실시간 시스템용 낸드 플래시 메모리를 위한 로그 버퍼 관리 기법

(Log Buffer Management Scheme for NAND Flash Memory in Real-Time Systems)

조 현 진 [†] 하 병 민 [†] 신 동 군 [††] 엄 영 익 [†††]
(Hyunjin Cho) (Byung Min Ha) (Dongkun Shin) (Young Ik Eom)

요 약 플래시 메모리는 일관된 성능, 저전력 및 내구성 등의 특징으로 인해 실시간 시스템에 적합한 저장장치로 주목 받고 있다. 하지만 플래시 메모리는 무효화된 페이지의 가비지 컬렉션 수행을 위한 정체 시간(blocking time)을 필요로 하는데, 기존의 플래시 메모리 관리 기법에서는 가비지 컬렉션을 위한 최대 정체 시간(worst case blocking time)과 최소 정체 시간(best case blocking time)의 차가 크다는 문제점이 있다. 본 논문에서는 KAST라 불리는 FTL(Flash Translation Layer)을 제안하며, 제안 시스템에서 사용자는 가비지 컬렉션에 따른 최대 정체 시간을 설정할 수 있도록 한다. 실험을 통해 KAST는 사용자가 설정한 시간 내 가비지 컬렉션을 완료하며, 기존 FTL 보다 10~15% 성능 향상을 보임을 확인한다.

키워드 : 플래시 메모리, 플래시 메모리 계층 변환, 로그 버퍼 관리 기법, 실시간 시스템

Abstract Flash memory is suitable for real time systems because of its consistent performance for random access, low power consumption and shock resistance. However, flash memory needs blocking time to perform a garbage collection to reclaim invalidated pages. Moreover, the worst-case garbage collection time is significantly longer than the best-case garbage collection time. In this paper, we propose a FTL (Flash Translation Layer) mapping scheme called KAST (K-Associative Sector Translation). In the KAST scheme, user can control the maximum association of the log block to limit the worst-case garbage collection time. Performance evaluation using simulation shows that not only KAST completes the garbage collection within the specified time but also provides about 10~15% better average performance than existing FTL schemes.

Key words : Flash memory, FTL(Flash Translation Layer), log buffer management scheme, real-time systems

· 이 논문은 2008년 정부(교육과학기술부)의 재원으로 한국연구재단의 지원을 받아 수행된 연구임(KRF-2008-314-D00351)
· 본 연구는 지식경제부 및 정보통신연구진흥원의 대학 IT연구센터 지원 사업의 연구결과로 수행되었음(IITA-2009-(C1090-0902-0046))

† 학생회원 : 성균관대학교 컴퓨터공학과
hjcho@ece.skku.ac.kr
chaosken@ece.skku.ac.kr
†† 정 회 원 : 성균관대학교 컴퓨터공학과 교수
dongkun@skku.edu
(Corresponding author임)
††† 종신회원 : 성균관대학교 컴퓨터공학과 교수
yieom@ece.skku.ac.kr

논문접수 : 2009년 2월 6일
심사완료 : 2009년 7월 30일

정보과학회논문지 : 시스템 및 이론 제36권 제6호(2009.12)

1. 서 론

플래시 메모리는 저전력, 비 휘발성, 높은 랜덤 접근 성능 및 작은 물리적 공간을 차지하는 특징으로 인해 휴대용 임베디드 기기에 널리 사용되고 있다. 낸드 플래시 메모리는 MP3 플레이어와 디지털 카메라 같은 휴대용 기기의 대용량 저장 장치로써 사용이 활성화 되고 있고, 이로 인해 점차 시장의 규모가 커지고 있다. 또한 낸드 플래시 메모리 기반의 SSD(Solid State Disk)는 데스크톱 컴퓨터의 저장 장치인 하드 디스크를 대체하려는 움직임을 보이고 있다[1,2].

플래시 메모리는 내구성으로 인해 산업 제어 시스템, 자동차 및 우주 왕복선 같은 높은 신뢰성을 요구하는 실시간 시스템에 적합하다. 또한 하드 디스크와는 달리 탐구 시간(seek time) 없이 원하는 데이터에 접근할 수

있기 때문에 입/출력 성능의 예측이 필요한 실시간 시스템에 적용하는 것이 유리하다.

하지만 플래시 메모리는 2가지 단점으로 인하여 DRAM을 사용할 때처럼 일관된 성능을 기대할 수는 없다. 이는 플래시 메모리를 실시간 시스템에 적용하기 위해 반드시 해결해야 하는 치명적인 장애요소라 할 수 있다. 플래시 메모리의 첫 번째 단점은 반드시 삭제된 상태에서만 데이터의 기록이 가능하다는 것이다. 두 번째 단점은 읽기, 쓰기 단위와 삭제의 단위가 다르다는 점이다. 플래시 메모리의 읽기와 쓰기는 페이지 단위로 수행하며 삭제는 블록 단위로 수행한다. 플래시 메모리의 블록은 다수의 페이지로 구성되며, 삼성전자의 대용량 페이지(large page) 기반 플래시 메모리의 경우 페이지의 크기는 2KB이며 블록은 총 64개의 페이지로 구성된다.

위에서 언급한 플래시 메모리의 특징으로 인해 플래시 변환 계층(FTL: Flash Translation Layer)이라는 시스템 소프트웨어가 필요하다. FTL은 파일 시스템의 논리적인 주소 영역과 플래시 메모리의 물리적인 주소 영역 간 매핑(mapping)을 수행한다. 플래시 메모리 매핑은 매핑 단위에 따라 블록 매핑과 페이지 매핑으로 구분된다. 블록 매핑을 사용하면 블록 단위의 데이터 갱신을 수행해야 한다. 따라서 블록 내 한 개의 페이지만 갱신이 된다고 해도 나머지 페이지는 새롭게 할당된 블록에 재 기록되어야 한다. 따라서 블록 매핑 방법을 사용할 경우 데이터 갱신에 따른 오버헤드가 크다.

페이지 매핑을 사용하면 데이터 기록 및 갱신을 빨리 수행할 수 있다. 예를 들어 논리 페이지 주소 100번지의 데이터 기록 요청이 발생하여 물리 페이지 주소 200번지에 데이터 기록을 했다고 가정한다. 이 후 논리 페이지 주소 100번지의 갱신이 발생하면 이전에 기록된 물리 페이지 주소 200번지의 데이터를 무효화(invalidation) 한 후 새로운 물리 페이지 주소 201번지에 갱신된 데이터를 기록한다. 따라서 페이지 매핑을 사용할 경우 갱신된 페이지만 새로운 페이지에 기록하면 된다.

이러한 데이터 갱신이 빈번하게 발생하게 되면 많은 수의 무효화된 페이지가 발생한다. 그러므로, 가비지 컬렉터(garbage collector)를 통해 무효화된 페이지를 소거하여 플래시 메모리의 공간을 확보해야 한다. 이 때 가비지 컬렉터는 블록 내 유효 페이지(valid page)만을 새로운 블록으로 복사한 후 해당 블록을 삭제한다. 플래시 메모리에서 수행하는 연산 중 가비지 컬렉션이 가장 큰 연산 비용을 소모 하며 해당 연산에 따라 플래시 메모리 응답 시간의 변동이 발생한다. 이러한 점을 고려하여 페이지 매핑을 위한 실시간 가비지 컬렉션 기법이 제안 되었다[3]. 하지만 페이지 매핑은 SSD와 같은 대용량 플래시 메모리를 사용하는 환경에서는 매핑 테이블의 크기가 매우 커진다는 단점이 있다.

따라서 최근 블록 매핑 기법의 장점과 페이지 매핑 기법의 장점을 혼합한 하이브리드 매핑(hybrid-level mapping) 기법에 대한 연구가 활발하게 진행되고 있다. 이 기법에서는 플래시 메모리의 물리 블록을 데이터 블록과 로그 블록으로 구분한다. 로그 블록은 로그 버퍼라고도 하며 하이브리드 매핑 기법을 사용하는 FTL을 로그 버퍼 기반 FTL(log buffer-based FTL)이라고 한다. 하이브리드 매핑 기법에서는 데이터 블록은 블록 매핑을, 로그 블록은 페이지 매핑을 통해 데이터를 관리한다. 하이브리드 매핑 기법에서 사용하는 로그 블록의 최대 개수는 유지 가능한 페이지 매핑 테이블의 최대 크기에 따라 결정된다. 파일 시스템에서 데이터 갱신이 발생하면 데이터 블록에 있는 기존의 데이터는 무효화된 후 로그 블록에 갱신된 데이터가 기록된다. 만약 모든 로그 블록에 데이터가 기록되어 더 이상 기록할 수 없다면 로그 블록 중 하나를 선택하여 유효한 데이터들을 새로운 데이터 블록으로 복사한 후 해당 로그 블록을 삭제한다. 이 과정을 로그 블록 병합이라 하며, 로그 블록 병합은 하나의 로그 블록에 대해서만 수행하기 때문에 페이지 매핑에서 수행하는 가비지 컬렉션에 비해 오버헤드가 작다. 따라서 로그 블록 기반의 플래시 변환 계층은 실시간 시스템에 적합하다고 할 수 있다. 또한 페이지 매핑 기법에 비해 작은 크기의 매핑 테이블을 사용한다는 장점이 있다.

