효율적인 플래시 변환 계층을 위한 블록 연관성 제한 기법

(Block Associativity Limit Scheme for Efficient Flash Translation Layer)

옥 동 석 † 이 태 훈 †
(Dongseok Ok) (Taehoon Lee)
정 기 동 ††
(Kidong Chung)

요 약 최근 NAND 플래시 메모리는 소형, 경량, 저전력 소모, 빠른 접근 속도 등의 장점으로 내장형 시스템과 개인용 컴퓨터, 서버 시스템에서 널리 사용되고 있다. 플래시 메모리를 하드 디스크처럼 사용하기 위해서는 플래시 변환 계층이 필요하다. 이전에 많은 플래시 변환 계층들이 제안되었지만 이전에 제안되었던 플래시 변환 계층들은 블록 스래싱 문제와 블록 연관성 등 몇 가지 문제점을 가지고 있다. 이 논문에서는 위의 문제를 해결하기 위한 새로운 플래시 변환 계층을 제안한다. 이 기법은 로그 블록의 연관성을 제한하고 데이터 블록의 연관성을 제한하지 않아 합병 연산의 횟수를 최소화 하고, 새로운 공간 회수 기법은 로그 블록 가비지 컬렉션을 이용하여 합병 연산의 비용을 줄인다.

키워드 : 플래시 메모리, 플래시 변환 계층

Abstract Recently, NAND flash memory has been widely used in embedded systems, personal computers, and server systems because of its attractive features, such as non-volatility, fast access speed, shock resistance, and low power consumption. Due to its hardware characteristics, specifically its "erase-before-write" feature, Flash Translation Layer is required for using flash memory like hard disk drive. Many FTL schemes have been proposed, but conventional FTL schemes have problems such as block thrashing and block associativity problem. The KAST scheme tried to solve these problems by limiting the number of associations between data block and log block to K. But it has also block thrashing problem in random access I/O pattern. In this paper, we proposed a new FTL scheme, UDA-LBAST. Like KAST, the proposed scheme also limits the log block association, but does not limit data block association. So we could minimize the cost of merge operations, and reduce merge costs by using a new block reclaim scheme, log block garbage collection.

Key words : Flash memory, Flash translation layer

· 이 논문은 부산대학교 자유과제 학술연구비(2년)에 의하여 연구되었음
· 이 논문은 제36회 추계학술발표회에서 '효율적인 플래시 변환 계층을 위한 블록 연관성 제한 기법'의 제목으로 발표된 논문을 확장한 것임

† 학생회원 : 부산대학교 컴퓨터공학과
ok3@pusan.ac.kr
withsoul@pusan.ac.kr
†† 종신회원 : 부산대학교 컴퓨터공학과 교수
kdchungf@melon.cs.pusan.ac.kr

논문접수 : 2009년 12월 24일
심사완료 : 2010년 3월 28일

Copyright©2010 한국정보과학회 : 개인 목적이나 교육 목적인 경우, 이 저작물의 전체 또는 일부에 대한 복사본 혹은 디지털 사본의 제작을 허가합니다. 이 때, 사본은 상업적 수단으로 사용할 수 없으며 첫 페이지에 본 문구와 출처를 반드시 명시해야 합니다. 이 외의 목적으로 복제, 배포, 출판, 전송 등 모든 유형의 사용행위를 하는 경우에 대하여는 사전에 허가를 얻고 비용을 지불해야 합니다.

정보과학회논문지 : 컴퓨팅의 실제 및 레터 제16권 제6호(2010.6)

1. 서 론

플래시 메모리는 소형, 경량, 빠른 접근 속도, 내구성 등의 장점을 가지고 있어, 현재 많은 내장형 시스템뿐만 아니라 개인용 컴퓨터나 서버 시스템에도 널리 사용되고 있다. 플래시 메모리는 하드 디스크와 다른 하드웨어적 특징들을 가지고 있다. 이와 같은 플래시 메모리상에서 일반적인 파일 시스템인 ext2나 FAT 파일 시스템을 사용하기 위해서는 플래시 변환 계층(FTL)이 필요하다.

