 나타내고 있다. 일반적으로 블록 병합이 발생하면 모든 플래시 메모리는 유효한 페이지의 복사를 발생한다. LDA-BM 기법에서는 중복 페이지중에 페이지 캐시에 더티상태로 존재하거나 MRU영역에서 클린상태로 존재한다면 플래시 메모리에서 복사를 하지 않는다. 따라서 추가적인 복사의 부하를 방지한다. LRU영역의 클린상태의 페이지와 중복되지 않은 페이지는 병합 과정에서 페이지 복사가 발생한다.

표 1 LDA-BM 기법에서 페이지 변화와 페이지 복사

| | 중복 페이지 | | | | 중복되지 않은 페이지 |
|---|---|---|---|---|---|
| | MRU 영역 | | LRU 영역 | | |
| | Dirty | Clean | Dirty | Clean | |
| 페이지 캐시 | Dirty | Dirty | Dirty | Clean | • |
| 플래시 페이지 | No copy | No copy | No copy | Copy | Copy |

### 4.3 LRU 더티 페이지 축출(LRU dirty page eviction)

LRU 더티 페이지 축출(LDE) 기법은 GC 비용을 줄이기 위해 페이지 캐시의 LRU 영역에 있는 더티상태의 중복 페이지를 활용한다. LRU 영역의 더티 페이지는 더 이상의 업데이트 없이 플래시 메모리에 기록 될 가능성

이 크기 때문에 GC 동안에 페이지 캐시에서 플래시 메모리로 더티상태의 중복 페이지를 복사하는 게 좋다. 이를 통해, DA-GC에 의해 사용되지 않은 플래시 메모리 페이지의 수를 줄여 효과적으로 데이터 블록을 이용할 수 있다. 페이지 캐시에서 플래시 메모리로 복사된 더티 페이지는 페이지 캐시에서 클린상태로 변경된다.

블록 병합 동작 중에 LDA-BM과 LDE 방법을 동시에 사용할 수 있는데, LRU 영역의 클린상태의 중복 페이지에 대해서는 LDA-BM를 적용하고, LRU 영역의 더티상태의 중복 페이지에 대해서는 LDE를 적용한다. LRU 영역의 페이지를 위해 다른 정책을 사용함으로써, DA-GC에 의해 발생되는 잠재적인 GC 부하를 줄일 수 있게 된다.

LDA-BM과 LDE 방법이 모두 사용될 때, 희생 로그 블록 $L_i$ 에 대해 GC 부하 $C_{GC}(L_i)$는 다음과 같다.

$$\begin{aligned} C_{GC}(L_i) &= (A(L_i)+1)\cdot C_e \\ &+ (\delta(L_i)+\pi(L_i))\cdot(C_r+C_w) \\ &+ \theta(L_i)\cdot(C_b+C_w) \end{aligned} \quad (6)$$

$\theta(L_i)$는 LRU 영역에 있는 더티상태의 중복 페이지의 수를 나타낸다. $C_b$ 는 페이지 캐시에서 플래시 메모리로 복사하는 비용을 나타내며 $C_b$ 가 $C_r$ 보다 더 크다고 가정하였다. 식 (1)과 (4)의 DA-GC 비용과 비교해 보면, $\theta(L_i)$만큼 페이지 캐시에서 플래시 메모리로 복사되기 때문에 LDA-BM과 LDE를 함께 사용한 방법이 더 큰 GC 비용을 발생시킨다. 그러나, LDE 방법은 GC에 대한 잠재적인 이득을 유발시키는데, 페이지 캐시 LRU 영역의 더티상태의 중복 페이지를 클린 페이지로 변경하기 때문에, 로그 블록의 쓰기 요청의 수가 감소한다. 그러므로, 전체 GC 비용은 다음과 같다.

$$\begin{aligned} &C_{total} = C_{GC}(L_i) + C_{loss}(L_i) - C_{benefit}(L_i), \\ &where\ \ C_{loss}(L_i) = \gamma_{MRU}(L_i)\cdot(C_w+\alpha) \text{ and} \\ &\quad C_{benefit}(L_i) = \gamma_{LRU}(L_i)\cdot(C_w+\alpha). \end{aligned} \quad (7)$$

$\gamma_{LRU}(L_i)$는 LDE에 의해 더티상태에서 클린상태로 변경되면서 호스트 요청에 의해 추가로 업데이트 되지 않는 LRU 영역의 중복 페이지의 수를 나타낸다.