하지만 하이브리드 매핑 기법은 2장에서 설명할 로그 블록 연관성에 따라 병합 연산 수행 시간이 달라질 수 있으며, 최악의 경우 병합 연산을 위해 큰 지연 시간이 발생할 수 있다. 시간 임수 시스템(time-critical system) 환경에서는 시스템의 활용률을 높이기 위해 최대 지연 시간을 감소시키는 것이 중요하다. 현재까지 플래시 메모리의 전체 I/O 수행 시간을 감소시키기 위해 하이브리드 매핑 기법 개선에 관한 많은 연구가 진행되었다[4-8]. 하지만 플래시 메모리를 사용하면서 성능상의 변동을 최소화 하는 기법에 대한 연구는 미비하다. 실 시간 시스템에서는 해당 작업의 완료를 예측하여 정해진 시간 내 작업을 완료할 수 있어야 한다.

본 논문에서는 이러한 점을 고려하여 플래시 메모리 사용 시 성능의 변동을 최소화 할 수 있는 매핑 기법을 제안한다. KAST라 불리는 제안 기법은 사용자가 플래시 메모리 사용 시 발생할 수 있는 최대 지연 시간을 제한할 수 있다. 본 제안 기법을 이용하여 사용자는 낸드 플래시 메모리 상에서 I/O 수행 시 가장 큰 오버헤드를 갖는 정체 시간을 제한할 수 있으며, 이에 따라 작업 완료 시간을 예측해야 하는 실 시간 시스템에 적용 가능하다. 또한 기존 매핑 기법과 비교하여 더 좋은 성

능을 보인다.

본 논문의 구성은 다음과 같다. 2장에서는 본 논문과 연관된 기존 기법에 대해 설명한다. 3장에서는 KAST의 전체 구조를 보인다. 4장에서 실험 결과를 통해 본 제안 기법과 기존 기법과의 비교를 수행한다. 마지막으로 5장에서 결론을 맺는다.

2. 관련연구

현재까지 로그 버퍼 기반 FTL에 대한 많은 연구가 진행되었다. 로그 버퍼 기반 FTL은 로그 블록 연관성에 따라 BAST(Block Associative Sector Translation), FAST(Fully Associative Sector Translation), SAST(Set Associative Sector Translation)로 구분할 수 있다(그림 1). 로그 블록 연관성이란 하나의 로그 블록이 얼마나 많은 데이터 블록과 연관되어 있는가를 나타낸다. BAST 기법에서는 로그 블록이 하나의 데이터 블록과 연관될 수 있다. FAST 기법에서 로그 블록은 모든 데이터 블록의 갱신된 데이터를 저장할 수 있다. 즉, 1개의 로그 블록이 모든 데이터 블록과 연관될 수 있다. SAST에서는 데이터 블록 그룹과 로그 블록 그룹 간 연관 관계를 갖게 된다. 위에서 언급한 세 가지 기법은 캐시 메모리에서 사용하는 직접 연관된(direct-mapped) 캐시, 완전 연관된(fully-associative) 캐시 그리고 세트 연관된(set-associative) 캐시 기법과 유사하게 동작한다. 그림 1에서 BAST, FAST, SAST 기법의 동작을 보인다.

그림 1(a)에서 전체 플래시 메모리 영역은 6개의 데이터 블록(B0~B5)과 4개의 로그 블록(L0~L3)으로 구성되며 각 블록은 4개의 페이지로 구성되어 있다. 데이터 블록에서 데이터 갱신이 발생하면 데이터 블록에 기록된 이전 페이지를 무효화 한 후 로그 블록에 기록한다. 논리 로그 블록 번호(LLBN: Logical Log Block Number) L0, L1, L2, L3로 지정된 각 로그 블록은 논리 블록 번호(LBN: Logical Block Number) B0, B1, B4, B5로 지정된 데이터 블록의 데이터 갱신을 담당한다. 논리 페이지 번호(LPN: Logical Page Number) p16으로 지정된 데이터는 데이터 블록 B4에 포함되기 때문에 로그 블록 L0과 B4는 연관 관계를 갖게 된다. 이 후 B2 또는 B3에 포함된 데이터의 갱신이 발생하면 로그 블록 중 하나를 선택하고 이와 연관된 데이터 블록과 병합 연산을 수행한 후 새로운 로그 블록을 할당한다. 병합 연산은 완전 병합(full merge), 부분 병합(partial merge) 그리고 스위치 병합(switch merge) 3가지로 구분된다[4]. 부분 병합 및 스위치 병합은 갱신된 데이터가 로그 블록의 첫 번째 오프셋부터 순차적으로 기록되어야 가능한 연산이다.

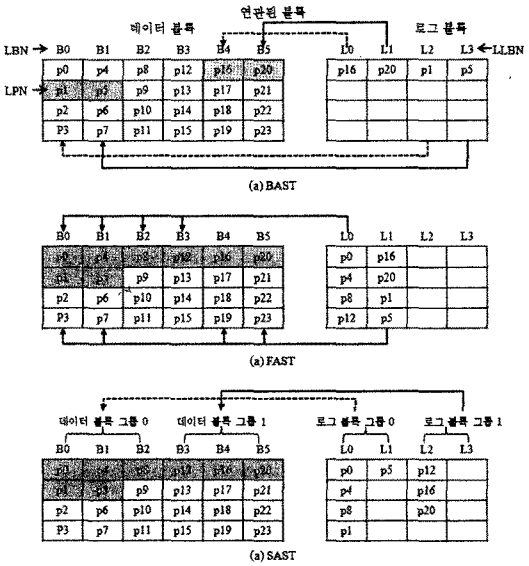


그림 1 로그 버퍼 기반 FTL 기법

BAST 기법에서는 임의 접근 갱신 데이터가 많이 발생한다면 로그 블록의 병합 연산을 자주 수행해야 하며 이에 따라 전체적인 성능 저하가 발생한다. BAST 기법에서 발생 가능한 이러한 문제점을 블록 쓰레싱(block thrashing)이라 한다. 그림 1(a)는 "p0, p4, p8, p12, p16, p20, p1, p5"의 순서대로 데이터 갱신이 발생했을 때의 블록들의 상태를 보여주고 있다. p16부터 갱신 요청 발생 시 매번 로그 블록을 병합한 후 새로운 로그 블록에 갱신 데이터를 기록해야 한다. 또한, 로그 블록에는 유효한 페이지 1개만 존재한 상태로 병합 연산을 수행해야 하며 나머지 3개의 페이지는 낭비된다. 따라서 BAST 기법에서 로그 블록의 사용률(utilization)이 매우 낮아짐을 알 수 있다.

위에서 언급한 BAST 기법의 문제점을 해결하기 위해 FAST 기법이 제안되었다. FAST 기법에서는 1개의 로그 블록에 어떠한 데이터 블록의 데이터도 기록 가능하며, 이에 따라 BAST 기법에 비해 블록 병합의 수가 감소하게 된다(그림 1(b)). 갱신된 페이지는 데이터 블록 번호와는 관계없이 갱신 요청이 발생된 순서대로 로그 블록에 기록된다. 따라서 FAST 기법을 사용하면 블록 쓰레싱 문제를 예방할 수 있다. FAST 기법에서는 "p0, p4, p8, p12, p16, p20, p1, p5"의 순서대로 데이터 갱신이 발생하여도 BAST와는 달리 로그 블록의 병합을 수행하지 않아도 된다. 또한 FAST 기법에서는 순차적인 데이터 갱신을 위해 1개의 SLB(Sequential Log Block)를 사용한다. SLB는 첫 번째 페이지에 오프셋 0을 갖는 데이터부터 순차적으로 기록된 로그 블록이다. 따라서 오직 하나의 데이터 블록과 연관되며, 따라서 부

분 병합 및 스위치 병합만을 수행하게 된다. RLB(Random Log Block)은 SLB와는 달리 임의의 순서대로 기록된 로그 블록을 말한다.

하지만 FAST 기법에서는 로그 블록이 높은 연관성(associativity)을 가질 수 있다는 문제점이 있다. 만약 로그 블록 L0를 병합 해야 하는 경우 자신과 연관된 모든 데이터 블록(B0, B1, B2, B3)과 병합해야 한다. 따라서 16번의 페이지 복사 연산을 수행해야 한다(16 페이지 = 4개 블록 * 4개 페이지). 결국 FAST는 병합 연산 수행 시 높은 병합 연산 비용을 발생시킬 수 있는 문제가 있다. 표 1에서 본 논문에서 사용하는 변수들을 보인다.