주요 FTL 기법으로는 로그 블록 기법, FAST 기법, KAST 기법이 있다. 로그 블록 기법은 블록 단위 주소 변환 방식의 단점을 보완한 기법이지만 로그 블록의 모든 섹터를 사용하지 않고 블록을 지우게 되는 블록 스래싱(Block thrashing) 문제가 발생한다. FAST 기법은 로그 블록 기법의 블록 스래싱 문제를 해결하기 위하여 로그 블록을 시스템의 모든 데이터 블록이 공유하는 방법을 사용하지만 로그 블록에 다양한 블록의 수정된 섹터가 저장될 경우 합병 연산의 비용이 증가하는 블록 연관성 문제가 발생한다. KAST 기법은 로그 블록에 저장할 수 있는 데이터 블록의 수를 제한하여 블록 연관성 문제를 해결하였지만 로그 블록 기법과 마찬가지로 데이터 블록이 하나의 로그 블록에만 수정된 섹터를 저장할 수 있어 합병 연산의 횟수가 증가할 수 있다.

본 논문에서는 블록 연관성 문제를 해결하기 위하여 로그 블록의 블록 연관성을 제한하고 데이터 블록의 연관성은 제한하지 않는 UDA-LBAST(Unlimited Data-block Associativity - Limit log-Block Associative Sector Translation) 기법을 제안한다.

본 논문은 다음과 같이 구성되어 있다. 2장은 플래시

메모리의 특징과 기존에 제안되었던 플래시 변환 계층에 대하여 설명한다. 3장은 본 논문에서 제안하는 UDA-LBAST 기법의 구조와 새로운 공간 회수 기법은 로그 블록 가비지 컬렉션에 대하여 설명하고, 4장에서 시뮬레이션을 통하여 FAST, KAST 기법과 성능을 비교한다.

2. 관련 연구

2.1 플래시 메모리

플래시 메모리는 기존의 저장장치인 하드 디스크와 비교하여 경량, 소형, 내구성, 빠른 접근 속도 등 많은 장점을 가지고 있지만 하드 디스크와 다른 몇 가지 특징들도 가지고 있다. 플래시 메모리는 한번 기록된 섹터의 데이터를 수정할 수 없다. 이 데이터를 수정하기 위해서는 블록 전체를 지워 모든 섹터를 초기화한 다음 데이터를 다시 기록해야 하므로 지움 연산이 많이 발생할 경우 시스템의 성능이 저하된다[2].

플래시 메모리의 데이터 수정을 위하여 일반적으로 Out-place 업데이트 방법을 사용한다. 플래시 메모리에서 Out- place 업데이트 방법을 사용할 경우 논리 섹터 주소와 물리 섹터 주소는 일치하지 않으므로 논리 섹터 주소와 물리 섹터 주소를 RAM에 주소 변환 테이블의 형태로 저장하여 파일 시스템에서 어떤 논리 섹터 주소에 대한 연산이 요청되면 플래시 변환 계층에서 주소 변환 테이블을 참조하여 이 논리 섹터 주소를 물리 섹터 주소로 변환한다.

플래시 메모리에 무효 섹터의 수가 증가하면 사용할 수 있는 플래시 메모리의 공간이 작아지므로 결과적으로 플래시 메모리의 공간 효율이 저하된다. 따라서 블록에 존재하는 무효 섹터들을 제거해야할 필요가 있다.

2.2 기존 플래시 변환 계층 기법들

로그 블록 기법은 대표적인 혼합 주소 변환 방식의 기법 중 하나이다[3]. 로그 블록 기법은 데이터 블록에서 수정된 섹터를 저장하기 위해, 해당 데이터 블록에 로그 블록을 하나 할당한다. 시스템에 존재할 수 있는 로그 블록의 수는 제한되어 있으므로 이보다 많은 수의 데이터 블록이 수정되어 로그 블록이 필요하면 가장 무효 섹터의 수가 많은 블록을 합병하게 된다. 만약 모든 로그 블록이 하나의 섹터만 사용했다고 가정했을 때, 블록 합병이 필요하다면 어느 하나의 로그 블록과 데이터 블록은 합병해야한다. 이 경우 로그 블록에서 하나의 섹터를 제외한 다른 섹터는 사용하지 않고 합병되어 지워지므로 로그 블록의 사용률이 저하되고, 합병 횟수가 증가하여 플래시 메모리의 수명이 감소된다. 이를 블록 스래싱 문제(Block Thrashing)라 한다.