표 2는 블록 병합 동안에 LDA-BM과 LDE 방법을 동시에 사용할 때 페이지 캐시에서 상태 변화와 페이지 복사의 발생 여부를 나타내고 있다. LDA-BM 방법과

표 2 LDA-BM+LDE기법에서 페이지 변화와 페이지 복사

| | 중복 페이지 | | | | 중복되지 않은 페이지 |
|---|---|---|---|---|---|
| | MRU 영역 | | LRU 영역 | | |
| | Dirty | Clean | Dirty | Clean | |
| 페이지 캐시 | Dirty | Dirty | Clean | Clean | • |
| 플래시 페이지 | No copy | No copy | Copy from page cache | Copy | Copy |

표 3 각 기법에서 페이지 캐시 상태의 변화

| 페이지 캐시 영역 | GC 전 | 개수 | GC 후 | | |
|---|---|---|---|---|---|
| | | | DA-GC LDA-VBS | LDA-BM | LDA-BM + LDE |
| MRU 영역 | Dirty | $\rho(L_i)$ | Dirty | Dirty | Dirty |
| | Clean | | Dirty | Dirty | Dirty |
| LRU 영역 | Dirty | $\theta(L_i)$ | Dirty | Dirty | Clean |
| | Clean | $\pi(L_i)$ | Dirty | Clean | Clean |

마찬가지로 LRU 영역의 클린상태의 페이지와 중복되지 않은 페이지는 복사를 한다. 또한 LDE 기법을 적용하여 LRU 영역의 더티상태의 중복 페이지는 페이지 캐시로부터 플래시 메모리로 복사가 발생한다. 복사된 LRU 영역의 더티상태 페이지는 추가적으로 플러쉬될 때 복사되지 않도록 클린상태로 변경된다.

표 3은 각 기법의 결과로 발생하는 중복 페이지의 변화를 보여주고 있다. DA-GC와 LDA-VBS는 가장 적은 GC 비용을 갖지만 모든 클린상태의 중복 페이지가 더티 페이지로 변경되기 때문에 가장 큰 잠재적인 GC 비용이 발생한다. LDA-BM은 DA-GC보다 더 큰 GC 비용을 유발하지만 LRU 영역의 클린상태의 중복 페이지의 상태가 변경되지 않기 때문에 더 적은 잠재적인 비용을 갖는다. LDA-BM에 추가적으로 LDE를 사용하면 GC 비용은 더욱 증가하지만, 페이지 캐시의 더티상태의 중복 페이지가 클린 페이지로 변경되기 때문에 잠재적인 이득이 있다.

## 5. 실 험

### 5.1 실험환경

제안된 기법을 평가하기 위하여 트레이스 기반의 시뮬레이터를 제작하여 사용하였다. 본 논문에서 제안하는 시뮬레이터의 전체적인 구조는 그림 4와 같다. 이 시뮬레이터는 페이지 캐시와 플래시 메모리로 구성되며, 논리적 주소를 플래시 메모리에서 사용되는 물리적 주소로 변경하는 FTL과 GC제어하는 부분을 포함한다. 사용된 플래시 메모리 모델은 삼성 SLC NAND 플래시 메모리[19]이다. 각 플래시 블록은 2KB 페이지 64개로 구성되며, 페이지 읽기, 페이지 쓰기, 블록 삭제의 비용은 각각 25 usec, 200 usec, 2 msec이다. 페이지 캐시는 LRU와 MRU 영역으로 나누기 위해 3-region LRU 버퍼 알고리즘[15]을 사용하여 실험했다. 시뮬레이터는 페이지 캐시와 플래시 메모리의 로그 블록 크기를 변경하여 설정할 수 있으며 희생 로그 블록 선택과 블록 병합 기법을 선택하여 설정 가능하다. 또한 GC가 발생할 때 저장 장치 시뮬레이터가 페이지 캐시의 중복 페이지를 참조하도록 하였다.