표 1 논문에서 사용하는 변수

순번	변수	설명
1	L	로그 블록
2	$A(L)$	해당 로그 블록과 연관된 데이터 블록
3	$k(L)$	로그 블록이 가진 연관성
4	N	로그 블록 내 페이지 개수
5	C_{copy}	페이지 복사 연산 값
6	C_{erase}	블록 삭제 연산 값

표 1의 변수에 따라 로그블록 L의 병합 연산 비용은 다음과 같다[6]:

$$C_{merge} = N \cdot k(L) \cdot C_{copy} + (k(L) + 1) \cdot C_{erase}$$

$k(L)$의 최대값은 블록 내 페이지 개수가 되며 따라서 가장 큰 병합 연산 비용은 $N^2 \cdot C_{copy} + (N+1) \cdot C_{erase}$가 된다. MLC(Multi-Level Cell) 낸드 플래시 메모리의 경우 블록당 페이지 개수는 128개이며 페이지 읽기, 쓰기 및 블록 삭제 비용은 각각 50μs, 650μs 그리고 2ms가 소요된다. 따라서 병합 연산의 최대값은 약 6초가 소요되는 반면, 최소값은 스위치 병합 연산 비용인 약 2ms가 된다. 따라서 병합 연산 비용의 최대값은 최소값에 비해 약 3,000배가 된다.

결국 BAST 기법은 블록 쓰레싱으로 인해 성능이 저하되며 FAST 기법은 로그 블록의 연관성에 따라 최대, 최소 병합 연산 비용의 차이가 매우 커질 수 있다. 따라서 두 기법 모두 실시간 시스템에는 적합하지 않다. 본 논문에서 제안하는 K-연관성 매핑을 이용한 FTL은 FAST와 유사하게 동작하지만 사용자가 결정한 K에 따라 최대 병합 시간을 제어할 수 있는 특징이 있다.

SAST[8] 기법에서는 1개의 로그 블록 그룹이 K개의 데이터 블록으로 구성된 1개의 데이터 블록 그룹과 연관된다. 그림 1(c)에서 2:3으로 구성된 SAST 매핑 기법을 보인다. SAST는 기존의 BAST와 FAST 기법을 절충한 매핑 기법이지만 블록 쓰레싱과 높은 블록 연관성 문제 모두 완전히 해결하지 못한다는 단점이 있다. 만약 여러 데이터 블록 그룹 내 데이터가 번갈아 가며 갱신되는 경우 로그 블록 쓰레싱 문제가 발생하게 된다.

3. K-연관성 매핑 기법

본 장에서는 본 논문의 제안 기법인 KAST에 대해 설명한다.

3.1 K-연관성 매핑 기법 흐름도

KAST 기법을 설명하기 위해 그림 2에서 KAST 전체 동작의 흐름도(flow chart)를 보인다.

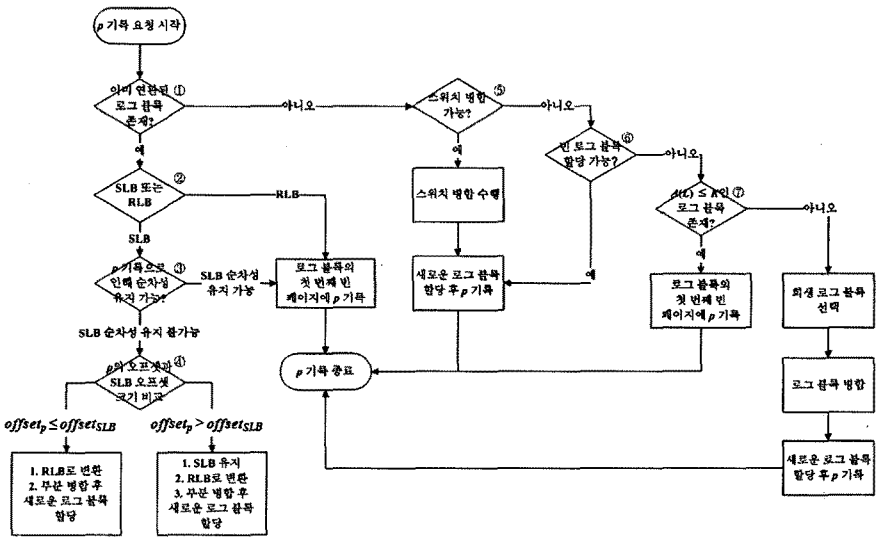


그림 2 KAST 흐름도(flow chart)

KAST에서 페이지 p의 기록 요청이 발생하면 이전에 연관된 로그 블록이 존재하는지 확인하고 ①, 존재한다면 해당 로그 블록이 RLB 또는 SLB인지 확인한다 ②. 만약 RLB라면 첫 번째 빈 페이지에 p를 기록하고, SLB라면 p기록으로 인해 순차성을 유지할 수 있는지 확인한다 ③. 순차성을 유지할 수 있다면 SLB의 첫 번째 빈 페이지에 p를 기록하고 순차성을 유지할 수 없다면 세부 알고리즘에 따라 다른 동작을 수행한다 ④.

만약 이전에 연관된 로그 블록이 존재하지 않는다면 우선 스위치 병합 가능한 로그 블록이 있는지 확인하고 ⑤, 존재 한다면 스위치 병합 후 새로운 로그 블록에 p를 기록한다. 스위치 병합이 불가능하면 연관성을 증가시키는 데이터 기록을 시도한다 ⑥. 만약 연관성을 증가시키는 데이터 기록을 할 수 없다면 희생 로그 블록을 선택하여 병합 연산 후 새로운 로그 블록에 p를 기록한다. KAST 세부 동작은 3.2~3.3절에서 설명한다.

3.2 K-연관성 매핑 기법 전체 구조

KAST 기법은 크게 3가지 기능을 갖는다.

- 로그 블록의 연관성을 최소화하기 위해 다른 LBN을 갖는 페이지를 여러 로그 블록에 분산 시켜 기록한다.
- 각 로그 블록을 최대 K개의 데이터 블록들과 연관되도록 한다. 따라서 $k(L) \leq K$를 보장하며 K를 제한함에 따라 병합 연산에 소요되는 최대 시간을 제어할 수 있다.
- FAST에서는 SLB를 1개만 사용하지만 KAST에서는 SLB를 2개 이상 사용 할 수 있다. 또한 순차적인 데이터의 갱신 요청이 발생하지 않게 되면 SLB를 RLB로 전환하여 사용할 수 있다.

SAST가 데이터 블록 그룹 내 블록 개수를 K보다 작게 유지하여 로그 블록의 연관성을 유지하는 반면에 KAST는 데이터 블록과 로그 블록을 그룹으로 구성하지 않는다. 앞에서 언급한 것처럼 데이터 블록과 로그 블록을 그룹으로 구성하면 블록 쓰레싱 문제가 발생할 수 있다. KAST는 각 로그 블록의 최대 연관성을 K로 제한한다. 또한 최소한의 $k(L)$을 유지하기 위해 다른 LBN을 갖는 데이터를 동일한 로그 블록에 최대한 기록하지 않도록 한다. 그림 3에서 FAST, SAST, KAST의 로그 블록 기록을 비교한다.

"p1, p5, p9, p13, p17, p21, p2, p6"의 순서대로 페이지 기록 요청이 발생한 경우 FAST는 첫 번째 로그 블록부터 순서대로 데이터를 기록하며, SAST는 해당 데이터 블록 그룹과 연관된 로그 블록 그룹에 순서대로 데이터를 기록한다. 하지만 KAST는 갱신 요청 데이터를 각 로그 블록에 분산시켜 기록한다. 먼저 p1, p5, p9, p13를 각각 다른 로그 블록에 기록한다. 데이터 기록 후 L0~L3은 연관성 1을 갖게 된다. 이 후 p17과 p21

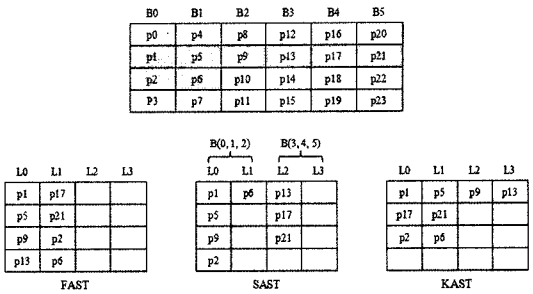


그림 3 FTL 매핑 기법에 따른 데이터 기록 비교

을 각각 L0과 L1에 기록하고 이에 따라 L0, L1의 연관성은 1에서 2로 증가한다. 마지막으로 연관성을 증가시키지 않기 위해 해당 데이터가 포함된 데이터 블록과 이미 연관된 로그 블록이 있는지 확인한다. 이에 따라 p2과 p6를 L0과 L1에 기록하며, 따라서 L0, L1의 연관성은 2로 유지된다.