FAST(Fully Associative Sector Translation) 기법은 로그 블록 기법의 블록 스래싱 문제를 해결하기 위하여 제안되었다[4]. FAST 기법은 로그 블록 기법과 달리 로그 블록을 시스템의 모든 데이터 블록이 공유할 수 있다. FAST 기법의 로그 블록은 최대 블록을 구성하는 섹터 수만큼의 데이터 블록과 연관될 수 있으므로 이 경우 합병 연산의 비용이 증가하여 시스템 성능이 저하될 수 있다. 이를 블록 연관성 문제라 한다.

KAST 기법은 FAST 기법의 블록 연관성 문제를 해결하기 위하여 제안되었다[5]. KAST 기법은 로그 블록의 연관성을 최대 K개로 제한하여 블록 합병 연산의 비용을 최소화한다. 하지만 KAST 기법의 경우 입출력 연산의 주소 접근이 지역성이 없고 임의적일 경우 블록 스래싱 문제가 발생하여 FAST 기법보다 합병 연산의 횟수가 많아져, 시스템 성능이 저하되고, 플래시 메모리의 수명이 감소된다.

그림 1은 KAST 기법에서 데이터 블록의 연관성이 제한된 경우 발생할 수 있는 문제점을 나타낸 것이다. 이때 섹터 1이 수정되면 데이터 블록의 수정된 섹터는 로그 블록 0에만 저장할 수 있으므로 로그 블록 0과 데이터 블록 0, 데이터 블록 2가 합병된다.

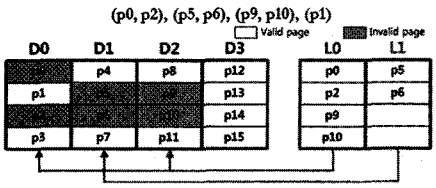

그림 1 KAST 기법의 문제점

3. UDA-LBAST

본 논문에서 제안하는 플래시 변환 계층 기법은 UDA-LBAST으로 이 기법의 구조와 기존 기법과의 차이점, 새로운 공간 회수 기법인 로그 블록 가비지 컬렉션 기법에 대하여 설명한다.

3.1 개요

UDA-LBAST 블록 연관성 문제를 해결하기 위하여 로그 블록의 연관성을 TH_{sh}로 제한한다. 하지만 KAST 기법의 문제점을 해결하기 위하여 데이터 블록의 연관성을 제한하지 않는다. 즉, 데이터 블록은 다수의 로그 블록에 수정된 섹터를 저장할 수 있다. 최초의 데이터는 모두 데이터 블록에 저장되고, 수정되는 데이터는 로그 블록에 저장된다. 로그 블록 기법과 유사하게 데이터 블록에 하나의 로그 블록을 할당하여, 해당 데이터 블록의 수정된 섹터는 모두 자신의 로그 블록에 기록하도록 한다. 시스템의 로그 블록 수는 정해져있으므로, 로그 블록의 수가 부족할 경우 가장 블록 연관성과 사용률이 낮은 로그 블록을 다수의 데이터 블록들이 공유하게 된다. 만일