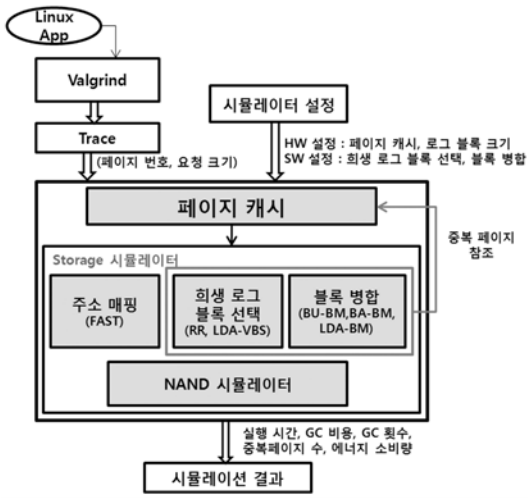


그림 4 시뮬레이터의 구조

여러 가지 하이브리드 매핑 FTL들이 존재하며 제안 기법은 이들 기법에서 모두 좋은 성능을 보이지만, 본 논문에서는 FTL종류에 따른 성능의 변화를 보여주기보다는 페이지 캐시와 로그 버퍼의 설정에 따른 성능 변화를 보여주기 위해 간단한 구조의 FAST 하이브리드 매핑 FTL을 사용하였다.

실험에서는 표 4에 나타난 7개의 기법을 비교하였는데, 각 기법은 서로 다른 희생 블록 선택 방법과 블록 병합 방법을 사용한다. 비교 대상인 기본적인 희생 블록 선택 알고리즘은 가장 오래된 로그 블록을 선택하는 라운드 로빈 선택 방법을 가정하였다. 일반적으로 가장 오래된 블록은 가장 적은 개수의 유효 페이지를 가지고 있기 때문에, GC 비용이 적게 발생한다.

실험에서 사용된 트레이스는 리눅스 시스템 환경에서 Valgrind[20]에 의해 수집된 표 5의 다섯 개의 가상 메모리 트레이스를 사용하였다.

### 5.2 성능 분석

그림 5는 각 GC 기법의 전체 I/O 실행시간을 DU-GC의 전체 I/O 실행시간에 정규화하여 보여준다. I/O 실행

표 4 평가된 기법의 요약

| 기법 | 희생블록선택 | 블록병합 | LDE |
|---|---|---|---|
| DU-GC | RR | DU-BM | No |
| DA-GC | RR | DA-BM | No |
| LDA-GC1 | LDA-VBS | DA-BM | No |
| LDA-GC2 | RR | LDA-BM | No |
| LDA-GC3 | LDA-VBS | LDA-BM | No |
| LDA-GC4 | RR | LDA-BM | Use |
| LDA-GC5 | LDA-VBS | LDA-BM | Use |

RR : Round-Robin policy

표 5 실험에서 사용된 워크로드의 특징

| 응용프로그램 | 쓰기 비율 | 시나리오 |
|---|---|---|
| acrobat | 23.1% | view PDF file |
| gqview | 21.1% | picture modification after viewing image file |
| kword | 14.9% | data modification |
| mozilla | 13.2% | web information search (google, yahoo, amazon etc.) |
| office | 19.7% | slide show |