그림 3에서 보이는 것처럼 KAST는 가능하다면 $k(L)$을 증가시키지 않은 기록을 우선적으로 수행한다. KAST는 페이지 기록 요청이 왔을 때 빈 로그 블록이 없는 상태 그리고 요청된 LBN에 연관된 로그 블록이 없는 경우에만 로그 블록의 연관성을 증가시키며 페이지를 기록한다. KAST에서 SLB는 순차적인 데이터를 기록하는 로그 블록으로 연관성 1을 갖는다. 처음 기록 요청이 발생하여 새로운 로그 블록을 할당할 때 SLB인지 RLB인지 결정하게 된다.

그림 4에서 KAST에서 사용하는 로그 블록의 상태 전이도(state transition diagram)를 보인다.

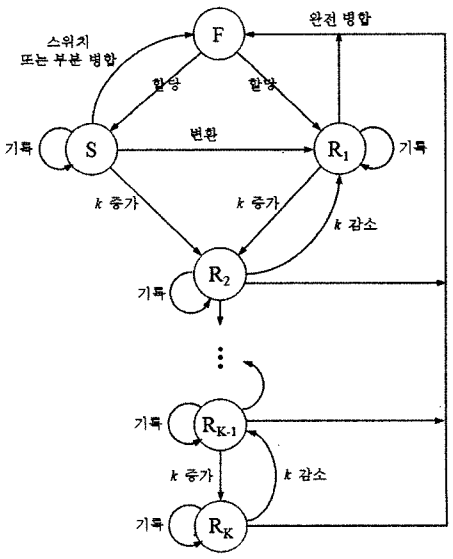


그림 4 KAST 로그 블록 상태 전이도

F는 빈 로그 블록(Free log block), S는 순차 로그 블록(SLB: Sequential Log Block) 그리고 R_i는 연관성 i를 갖는 랜덤 로그 블록(RLB: Random Log Block)이다. 처음 모든 로그 블록은 데이터가 기록되어 있지 않은 빈 블록 상태를 갖는다(F 상태). 빈 블록은 SLB(S 상태)또는 RLB(R_i)로 전이 가능하다. 만약 R_i 상태를 갖는 RLB에 $d(p) \in A(L)$ ($d(p)$는 페이지 p가 속하는 LBN을 의미한다)인 데이터를 기록한다면 로그 블록 L은 현재 상태 R_i를 유지할 수 있다. 또한 SLB의 순차성을 무시하지 않고 데이터를 기록한다면 SLB는 S 상태를 유지할 수 있다. 하지만 $d(p) \notin A(L)$인 페이지를 RLB L에 기록한다면 R_i는 R_{i+1}로 전이된다. 이 후 R_{i+1}인 L은 자신이 가지고 있는 페이지의 무효화를 통해 R_i로 전이될 수도 있다. SLB는 해당 블록 내 빈 페이지의 개수에 따라 R_1또는 R_2 상태로 전이 가능하다. 이 후 SLB는 부분 병합(partial merge) 및 스위치 병합(switch merge)을, RLB는 완전 병합(full merge)을 수행하여 F 상태가 된다. 완전 병합 시 R_i 상태를 갖는 RLB의 병합 연산 비용은 $N \cdot i \cdot C_{copy} + (i+1) \cdot C_{erase}$가 된다.

3.3 로그 버퍼 데이터 기록

본 절에서는 KAST 기법의 세부 동작을 설명한다.

KAST에서 페이지 p를 로그 블록에 기록하는 과정은 $d(p) \in A(L)$인 RLB L이 존재하는 경우와 그렇지 않은 경우로 구분된다. 표 2에서 p 기록 과정을 설명하기 위한 6가지 임계 값을 보인다.

표 2 KAST에서 사용하는 6가지 임계값

임계 값	설명
K	로그 블록이 가질 수 있는 최대 연관성
θ_{FP1}	SLB의 RLB(R_1) 변환 기준치(빈 페이지 개수)
θ_{FP2}	SLB의 RLB(R_2) 변환 기준치(빈 페이지 개수)
θ_{FP3}	SLB의 부분 병합 기준치(빈 페이지 개수)
θ_{gap}	SLB 유지 기준치(오프셋 간격)
θ_{SLB}	할당 가능한 최대 SLB 개수

3.3.1 $d(p) \in A(L)$인 로그블록 L이 존재하는 경우

$d(p) \in A(L)$인 로그블록 L이 존재 존재한다면 $k(L)$의 증가 없이 L에 p를 기록 가능하다. 이때 p의 오프셋이 0이라면 새로운 로그 블록의 첫 번째 페이지에 p를 기록하여 SLB로 할당할 수 있다. 만약 p의 오프셋이 0이면서 $d(p) \in A(L)$인 RLB L이 존재하는 경우 2가지 동작 중 한가지를 선택할 수 있다(그림 5).

첫 번째 방법은 p를 RLB L에 기록하는 것이며(그림 5(b)), 두 번째 방법은 새로운 로그 블록의 첫 번째 페이지에 기록하여 SLB로 할당하는 것이다. 이 때 $p' \in L$ 이면서 $d(p') = d(p)$에 해당하는 p'가 존재하는 경우 갱신 요청이 발생할 때까지 아무런 동작도 수행하지 않거나(그림 5(c)) 이전에 기록되어 있는 L에서 SLB로 p' 복사 후 무효화 할 수 있다(그림 5(d)). 이 후 SLB의 순차성을 유지하기 위해 중간에 위치한 페이지를 데이터 블록 $d(p)$에서 SLB로 복사하여 채운다.

SLB S가 있고 $d(p) \in A(S)$인 경우, p를 S에 기록해야 하는데 만약 p의 기록으로 인해 S의 순차성을 유지 못하게 되는 경우에는 아래와 같이 크게 2가지 동작을 수행한다(그림 6).

먼저 p의 오프셋이 S의 첫 번째 빈 페이지의 오프셋보다 작은 경우($offset_p \le offset_S$) p는 S에 이미 기록되어 있는 데이터를 갱신하게 되며, 이 때 S의 빈 페이지 수(FP_{SLB})를 확인한다(그림 6(a)). 만약 θ_{FP1}보다 FP_{SLB}가 크다면($\theta_{FP1} < FP_{SLB}$), S에 p를 기록한 후 S를 R_1 상태를 갖는 RLB로 변환한다(그림 6(b)). 그렇지 않다면 ($\theta_{FP1} \ge FP_{SLB}$), S를 부분 병합한 후 새로운 로그 블록을 할당하여 첫 번째 오프셋에 p를 기록한다(그림 6(c)).

페이지 p의 기록이 S에 이미 기록된 데이터의 갱신이 아니라면($offset_p > offset_S$) p의 오프셋($offset_p$)과 S의 첫 번째 빈 페이지의 오프셋($offset_s$) 간격의 크기($offset_{gap} = offset_p - offset_s$), 그리고 FP_{SLB}에 따라 다음 3가지 중 한가지 동작을 선택한다(그림 6(d)). 만약 θ_{gap}보다 $offset_{gap}$이 작거나 같다면($\theta_{gap} \ge offset_{gap}$) 두 오프셋