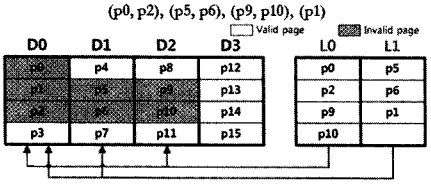

그림 2 UDA-LBAST 기법에서의 예

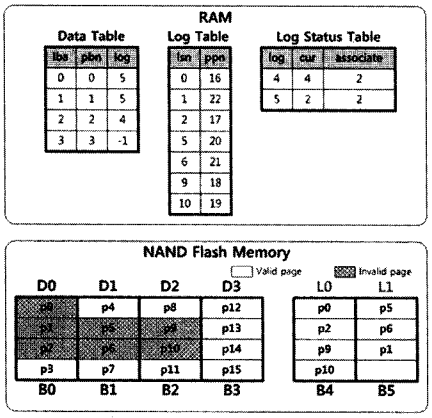

그림 3 UDA-LBAST 기법의 주소 변환 테이블

로그 블록의 블록 연관성이 TH_{sh}보다 낮은 블록이 없다면 순차적으로 기록된 로그 블록이나, 블록 연관성이 가장 낮은 로그 블록을 합병한다. 만약 데이터 블록이 자신에게 할당된 로그 블록에 빈 공간이 없을 경우 공유할 다른 로그 블록을 검색하여 그 곳에 저장하도록 한다.

그림 2는 KAST 기법에서 발생한 문제를 해결한 모습이다. 섹터 1이 수정되었을 때 로그 블록 1의 블록 연관성은 1이고 블록의 50%가 사용되었다. UDA-LBAST 기법은 섹터 1을 로그 블록 1에 저장하고, 이후에 수정될 데이터 블록 1의 섹터는 로그 블록 1에 저장될 것이다.

그림 3은 UDA-LBAST 기법의 테이블이다. 데이터 테이블은 데이터 블록을 관리하기 위한 테이블로써 데이터 블록의 경우 모든 섹터가 순서대로 기록되므로 블록 단위 주소 변환 방식을 사용한다. 로그 테이블은 로그 블록에 수정된 섹터가 비순차적으로 저장되므로 섹터 단위 주소 변환 방식을 사용한다. 로그 상태 테이블은 로그 블록의 연관성과 사용률을 관리하기 위한 테이블로 로그 블록의 공간이 부족할 경우 공유할 로그 블록을 찾기 위해 사용한다.

3.2 블록 연관성 제한

그림 4는 KAST 기법의 로그 블록과 데이터 블록의 연관 관계를 나타낸 것이다. KAST 기법의 로그 블록은 다수의 데이터 블록이 수정된 섹터를 저장할 수 있다. 하지만 데이터 블록의 경우 데이터 블록은 그 자신에게 할당된 로그 블록에만 수정된 섹터를 저장할 수 있다.

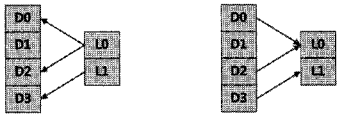

그림 4 KAST 기법의 블록 연관 관계

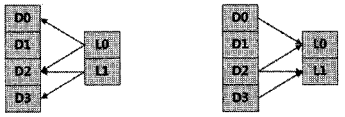

그림 5 UDA-LBAST 기법의 블록 연관 관계

KAST 기법과 달리 UDA-LBAST 기법은 그림 5와 같이 데이터 블록이 다수의 로그 블록에 수정된 섹터를 저장할 수 있다.

UDA-LBAST 기법이나 KAST기법과 같이 로그 블록의 블록 연관성을 제한할 경우 합병 연산의 비용이 증가할 수 있다. 플래시 메모리의 읽기, 쓰기 지움 연산의 비용을 각각 C_r, C_w, C_e, 로그 블록 연관성 제한 임계값을 TH_{sh}, 블록을 구성하는 섹터의 수를 N_p, 블록 연관성을 제한하지 않았을 때 합병 연산 발생 횟수를 N_m라 할 때, 가장 성능이 좋지 않은 경우의 전체 합병 연산의 비용은 다음과 같다.

$$C_m^{Worst} = \{TH_{sh} \cdot N_p \cdot (C_w + C_r) + (TH_{sh}+1) \cdot C_e\} \cdot \frac{N_p}{TH_{sh}} N_m$$

로그 블록의 연관성을 TH_{sh}로 제한할 경우 입출력 연산 주소 접근에 지역성이 없고 임의적일 경우 로그 블록의 모든 섹터를 사용하지 못하고 합병하게 되는 블록 스래싱 문제가 발생한다. 블록 스래싱 문제가 발생할 경우 로그 블록의 크기가 TH_{sh}/N_p만큼 감소하는 것과 같으므로 합병 연산의 횟수는 N_p/TH_{sh}의 비율로 증가한다. 이 경우 1회 합병 연산의 비용은 로그 블록의 연관성을 제한하지 않았을 경우보다 낮지만 합병 연산 횟수가 증가하여 전체적인 성능이 저하된다.