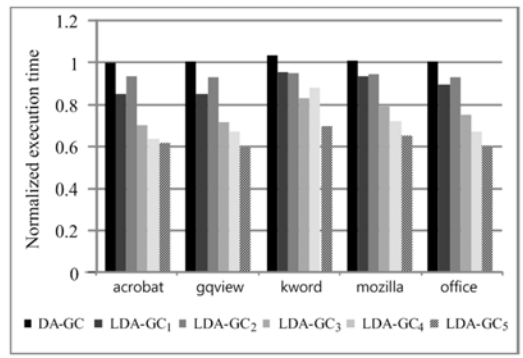


그림 5 전체 실행 시간 비교

시간은 페이지 스왑-아웃 뿐만 아니라 GC에 의해 발생되는 플래시 읽기와 쓰기 비용을 포함한다. 페이지 캐시 크기는 4MB이고 플래시 메모리는 32개의 로그 블록을 가지고 있는 것으로 가정했다. DA-GC의 잠재적인 손실이 중복 페이지를 복사하지 않음으로써 감소하는 GC 비용보다 더 크기 때문에 DA-GC의 성능은 DU-GC와 유사하거나 더 나쁘다. LDA-$GC_1$과 LDA-$GC_2$의 결과에서, 평균 10%의 성능 향상을 보인 LDA-VBS 방법이 평균 6%의 성능 향상을 보인 LDA-BM 방법보다 더 효과적인 것을 알 수 있다. 이것은 LDA-VBS가 잠재적인 손실뿐만 아니라 GC 부하를 줄일 수 있기 때문이다.

LDA-VBS와 LDA-BM을 모두 사용하는 LDA-$GC_3$는 두 기법의 상승 작용 때문에 평균 24% 성능 향상으로 더 큰 효과를 보여준다. LDA-BM과 LDE 모두 사용하는 LDA-$GC_4$기법은 성능을 평균 28% 향상시켰다. 제안된 모든 방법을 사용하는 LDA-$GC_5$는 DU-GC와 비교해서 평균 37%의 I/O 실행 시간을 줄였다.

본 실험에서는 또한 페이지 캐시 크기의 영향을 조사하였다. 그림 6은 페이지 캐시의 크기를 1MB부터 16MB까지 변경하면서 kword 워크로드의 전체 I/O 실행 시간과 중복 페이지($N_{dup}$)의 평균 수를 보여준다. 로그 블록의 수는 32개로 고정되었다. 페이지 캐시의 크기가 증가하면 페이지 캐시의 적중률(hit rate)이 증가하기 때문에 전체적으로 성능은 향상된다. LDA-$GC_3$와 LDA-

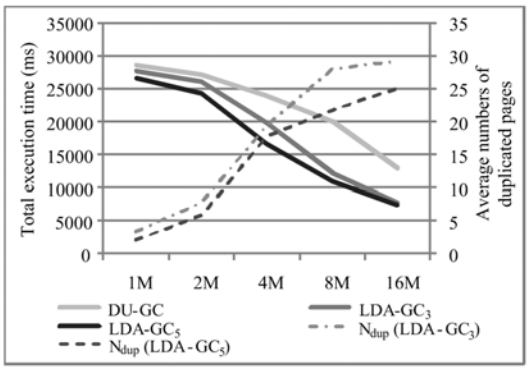


그림 6 페이지 캐시의 크기의 변경에 따라 실행시간과 중복 페이지의 평균 수(kword 워크로드)

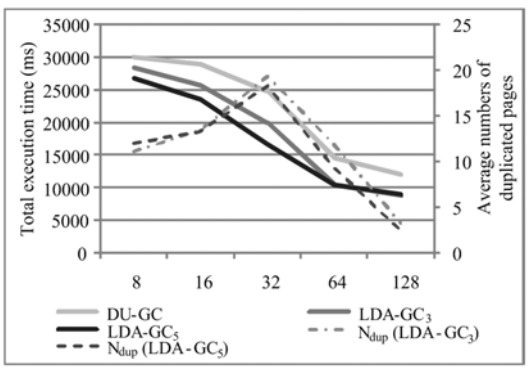


그림 7 로그 블록 수의 변경에 따라 실행시간과 평균 중복 페이지의 수(kword 워크로드)

$GC_5$는 페이지 캐시 크기와 상관없이 항상 DU-GC보다 더 좋은 성능을 보인다. 더구나, 그림 6에서 나타난 것처럼 페이지 캐시 크기가 증가하면서 중복 페이지의 수도 증가하기 때문에 DU-GC와 다른 방법들 사이에 성능 차이가 증가하는 추세이다. 많은 중복 페이지가 있을 때, 제안된 기법은 GC 부하를 줄이기 위한 더 큰 여지를 갖는 것이다.