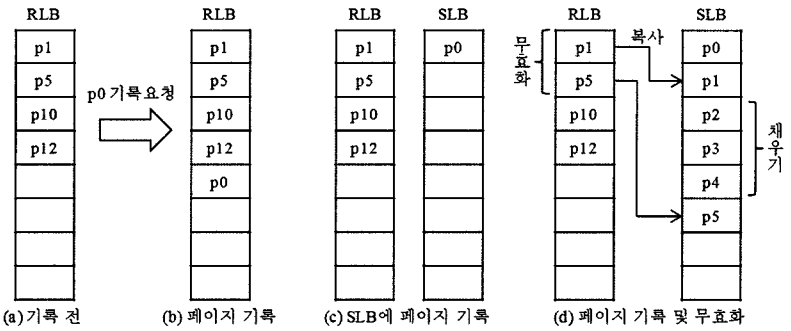


그림 5 연관성 증가시키지 않는 페이지 기록

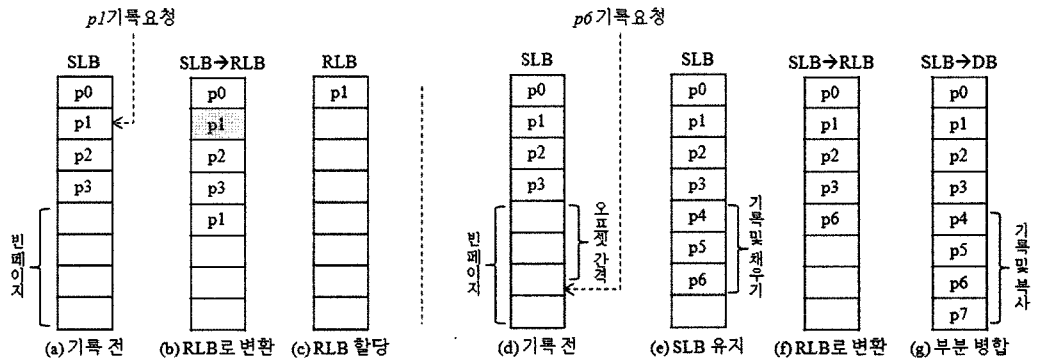


그림 6 SLB의 순차성을 무시하는 페이지 기록

사이에 위치한 페이지를 데이터 블록에서 S로 복사한 후 p를 기록하여 S의 현재 상태를 유지한다(그림 6(e)). θ_{gap}보다 $offset_{gap}$이 크다면($\theta_{gap} < offset_{gap}$) FP_{SLB}를 확인한다. 이 때 θ_{FP1}보다 FP_{SLB}가 크다면, ($\theta_{FP1} < FP_{SLB}$) S에 p를 기록한 후 S를 R_1 상태를 갖는 RLB로 변환하고 (그림 6(f)) 그렇지 않다면($\theta_{FP1} \geq FP_{SLB}$) S에 p를 기록하면서 S를 부분 병합한다(그림 6(g)).

앞에서 언급한 것처럼 SLB에 빈 페이지가 많은 경우에는 SLB를 R_1 상태를 갖는 RLB로 변환하게 되는데 이것은 해당 SLB에 앞으로 순차적인 페이지 기록 요청이 작은 확률로 발생할 것이라 판단하기 때문이다. 또한 SLB에 빈 페이지가 많을 경우에 SLB를 부분 병합 하는 것 보다는 RLB로 변환하여 빈 공간을 최대한 활용하는 것이 더 효율적이기 때문이다.

3.3.2 $d(p) \in A(L)$인 L이 존재하지 않는 경우

$d(p) \in A(L)$에 해당하는 로그블록이 존재하지 않는 경우 로그 블록의 연관성을 증가시키는 페이지 기록을 수행해야 한다. 먼저 스위치 병합이 가능한 SLB가 존재하는지 확인한다. 만약 존재한다면 해당 SLB와 연관된 데이터 블록을 삭제한 후 SLB를 데이터 블록으로 교체한다(스위치 병합). 스위치 병합은 복사 연산이 필요 없기 때문에 다른 로그 블록에 페이지를 기록하여 연관성을 증가시키는 것 보다 효율적이다.

만약 스위치 병합 가능한 SLB가 존재하지 않으면 로그 블록 중 연관성 증가 가능한 RLB L을 선택한다. 이 때 연관성이 최소인 RLB L을 선택하게 되며 만약 최소 연관성이 동일한 L이 2개 이상이라면 빈 페이지가 많은 L을 선택한다. 만약 빈 페이지 개수도 같다면 LRU인 L을 선택한다. 이에 따라 로그 블록들의 연관성 및 빈 페이지 개수를 균형 있게 유지하게 된다. 이 후 RLB L의 $k(L)$이 K보다 작을 때까지 $k(L)$을 증가시키며 L에 페이지를 기록할 수 있다.

위의 과정을 수행하는 동안 $d(p) \notin A(S)$인 SLB L이 존재하더라도 빈 페이지가 많거나 오랜 시간 동안 페이지 기록이 없었다면 L의 순차성을 무시하는 페이지 기록을 수행할 수 있다. 이때 θ_{FP2}보다 큰 수의 빈 페이지를 갖는 SLB L이 존재 하는지 확인하고 존재한다면 L에 p를 기록한다. 이 후 SLB L은 R_2 상태를 갖는 RLB로 변환된다. 현재 할당된 로그 블록들 중 빈 페이지가 없거나 K보다 연관성이 작은 로그 블록이 존재하지 않는다면 어떠한 로그 블록에도 데이터를 기록할 수 없다. 이 때 기존에 할당된 로그 블록 중 하나를 선택한 후 병합하여 새로운 로그 블록을 생성해야 한다.

병합 대상 로그 블록을 선택하는 과정은 다음과 같다. 첫 번째로 부분 병합 가능한 SLB를 선택한다. 만약 θ_{FP3}보다 작은 수의 빈 페이지를 갖는 SLB L이 있다면 부분 병합의 대상이 되고 그렇지 않다면 현재 상태를 유지한다. SLB를 부분 병합하지 않을 경우에 S 상태를 유지하여 다음 번 기록 요청이 발생한 경우 사용한다. 두 번째로 병합 연산 비용과 빈 페이지의 개수가 최소인 RLB를 병합 대상으로 선정하는데, 병합 연산의 비용의 크기는 로그 블록의 연관성에 따라 달라진다. 병합 대상 로그 블록이 선택되면 로그 블록을 병합하여 빈 로그 블록 1개를 생성한다.

병합을 통해 확보한 새로운 로그 블록을 사용할 때, 만약 현재 기록 요청이 발생한 페이지의 오프셋이 0이라면 새로운 로그 블록의 첫 번째 오프셋에 기록하여 SLB로 사용한다. KAST에서 SLB의 개수는 FAST와는 달리 0개부터 최대 로그 블록의 개수만큼 할당 가능하며, SLB의 최대 할당 가능 개수는 사용자가 설정한 θ_{SLB}가 된다.

4. 성능 평가

본 논문의 제안 기법인 KAST의 성능 평가 및 검증을 위해 트레이스(trace) 기반의 실험을 수행하였다. 실험을 위해 FTL 시뮬레이션을 위한 프로그램을 작성하

표 3 실험에 사용한 낸드 플래시 모델 사양

항목	세부 사항
낸드 플래시 모델	SLC(Single Level Cell)
페이지 크기	2KB
블록 내 페이지 개수	64
페이지 읽기 속도	25 us
페이지 기록 속도	200 us
블록 삭제 속도	2 ms

였으며 이를 통해 KAST를 BAST, FAST 및 SAST 기법과 비교하였다. 시뮬레이션에 적용한 낸드 플래시 모델의 사양은 표 3과 같다.

4.1 로그 블록 개수 변화에 따른 성능 변화 비교

본 실험에서는 로그 블록 개수 변화에 따른 로그 블록의 사용률(utilization)과 I/O 수행 시간을 비교하였다. 로그 블록의 사용률은 로그 블록 병합 시 로그 블록에 기록되어 있었던 페이지의 비율을 말한다. 실험에 사용한 트레이스는 총 4가지로 이중 2가지는 실제 PC 상에서 응용 프로그램을 동작하여 추출한 것이며 나머지 2가지는 bonnie[9], postmark[10] 벤치 마크 프로그램을 이용하여 추출하였다(표 4).

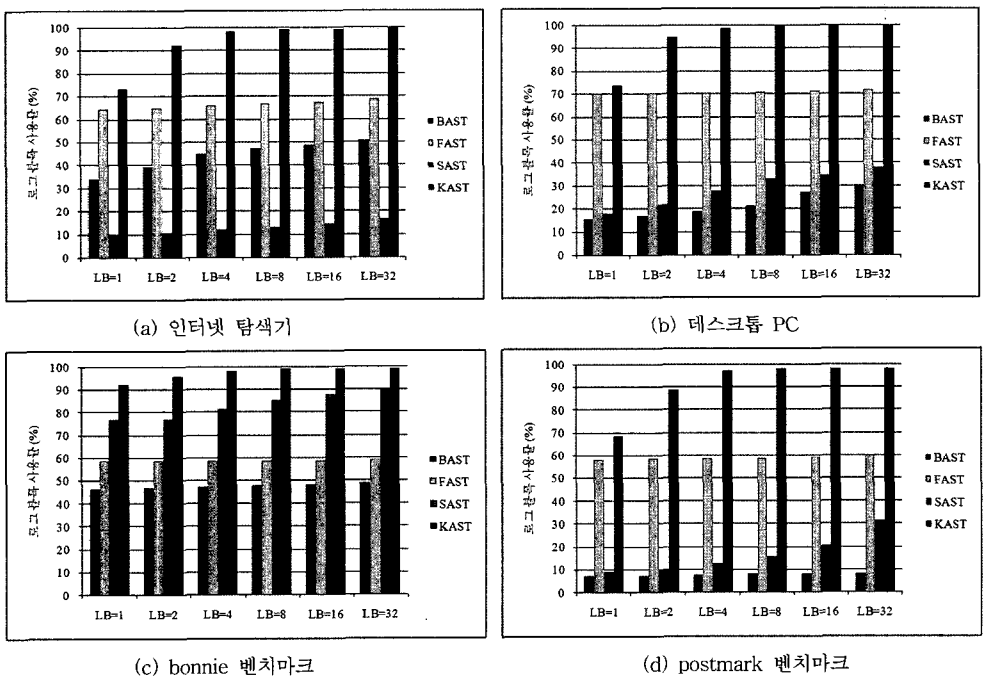
그림 7에서 각 FTL 매핑 기법의 로그 블록 사용률을 보인다. 실험을 위해 로그 블록의 개수는 1개부터 32개까지 2배씩 증가시켰으며 SAST, KAST에서 K는 모두 64로 설정하였다.