만일 입출력 주소 접근이 순차적이거나 지역성이 있어 로그 블록의 연관성을 제한하더라도 로그 블록의 모든 섹터를 다 사용하는 경우의 전체 합병 비용은 다음과 같다.

$$C_m^{Best} = \{TH_{sh} \cdot N_p \cdot (C_w + C_r) + (TH_{sh}+1) \cdot C_e\} \cdot N_m$$

로그 블록의 연관성을 제한하였기 때문에 1회 합병 연산의 비용도 낮고 블록 스래싱 문제도 발생하지 않으므로, 블록 연관성이 낮을수록 시스템의 성능이 향상된다. 이 경우 블록 연관성이 낮아지더라도 주소 접근에 지역성이 높아져 로그 블록의 모든 섹터가 다 사용되었다고 가정한다.

3.3 플래시 메모리 공간 회수 기법

Out-place 업데이트 방법을 사용하여 수정된 섹터를 저장하면 데이터 블록과 로그 블록의 무효 섹터 비율이 증가하게 된다. 플래시 메모리에 무효 섹터가 많아지게 되면 공간 사용률이 저하되므로 무효 섹터를 제거하기 위하여 데이터 블록과 로그 블록의 유효 섹터를 다른 블록으로 옮기는 블록 합병 연산을 수행한다. 블록 합병 연산은 스위치 연산, 부분 합병 연산, 완전 합병 연산 세 가지가 있다. UDA-LBAST는 완전 합병 연산의 비용을 줄이기 위하여 로그 블록 가비지 컬렉션이라는 새로운 공간 회수 기법을 사용한다.

스위치 연산은 로그 블록의 모든 섹터가 순서대로 저장된 경우로 로그 블록이 새로운 데이터 블록으로 치환된다. 부분 합병 연산은 로그 블록이 순서대로 저장되었지만 모든 섹터가 다 사용되지 않았을 경우 수행된다. 로그 블록의 빈 섹터를 데이터 블록에서 복사하고 로그 블록을 치환한다. 완전 합병 연산은 로그 블록이 비순차적으로 저장되었거나 2개 이상의 데이터 블록과 연관되었을 경우 수행된다. 완전 합병 연산은 로그 블록과 연관된 모든 데이터 블록을 새로운 데이터 블록으로 옮기는 과정을 수행한다.

그림 6은 완전 합병 연산의 과정을 보여준다.

완전 합병 연산의 비용은 로그 블록의 연관성과 비례하다. 따라서 UDA-LBAST 기법에서는 로그 블록의 연관성이 가장 낮은 블록을 선택하여 완전 합병 연산을 수행한다.

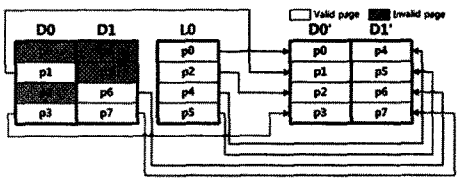

그림 6 완전 합병 연산의 예

3.3.4 로그 블록 가비지 컬렉션

UDA-LBAST 기법에서 로그 블록의 연관성을 제한하여 완전 합병 연산의 비용을 감소시킨다. 하지만 완전 합병 연산의 비용은 로그 블록의 연관성과 비례하므로 전체 로그 블록의 연관성이 높아지면 완전 합병 연산의 비용도 증가하게 된다. 로그 블록 가비지 컬렉션 기법은 로그 블록과 데이터 블록을 하나의 데이터 블록으로 합치는 합병 기법이 아니라 로그 블록을 새로운 로그 블록으로 옮기는 방법이다.