본 실험에서는 또한 플래시 메모리 로그 버퍼 크기의 영향을 조사하였다. 그림 7은 플래시 메모리의 로그 블록의 개수를 8부터 128까지 변경하면서 I/O의 실행 시간과 중복 페이지의 평균 개수를 보여준다. 페이지 캐시는 4MB로 고정되어 있다. 로그 블록의 수가 증가하면서 전체적으로 실행 시간은 감소하는데, 이것은 많은 로그 블록이 있을 때 어떤 로그 블록이 희생 블록으로 선택될 때까지 시간이 오래 걸려서 희생 블록의 대부분의 페이지가 무효화되어 있기 때문에 GC가 적은 페이지 복사 비용을 발생시키기 때문이다. 로그 블록의 수가 증가하면서 DU-GC와 제안된 방법 사이의 성능 차이가

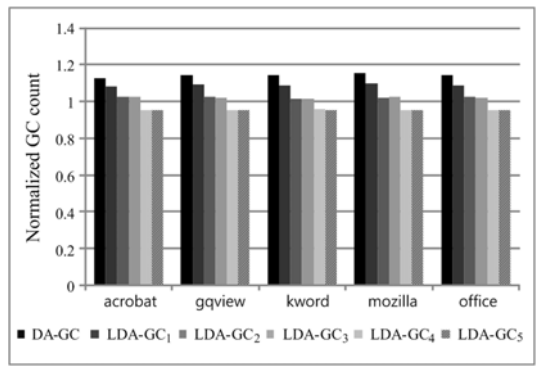


그림 8 전체 GC 횟수

증가하는데, 이것은 많은 로그 블록이 있을 때 LDA-VBS가 더 적절한 희생 블록을 찾을 수 있기 때문이다.

많은 중복 페이지를 가지는 로그블록을 선택하는 LDA-VBS는 로그 블록의 수가 증가하면서 더 많은 희생 후보 블록을 가지게 됨에 따라 중복 페이지의 평균 개수가 증가하는 것이다. 그러나, 로그 블록 수가 32개일 때 최고점에 달하고, 너무 많은 로그 블록이 있을 때는 희생 블록이 적은 개수의 유효 페이지를 가지므로 중복 페이지의 수가 다시 감소한다.

그림 8은 제안된 GC 방법에서 벤치마크의 실행 중에 발생하는 GC의 횟수를 나타낸다. 이 값은 DU-GC에서의 GC 발생 횟수에 정규화 되었다. LDE방법을 사용하는 LDA-$GC_4$와 LDA-$GC_5$는 DU-GC보다 4~5% 정도 GC 횟수를 감소시켰다. 다른 방법들(DA-GC, LDA-$GC_1$, LDA-$GC_2$, LDA-$GC_3$)은 GC 비용에서 잠재적인 손실 때문에 DU-GC보다 더 자주 GC를 발생시킨다. 그림 5에서 제안된 모든 GC 방법에서 성능 향상이 있었으므로 DU-GC보다 제안된 GC 방법들이 각 GC 동작에서 더 적은 비용을 발생시킨다는 것을 추론할 수 있다.

그래서, 그림 9에서 보여주는 것처럼 제안된 GC 방법에서 GC 마다 발생하는 평균 비용을 측정하였다. 이 수치는DU-GC의 평균 GC비용을 기준으로 정규화한 값을 나타낸다. 모든 GC 방법들은 DU-GC보다 더 적은 평균 GC비용을 발생시킨다. 특히, LDA-VBS는 GC 비용을 고려하여 희생 로그 블록을 선택하기 때문에, LDA-VBS방법을 사용하는 LDA-$GC_1$, LDA-$GC_3$, LDA-$GC_5$의 평균 GC 비용이 다른 방법보다 더 낮다. 비록 그림 8에서 LDA-$GC_4$와 LDA-$GC_5$는 GC 횟수가 유사할지라도 LDA-$GC_5$가 평균 GC비용이 더 낮기 때문에 LDA-$GC_4$보다 더 좋은 성능을 보여주고 있다.