그림 7을 통해 KAST의 로그 블록 활용률이 가장 높은 것을 알 수 있다. BAST와 SAST에서는 로그 블록 쓰레싱 문제가 발생하며 FAST에서는 SLB의 블록 쓰레싱 문제가 발생한다. 따라서 로그 블록의 사용률이 낮아지게 된다. KAST에서는 순차 접근 데이터의 기록 요청이 발생하면 최대 SLB 개수 범위 내에서 SLB를 할당하고 이 후 해당 데이터의 패턴이 임의 접근으로 변경되면 SLB를 RLB로 변경하여 로그 블록에 데이터를 기록한다. RLB로 변경된 로그 블록은 연관성이 K보다 작을 때까지 페이지 기록을 수행할 수 있다. 따라서 로그 블록의 활용률이 다른 기법에 비해 높다.

그림 8에서 로그 블록 개수 변경에 따른 각 FTL 매

표 4 실험에 사용한 트레이스

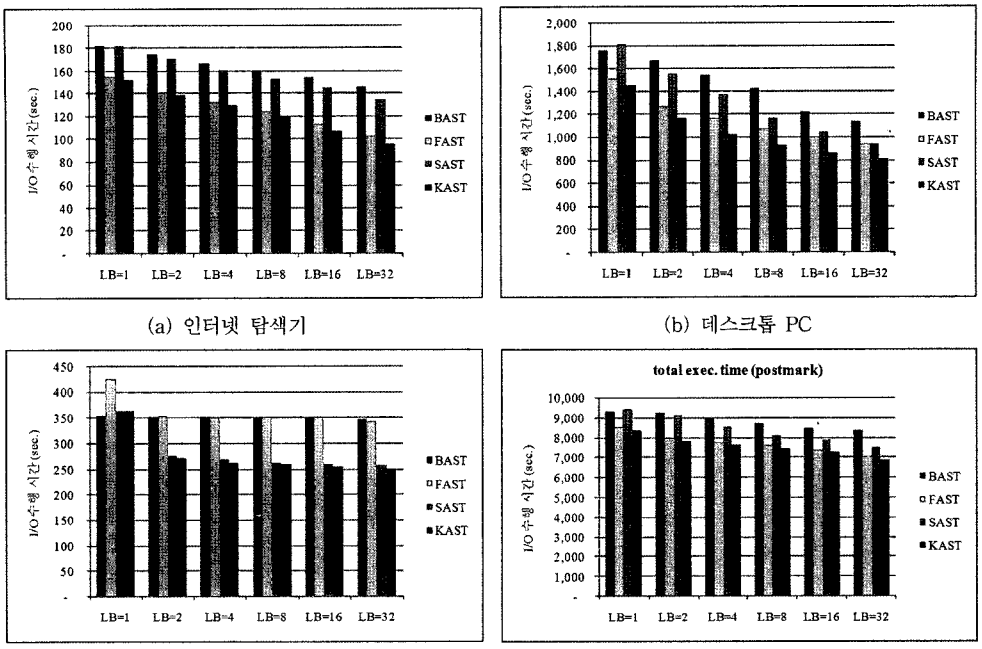
트레이스	트레이스 설명	쓰기 발생 횟수
인터넷 탐색기	인터넷 탐색기 사용시 발생한 디스크 I/O 추출	168,340
테스크톱 PC	윈도우 XP 기반 PC 사용시 발생한 디스크 I/O 추출	1,614,175
bonnie 벤치 마크	bonnie 벤치 마크 프로그램을 이용한 디스크 I/O 추출	480,189
postmark 벤치 마크	Postmark 벤치 마크 프로그램을 이용한 디스크 I/O 추출	2,514,990



(a) 인터넷 탐색기 (b) 데스크톱 PC

(c) bonnie 벤치마크 (d) postmark 벤치마크

그림 7 로그 블록 사용률



(a) 인터넷 탐색기 (b) 데스크톱 PC

(c) bonnie 벤치마크 (d) postmark 벤치마크

그림 8 전체 실행 시간

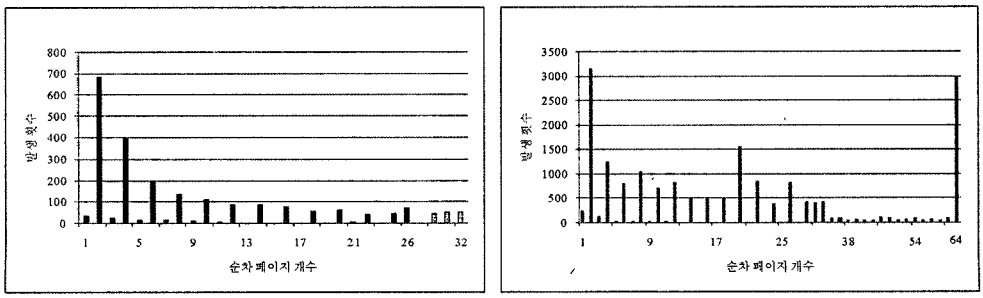
핑 기법의 I/O 수행 시간을 보인다.

이 결과를 통해 로그 블록의 활용률에 비례하여 I/O 수행 성능이 좋아지는 것을 알 수 있다. 로그 블록의 활용률이 높을수록 더 많은 데이터를 로그 블록에 기록할 수 있으며 이에 따라 로그 블록 쓰레싱 문제가 해결된다. 또한 병합 되기 전 로그 블록에 더 많은 데이터를 기록함으로써 로그 블록의 병합 연산의 수행을 지연시킬 수 있다. 로그 블록의 병합 연산을 지연시킴으로써 동일한 LSN을 갖는 데이터가 중복 기록될 확률이 높아진다. 이에 따라 이전에 로그 블록에 기록된 동일한 페이지를 무효화하게 되며 결국 병합 연산의 비용을 절감할 수 있다.

4.2 K 변화에 따른 성능 변화 비교

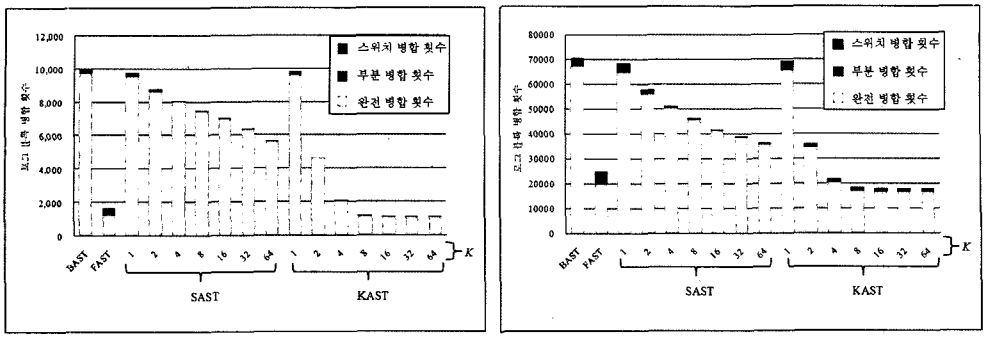
본 실험에서는 K의 증가에 따른 로그 블록 병합 횟수, 로그 블록 연관성 크기, 로그 블록 병합 연산 비용 및 I/O 수행 시간을 비교하였다. 실험에 사용한 트레이스는 인터넷 탐색기와 데스크톱 PC 사용시 추출한 것이며, 로그 블록의 개수를 32개로 설정하여 실험하였다. 또한 K는 1부터 64까지 증가시켰다. 우선, 2가지 트레이스의 패턴이 SLB 활용에 얼마나 적합한지를 비교하기 위해 오프셋 0을 갖는 페이지 기록 요청 이후에 입력된 순차적인 페이지의 개수를 측정하였다(그림 9).

인터넷 탐색기의 트레이스에서는 오프셋 0을 갖는 페이지 이후 순차적인 기록 요청 발생 횟수가 대부분 작



(a) 인터넷 탐색기 (b) 데스크톱 PC

그림 9 순차 페이지 발생 횟수



(a) 인터넷 탐색기 (b) 데스크톱 PC

그림 10 로그 블록 병합 횟수

으며 최대 연속 페이지 개수는 32개 이다. 하지만 데스크톱 PC 트레이스에서는 스위치 병합 가능한 연속 페이지 기록 요청이 (64개의 순차 페이지 요청) 약 3,000회 발생한다. 이에 따라 데스크톱 PC 트레이스가 인터넷 탐색기 트레이스에 비해 순차적인 데이터 패턴이 많다는 것을 알 수 있다.