먼저 로그 블록을 무효 섹터가 많은 순서로 정렬한다. 그 후 로그 블록을 하나씩 가비지 컬렉션 리스트로 넣게 되는데 이 과정은 리스트 내의 로그 블록의 무효 섹터 수의 합이 블록을 구성하는 섹터의 수인 N_p보다 커질 때까지 반복한다. 이는 가비지 컬렉션 후 하나의 블록 크기 이상의 공간을 확보하기 위해서이다. 리스트의 구성이 끝나면 블록을 하나씩 할당하여 로그 블록의 유효 섹터들을 복사하고 이전 로그 블록은 지운다. 로그 블록이 옮겨지므로 로그 테이블을 수정한다.

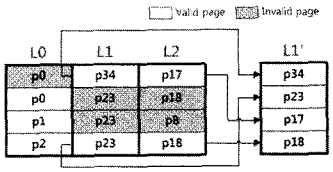

그림 7 로그 블록 가비지 컬렉션의 예

그림 7은 로그 블록 가비지 컬렉션의 예를 보여주는 것이다. 로그 블록 중 로그 블록 1과 로그 블록 2의 무효 섹터가 가장 많으므로 이 두 블록이 가비지 컬렉션 리스트에 삽입된다. 그 후 하나의 로그 블록이 할당되고 로그 블록 1과 로그 블록 2의 유효 섹터가 새로운 로그 블록으로 복사된다.

UDA-LBAST는 공간 회수가 필요할 경우, 로그 블록 가비지 컬렉션과 완전 합병 연산의 비용을 비교하여 가장 비용이 낮은 연산을 선택하도록 한다.

4. 실험 및 성능 평가

본 논문에서는 시뮬레이션을 통하여 FAST, KAST 기법과 비교하여 성능을 평가하였다. 시뮬레이션 시 플래시 메모리는 512MB 소블록 플래시 메모리로 가정하였고 로그 블록의 크기는 10%로 설정하였다.

그림 8은 시뮬레이션에서 사용한 트레이스 파일의 주소 접근을 나타낸 것이다. 좌측의 POSTMARK의 경우 임의적인 주소 접근이 많고, 우측의 IOZone은 순차적인 주소 접근이 많다.

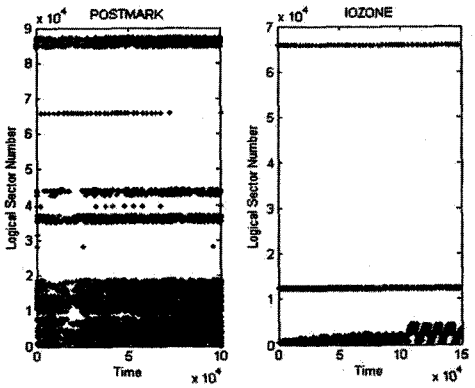

그림 8 시뮬레이션의 입출력 주소 접근

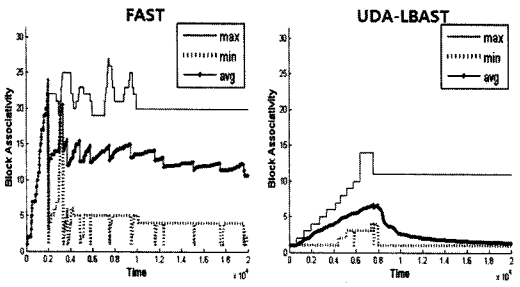

그림 9 POSTMARK 트레이스에 FAST 기법과 UDA-LBAST 기법의 블록 연관성 비교

그림 9는 임의적인 접근이 많은 POSTMARK 트레이스에서 FAST 기법과 UDA-LBAST 기법의 최대, 최소, 평균 블록 연관성을 비교한 것이다. UDA-LBAST 기법의 경우 FAST 기법과 달리 블록 연관성을 제한하고, 합병 연산 시 블록 연관성이 가장 낮은 블록을 선택하여 합병하므로 FAST 기법보다 더 낮은 블록 연관성을 보임을 알 수 있다. 특히 UDA-LBAST의 경우 시간이 지남에 따라 평균 블록 연관성이 계속 감소하는 것을 볼 수 있는데 이는 합병 연산과 로그 블록 가비지 컬렉션으로 인해 블록 연관성이 가장 높은 하나의 블록 외 모든 블록의 연관성이 낮아지기 때문이다.