그림 10은 각 GC 기법의 에너지 소비량을 DU-GC의 에너지 소비량에 정규화하여 보여준다. 플래시 메모리의 에너지 소비값은 [6]을 참조하였다. 각 기법의 GC의 횟

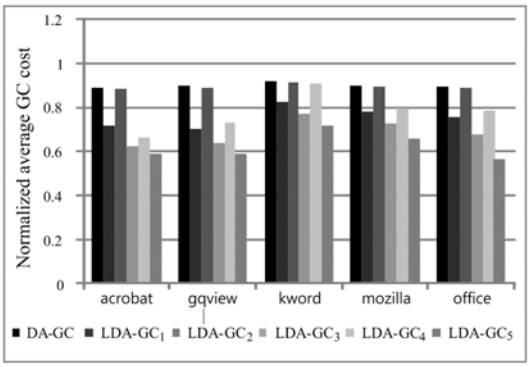


그림 9 평균 GC 비용

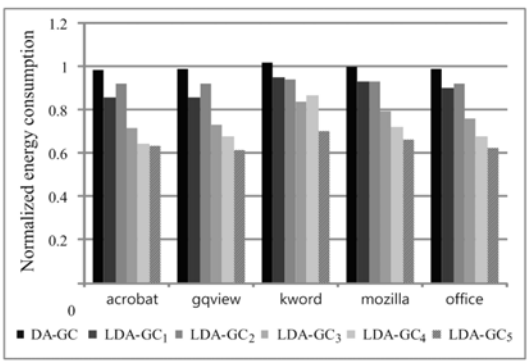


그림 10 에너지 소비량 비교

수 감소와 평균 GC 비용의 감소로 인해 제안한 기법들의 에너지 소비량이 감소하고 있다. 제안된 모든 방법을 사용하는 LDA-$GC_5$는 DU-GC와 비교해서 평균 34%의 에너지 소비량을 감소시켰다.

## 6. 결 론

플래시 메모리는 가상 메모리 시스템의 스왑 공간으로 좋은 장치이다. 본 논문에서는 플래시 메모리 기반의 가상 메모리 시스템을 위해, 플래시 메모리에서 중복된 페이지를 제거함으로써 GC의 부하를 줄일 수 있는 지역성과 중복을 고려한 GC 기법을 제안했다. 기존의 중복 페이지를 고려한 GC기법이 가진 하이브리드 매핑 FTL에서의 잠재적인 손실 문제를 해결하기 위해, 제안된 희생 로그 블록 선택(LDA-VBS) 기법은 GC 부하와 잠재적인 손실을 모두 고려한다. 또한, 제안된 블록 병합(LDA-BM)과 LRU 더티 페이지 축출(LDE) 기법은 페이지 캐시에서 각 페이지의 지역성을 고려하여 선택적으로 중복을 고려한 페이지 복사를 적용한다.

실험 결과, LDA-VBS와 LDA-BM 기법이 DU-GC와 비교했을 때 각각 10%와 6%의 전체 I/O 실행 시간을 감소시킴을 확인했다. LDA-VBS와 LDA-BM기법을 동시에 적용함으로써, 성능은 DU-GC와 비교해서 평균 24% 향상될 수 있었으며, 추가적으로 LDE를 채택함으로써 DU-GC와 비교해서 성능이 평균 37% 향상되었다.

## 참 고 문 헌

[1] C. Park, J. Kang, S. Park, and J. Kim, "Energy-aware demand paging on NAND flash-based embedded storages," In *Proc. International Symposium on Low Power Electronics and Design*, pp.338-343, 2004.

[2] Y. Joo, Y. Choi, C. Park, S. W. Chung, E.-Y. Chung, and N. Chang, "Demand paging for One NAND flash eXecute-In-Place," In *Proc. CODES+ISSS'06*, pp.229-234, 2006.