그림 10에서 4가지 FTL 기법 상에서 얼마나 많은 수의 병합 연산이 발생하는지를 보인다.

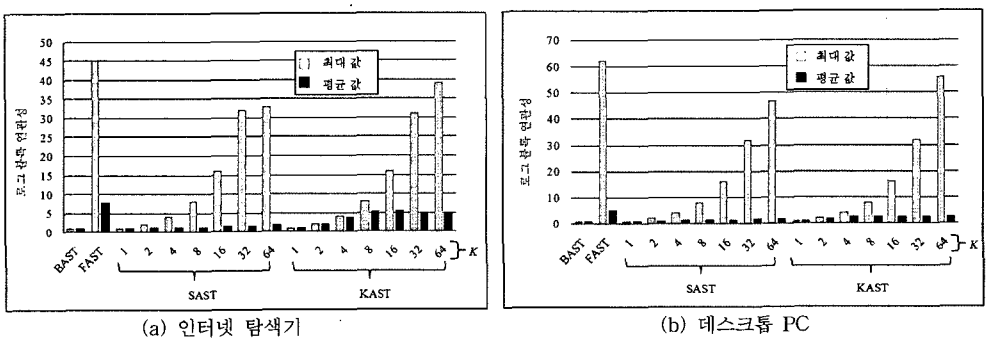
SAST와 KAST의 경우 로그 블록의 연관성인 K를 변경하여 실험하였으며 SAST에서는 데이터 블록 그룹과 로그 블록 그룹 내 블록의 개수를 동일하게 설정하였다. 실험 결과를 통해 FAST에서는 로그 블록 병합 횟수가 작은 반면 BAST에서는 블록 쓰레싱으로 인해 로그 블록 병합 횟수가 크다는 것을 알 수 있다. SAST와 KAST에서 K를 1로 설정하였을 경우 BAST와 유사한 횟수의 로그 블록 병합 횟수가 발생하였다. 하지만 K를 크게 설정하면서 병합 연산의 횟수가 감소하는 것을 알 수 있다. KAST는 K가 8 이상이 되면서 FAST보다 적은 수의 병합을 수행하지만 SAST는 대부분의 경우 FAST보다 더 많은 수의 병합 연산을 수행하였다. 이는 SAST가 블록 쓰레싱 문제를 해결하지 못하기 때문이다.

그림 11에서 로그 블록 병합 시 측정한 로그 블록의 연관성의 크기를 보인다.

로그 블록의 최대 병합 비용은 최대 연관성에 따라 결정된다. 실험 결과에 따라 SAST와 KAST의 로그 블록 연관성의 최대값이 K보다 작게 유지되는 것을 알 수 있다. FAST의 최대 로그 블록 연관성이 가장 크며 이에 따라 FAST의 최대 정체 시간이 길어질 수 있다. SAST와 KAST는 K를 변경함에 따라 최대 정체 시간을 제어할 수 있다. KAST는 BAST와 SAST에 비해 적은 수의 블록 병합을 수행하였으며 또한 병합 연산에 따른 비용 역시 FAST보다 작다.

그림 12에서 단일 로그 블록 병합에 따른 정체 시간 분포를, 그림 13에서 로그 블록 연관성 분포를 보인다.

실험 결과는 인터넷 탐색기 작업부하를 이용하여 측정하였으며 KAST와 SAST의 경우 K를 16으로 설정하였다. 그림 12에서 FAST에서 1회 병합 비용의 최대값은 548ms가 소요되었지만 KAST에서는 232ms가 소요되는 것을 알 수 있다. SAST는 병합 하기 전 로그 블록에 적은 수의 데이터가 남아있기 때문에 대부분의 경우 병합 비용이 KAST보다 작게 발생하였다. 하지만 SAST는



(a) 인터넷 탐색기 (b) 데스크톱 PC

그림 11 로그 블록 연관성 크기

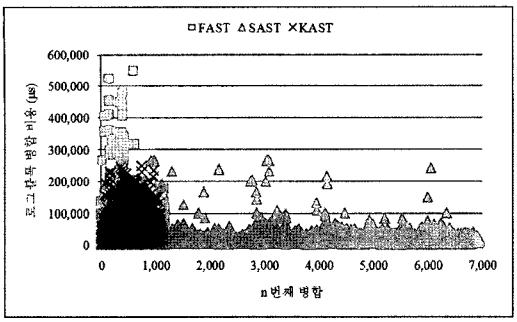


그림 12 로그 블록 병합 연산 비용 분포

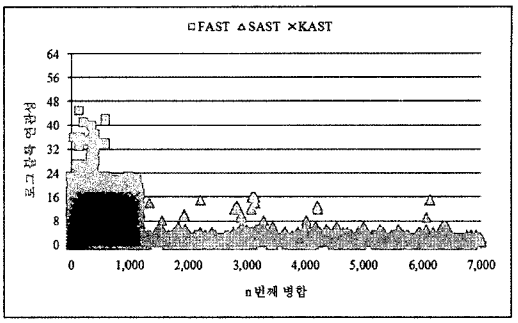


그림 13 로그 블록 연관성 분포

가장 많은 수의 병합 연산을 수행하였다(그림 12).

그림 13에서 KAST와 SAST의 경우 로그 블록 연관성은 최대값인 16으로 제한되는 것을 알 수 있다. SAST의 경우 가장 낮은 로그 블록 연관성을 갖지만 블록 쓰레싱으로 인해 가장 많은 병합 연산을 수행해야 한다. FAST의 경우 KAST와 비슷한 수의 병합 연산을 수행하지만 로그 블록 연관성의 제한이 없기 때문에 KAST보다 더 높은 연관성을 갖는다.

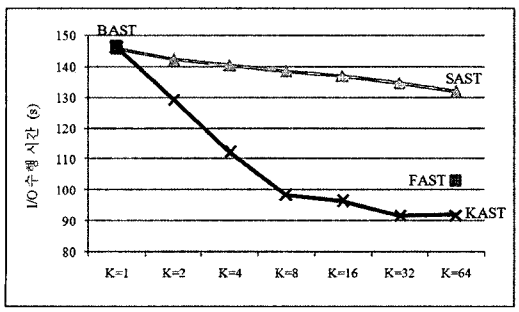
그림 14에서 각 FTL의 전체 I/O 수행 시간을 보인다. SAST와 KAST는 로그 블록 병합 수의 감소로 인해 K가 커질수록 전체 성능이 향상된다. 하지만 SAST는 인터넷 탐색기 트레이스의 임의 접근 데이터로 인해 블록 쓰레싱 문제가 발생하며 이에 따라 KAST보다 성능이 좋지 않았다. KAST는 K가 8일 때부터 SAST, FAST 두 기법 보다 높은 성능을 보였다.

4.3. θ_{FP1}의 변화에 따른 성능 변화 비교

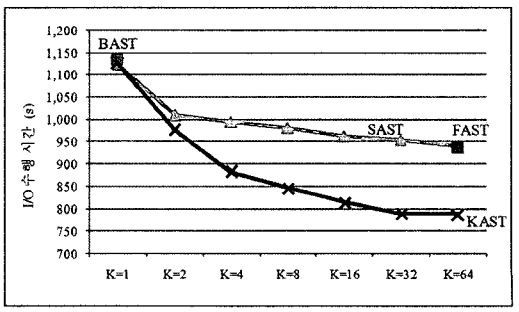
본 실험에서는 데스크톱 PC 트레이스를 사용하여 θ_{FP1}의 변화에 따른 성능을 측정하였다(그림 15). KAST에서는 SLB의 순차성을 무시하는 데이터 기록 요청 시 해당 SLB의 빈 페이지가 θ_{FP1}보다 작으면 부분 병합 후 새로운 로그 블록을 할당하고 그렇지 않은 경우 R_1 상태를 갖는 RLB로 변환한 후 다음 기록 요청을 수행한다. 따라서 θ_{FP1}을 작게 설정할수록 RLB로의 변환이 많아지고 크게 설정할수록 부분 병합을 많이 수행한다.