그림 10은 POSTMARK 트레이스와 IOZone 트레이스를 모두 수행했을 때 FAST, KAST, UDA-LBAST 기법의 수행 시간을 비교한 것이다. 이 실험에서 KAST 기법과 UDA-LBAST 기법의 로그 블록 연관성 제한 임계값은 N_p의 2/3정도인 21로 설정하였다. 실험 결과 UDA-LBAST는 임의적인 주소 접근이 많은 POSTMARK에서 FAST 기법과 비교하여 약 66%, KAST 기법과 비교하여 약 15% 감소된 수행 시간을 보였다. 순차적인 주소 접근이 많은 IOZone 트레이스에서는 UDA-LBAST 기법이 FAST 기법과 비교하여 약 60%, KAST 기법과 비교하여 약 21% 감소된 수행 시간을 보였다.

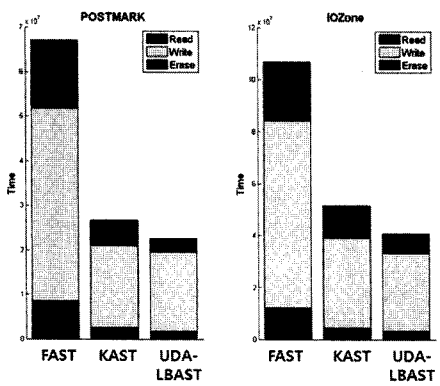

그림 10 POSTMARK와 IOZone 트레이스에서 FAST, KAST, UDA-LBAST의 성능 평가

5. 결 론

본 논문에서는 기존의 FTL 기법에서 나타날 수 있는 블록 연관성 문제를 해결할 뿐만 아니라 블록 연관성 문제를 해결하기 위해 제안되었던 KAST의 문제점도 해결할 수 있는 새로운 FTL 기법인 UDA-LBAST를 제안하였다. UDA-LBAST 기법은 로그 블록 기법에서 나타날 수 있는 블록 스래싱 문제 역시 동시에 해결할 수 있는 기법으로, 임의적인 환경에서 실험 결과 FAST 기법보다 약 66% 적은 연산 시간을 보였고, KAST 기법과 비교하였을 때는 약 15% 적은 연산 시간을 보였다. 순차적인 환경에서는 FAST 기법과 비교하였을 때 약 60%, KAST 기법과 비교하였을 때 약 21% 적은 연산 시간이 단축되었다.

참 고 문 헌

[1] Chung, T., Park, D., Park, S., Lee, D., Lee, S., and Song, H., "A survey of Flash Translation Layer. J. Syst. Archit," 55, pp.5-6, May. 2009.

[2] Samsung Electronics, K9XXG08UXM(1G × 8 Bit / 2G × 8 Bit NAND Flash Memory), 2005.

[3] Jesung Kim, Jong Min Kim, Sam H. Noh, Sang Lyul Min, Yookun Cho, "A space efficient flash translation layer for compactflash systems," *IEEE Transactions on Consumer Electronics*, vol.48, no.2, 2002.

[4] Sang-Won Lee, Dong-Joo Park, Tae-Sun Chung, Dong-Ho Lee, Sangwon Park, Ha-Joo Song, "A log buffer-based flash translation layer using fully-associative sector translation," *ACM Transaction on Embedded Computing Systems*, vol.6, no.3, 2007.

[5] Hungjin Cho, Dongkun Shin, Young Ik Eom, "KAST: K-Associative Sector Translation for NAND Flash Memory in Real-Time Systems," *DATE conference*, pp.507-512, 2009.