[3] J. In, I. Shin, and H. Kim, "SWL: A Search-While-Load demand paging scheme with NAND flash memory," In *Proc. Conference on Languages, Compilers, and Tools for Embedded Systems (LCTES)*, pp.217-225, 2007.

[4] Intel Corporation, "Understanding the flash translation layer (FTL) specification," http://developer.intel.com/.

[5] CompactFlash Association, http://www.compactflash.org/.

[6] H.-L. Li, C-L. Yang, H-W. Tseng, "Energy-aware flash memory management in virtual memory system," *IEEE Trans. Very Large Scale Integration System*, 16(8):952-964, 2008.

[7] A. Ban. Flash file system optimized for page-mode flash technologies, US Patent 5,937,425, Aug. 10, 1999.

[8] A. Ban. Flash file system, US Patent 5,404,485, Apr. 4, 1995.

[9] J. Kim, J. M. Kim, S. H. Noh, S. L. Min, and Y. Cho, "A space-efficient flash translation layer for compact flash systems," *IEEE Trans. on Consumer Electronics*, vol.48, no.2, pp.366-375, 2002.

[10] S. W. Lee, D. J. Park, T. S. Chung, W. K. Choi, D. H. Lee, S. W. Park, and H. J. Song, "A log buffer based flash translation layer using fully associative sector translation," *ACM Trans. on Embedded Computing Systems*, vol.6, no.3, 2007.

[11] C. I. Park, W. M. Cheon, J. G. Kang, K. H. Roh, W. H. Cho, and J. S. Kim, "A reconfigurable FTL (Flash Translation Layer) architecture for NAND flash-based applications," *ACM Trans. on Embedded Computing Systems*, vol.7, no.4, 2008.

[12] H. Jo, J. Kang, S. Park, J. Kim, and J. Lee, "FAB: Flash-aware buffer management policy for portable media players," *IEEE Trans. On Consumer Electronics*, vol.52, no.2, pp.485-493, 2006.

[13] S. Y. Park, D. Jung, J. U. Kang, J. S. Kim, and

J. Lee, "CFLRU: A replacement algorithm for flash memory," In *Proc. of the Int. Conf. on Compilers, Architecture and Synthesis for Embedded Systems*, pp.234-241, 2006.
[14] H. Kim and S. Ahn, "BPLRU: A buffer management scheme for improving random writes in flash storage," In *Proc. of the USENIX Conf. on File and Storage Technologies*, pp.239-252, 2008.
[15] S. Lee, D. Shin, and J. Kim, "Buffer-aware garbage collection techniques for NAND flash memory-based storage systems," In *Proc. of the Int. Work. on Software Support for Portable Storage (IWSSPS'08)*, pp.27-32, 2008.
[16] D. W. Jung, J. S. Kim, S. Y. Park, J. U. Kang, and J. Lee, "FASS: A flash-aware swap system," In *Proc. of the Int. Work. on Software Support for Portable Storage (IWSSPS'05)*, 2005.
[17] O. Kwon, K. Koh, "Swap-aware garbage collection for NAND flash memory based embedded systems," In *Proc. of the Seventh IEEE Int. Conf. on Computer and Information Technology*, 2007.
[18] M. Wu and W. Zwaenepoel, "eNVy: A non-volatile, main memory storage system," In *Proc. of the Int. Conf. on Architectural Support for Programming Languages and Operating Systems*, pp.86-97, 1994.
[19] Samsung Corp, "K9WBG08U1M NAND Flash Memory," 2007.
[20] N. Nethercote and J. Seward, "Valgrind: A framework for heavyweight dynamic binary instrumentation," *SIGPLAN Not.*, vol.42, no.6, pp.89-100, 2007.

지 승 구
2007년 2월 단국대학교 컴퓨터과학과 학사. 2008년 9월~현재 성균관대학교 정보통신공학부 임베디드 소프트웨어학과 석사과정. 관심분야는 플래시 메모리, 임베디드 소프트웨어

신 동 군
정보과학회논문지 : 시스템 및 이론
제 37 권 제 2 호 참조