실험 결과에 따라 K가 1일 때는 θ_{FP1}를 크게 설정할수록 I/O 수행 성능이 조금 좋아지는 반면, K가 2이상이 되면 I/O 수행 성능이 θ_{FP1}이 커질수록 저하되는 것을 알 수 있다(그림 15(a)). 실험 결과를 설명하기 위해 K가 1과 64인 경우의 부분 병합과 완전 병합으로 인한 비용들을 비교하였다. θ_{FP1}을 크게 하여 부분 병합 횟수를 증가시키면 상대적으로 완전 병합의 수가 감소하여 완전 병합 비용이 감소하게 된다. K가 1일 때는 블록 쓰레싱으로 인해 K가 64일 때보다 더 많은 수의 완전 병합을 수행한다. 따라서 완전 병합 비용은 K가 1일 때 K가 64일 때보다 더 큰 폭으로 감소하게 된다(그림 15(b)). 하지만 θ_{FP1}가 커지면서 부분 병합의 횟수가 증가하기 때문에 부분 병합에 따른 비용은 증가하게 된다. 따라서 두 연산 비용의 합을 통해 K가 1일 때 θ_{FP1}가 커지면 I/O 수행 성능이 좋아지는 반면 K가 64일 때는 I/O 수행 성능이 저하된다.

K와 θ_{FP1}의 변경 외 θ_{FP2}, θ_{FP3}, θ_{gap}, θ_{SLB}의 변경 시 전체 성능에 어떠한 영향을 주는지에 대해 실험하였다. 하지만 해당 변수의 변화에 따른 전체 성능의 변화는 K와 θ_{FP1}의 변경에 따른 성능의 변화만큼 크지 않았으며,

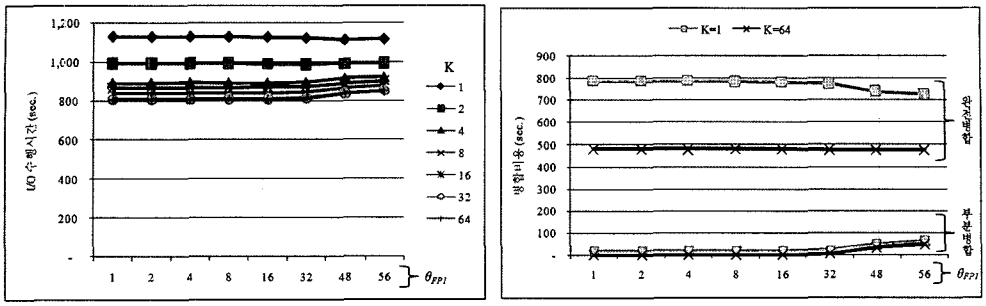


(a) 인터넷 탐색기



(b) 데스크톱 PC

그림 14 전체 I/O 수행 시간



(a) I/O 수행 시간 (b) 병합 연산 비용

그림 15 θ_{FP1}의 변화에 따른 성능 변화

이는 로그 블록에 데이터 기록 시 해당 변수가 적용되는 부분이 미비하기 때문이다. 따라서 실험 시 고정 된 값을 적용하였으며, θ_{FP2}와 θ_{FP3}는 8로 설정하였고 θ_{gap}과 θ_{SLB}의 값은 4로 설정하였다.

5. 결 론

본 논문에서는 실시간 시스템을 위한 로그 버퍼 기반 FTL 매핑 기법인 KAST를 제안하였다. 로그 버퍼를 사용하는 기존 FTL 매핑 기법으로는 BAST, FAST, SAST가 있다. 하지만 3가지 기법 모두 단점을 가지고 있는데, BAST는 블록 쓰레싱으로 인해 로그 블록의 사용률이 저하되며 이에 따라 로그 블록 병합 연산을 가장 많이 수행한다. 따라서 가장 좋지 않은 성능을 보인다. FAST는 BAST의 블록 쓰레싱 문제를 개선한 기법이지만 하나의 로그 블록이 많은 데이터 블록과 연관될 수 있는 문제점이 있다. 이에 따라 로그 블록 병합 연산의 최소값과 최대값의 차가 MLC 낸드 플래시의 경우 최대 3,000배까지 될 수 있다. SAST는 FAST의 높은 연관성 문제를 개선하고자 하였으나 블록 쓰레싱 문제를 해결하지는 못했다.

낸드 플래시 메모리를 실시간 시스템에 적용할 경우 낸드 플래시 메모리의 동작에 따른 성능상의 변동을 최소화 해야 하며 병합 연산의 종료 시간을 예측할 수 있어야 한다. 로그 버퍼를 사용하는 FTL의 경우 로그 블록의 병합 연산이 가장 큰 정체 시간이 되며, 또한 로그 블록 병합 연산 비용은 해당 로그 블록의 연관성에 따라 달라진다. 본 논문에서 제안한 KAST에서는 사용자가 로그 블록의 연관성을 K로 제한할 수 있으며 이에 따라 병합 연산은 사용자가 지정한 K에 따른 시간 내에 종료되는 것을 보장한다.

실험 결과를 통해 KAST는 K 변경에 따른 로그 블록 병합 비용 최대값을 제한할 수 있으며 이에 따라 로그 블록 병합 종료 시간을 예측할 수 있음을 알 수 있었다. 이에 따라 KAST는 정해진 시간 내 작업을 완료해야 하는 실 시간 시스템에 적합함을 알 수 있다. 또한 다른 기법에 비해 전체 I/O 수행 성능 역시 뛰어남을 알 수 있었다. KAST는 전체 I/O 수행 성능을 향상시키기 위해 데이터 기록 시 로그 블록의 연관성을 최대한 분산하여 기록한다. 또한 기록 요청이 순차적인 성질에서 랜덤한 성질로 바뀌면 기존에 사용하던 로그 블록의 특성도 함께 변경하여 로그 블록의 사용률을 최대한 높이게 된다. 이에 따라 다른 FTL 매핑 기법에 비해 효율적으로 로그 블록을 사용할 수 있다.

참 고 문 헌

[1] Y.-H. Bae. Design of a high performance flash memory-based solid state disk. *Journal of Korean Institute of Information Scientists and Engineers*, 25(6), 2007.

[2] J.-U. Kang, J.-S. Kim, C. Park, H. Park, and J. Lee. A multichannel architecture for high-performance nand flash-based storage system. *Journal of Systems Architecture*, 53(9):644-658, 2007.

[3] 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*, 3(4):837-863, 2004.

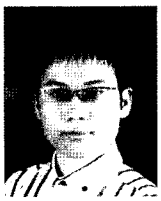
[4] J.-U. Kang, H. Jo, J.-S. Kim, and J. Lee. A superblock-based flash translation layer for nand flash memory. In *Proc. of International Conference on Embedded Software (EMSOFT)*, pp.161-170, 2006.

[5] 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 Transactions on Consumer Electronics*, 48(2):366-375, 2002.

[6] S. Lee, D. Shin, Y. Kim, and J. Kim. Last: locality-aware sector translation for nand flash memory-based storage systems. In *Proc. of IEEE International Workshop on Storage and I/O Virtualization, Performance, Energy, Evaluation and*

Dependability (SPEED08), 2008.
[7] S.-W. Lee, D.-J. Park, T.-S. Chung, D.-H. Lee, S. Park, and H.-J. Song. A log buffer-based flash translation layer using fully-associative sector translation. *ACM Transactions on Embedded Computing Systems*, 6(3), 2007.
[8] S.-Y. Park, W. Cheon, Y. Lee, M.-S. Jung, W. Cho, and H. Yoon. A re-configurable ftl (flash translation layer) architecture for nand flash based applications. In *Proc. of International Workshop on Rapid System Prototyping*, pp.202-208, 2007.
[9] Bonnie benchmark, http://www.textuality.com/bonnie/.
[10] Postmark benchmark, J. Katcher. Postmark: A new file system benchmark, 1997.

조 현 진
2004년 배재대학교 컴퓨터공학과(학사) 2004년~현재 성균관대학교 전자전기컴퓨터공학과 석박사 통합과정. 관심분야는 운영체제, 낸드 플래시 메모리 등



하 병 민
2009년 성균관대학교 전자전기컴퓨터공학과(학사). 2009년~현재 성균관대학교 전자전기컴퓨터공학과(석사과정) 관심분야는 운영체제, 낸드 플래시 메모리 등

신 동 군
1994년 서울대학교 계산통계학과(학사) 2000년 서울대학교 전산과학과(이학석사) 2004년 서울대학교 컴퓨터공학부(공학박사). 2004년~2007년 삼성전자 소프트웨어 책임연구원. 2007년~현재 성균관대학교 정보통신공학부 조교수. 관심분야는 임베디드 시스템, 실시간 시스템, 저전력 시스템 등

엄 영 익
1983년 서울대학교 계산통계학과(학사) 1985년 서울대학교 전산과학과(이학석사) 1991년 서울대학교 전산과학과(이학박사) 1993년~현재 성균관대학교 정보통신공학부 교수. 2007년~현재 성균관대학교 정보통신처 처장. 관심분야는 분산 컴퓨팅, 시스템 소프트웨어, 운영체제, 미들웨어, 시스템 보안 등