효율적 플래시 메모리 관리를 위한 워크로드 기반의 적응적 로그 블록 할당 기법

(Workload-Driven Adaptive Log Block Allocation for Efficient Flash Memory Management)

구 덕 회 † 신 동 군 ††
(Duckhoi Koo) (Dongkun Shin)

요 약 플래시 메모리는 저전력, 비휘발성, 충격 내구성의 특성 때문에 임베디드 시스템에서 가장 중요한 저장 장치로 사용되고 있다. 하지만, 플래시 메모리는 덮어쓰기가 안 되는 제약 때문에 FTL이라고 하는 주소 변환을 위한 소프트웨어를 사용하며, 효율적인 주소변환을 위해서 로그 버퍼 기반의 FTL이 많이 사용되고 있다. 로그 버퍼 기반 FTL의 설계시에 중요한 사항으로서 데이터 블록과 로그 블록의 연관 구조를 결정하는 문제가 있다. 기존의 기법들은 설계시에 결정된 정적인 구조를 사용하지만, 본 논문에서는 어플리케이션의 시간적 공간적 워크로드의 변화를 고려한 적응적 로그 블록 연관 구조를 제안한다. 제안하는 FTL은 실행시간에 어플리케이션의 워크로드의 변화에 최적화된 로그 블록 연관 구조를 사용함으로써 정적으로 최적의 연관 구조를 선택하는 기존의 기법 대비 5~16%의 성능 향상을 가져왔다.

키워드 : 플래시 메모리, 플래시 변환 계층, 로그 버퍼, 실시간 시스템, 하이브리드 매핑, 임베디드 스토리지

Abstract Flash memory has been widely used as an important storage device for consumer electronics. For the flash memory-based storage systems, FTL (Flash Translation Layer) is used to handle the mapping between a logical page address and a physical page address. Especially, log buffer-based FTLs provide a good performance with small-sized mapping information. In designing the log buffer-based FTL, one important factor is to determine the mapping structure between data blocks and log blocks, called associativity. While previous works use static associativity fixed at the design time, we propose a new log block mapping scheme which adjusts associativity based on the run-time workload. Our proposed scheme improves the I/O performance about 5~16% compared to the static scheme by adjusting the associativity to provide the best performance.

Key words : flash memory, flash translation layer, log buffer, hybrid mapping, embedded storage

1. 서 론

낸드 플래시 메모리(NAND Flash Memory)는 저전력, 비휘발성, 충격 내구성의 특성 때문에 MP3 플레이어, 디지털카메라, PDA와 같은 모바일 임베디드 시스템에서 저장장치로 널리 사용되고 있다. 최근에는 낸드 플래시 메모리의 가격이 하락하고 용량이 늘어남에 따라 범용컴퓨터 시장에서도 사용이 확산되고 있다.

하드 디스크와는 달리, 낸드 플래시 메모리는 쓰기작업을 하기 전에 삭제작업을 먼저 해야 되는 특징 때문에 덮어쓰기가 지원되지 않는다. 그러므로, 특정 페이지의 데이터를 수정할 때는, 새로운 데이터를 다른 빈 페이지에 기록하고 이전 데이터는 무효화하여야 한다.

낸드 플래시 메모리의 이런 특징은 두 가지의 저장장

· 이 논문은 2008년 정부(교육과학기술부)의 재원으로 한국연구재단의 지원을 받아 수행된 연구임(KRF-2008-314-D00351)

† 학생회원 : 성균관대학교 정보통신공학부
michaelk92@skku.edu
†† 정 회 원 : 성균관대학교 정보통신공학부 교수
dongkun@skku.edu

논문접수 : 2009년 8월 28일
심사완료 : 2009년 12월 24일

정보과학회논문지: 시스템 및 이론 제37권 제2호(2010.4)

치 관리 기법을 요구한다. 첫 번째는 주소 매핑 테이블을 유지함으로써 파일 시스템으로부터의 논리적 주소를 낸드 플래시 메모리의 물리적 주소로 연결시켜주는 주소 매핑 기법(address mapping scheme)이고, 두 번째는 무효화된 페이지들을 재사용하기 위한 가비지 컬렉션 기법(garbage collection scheme)이다. 가비지 컬렉션 기법은 무효화된 페이지를 많이 포함하는 블록을 선택하여 유효한 페이지들을 다른 블록에 복사한 후에 해당 블록을 삭제하여 재사용 할 수 있도록 해주는 기법이다. 이 두 가지 관리 기법을 제공하기 위해 파일 시스템과 플래시 메모리 사이에 플래시 변환 계층(Flash Translation Layer, 이하 FTL)이라 불리는 소프트웨어 계층이 일반적으로 사용된다[1,2].

FTL은 주소 매핑 기법에 따라 블록 수준, 페이지 수준, 하이브리드 수준의 매핑 기법들로 분류된다. 블록 수준 매핑 기법[3]에서는 파일시스템으로부터의 논리 블록 주소를 낸드 플래시의 물리 블록 주소에 매핑하기 위한 블록 수준의 정보만을 관리하고, 블록내의 페이지는 논리 블록과 물리 블록에서 동일한 오프셋에 기록되도록 한다. 블록 단위로 매핑 테이블을 유지하므로 매핑 테이블 크기가 작은 장점이 있지만, 블록 내의 일부 페이지만 갱신되어도 블록 내의 모든 데이터를 새로운 빈 블록으로 복사해야 하는 오버헤드가 발생한다.

반면, 페이지 기반 FTL 기법[4]에서는 파일 시스템으로부터의 논리 페이지 주소를 낸드 플래시 메모리의 물리 페이지 주소로 매핑하는 기법으로 데이터의 업데이트나 가비지 컬렉션 수행 시 오버헤드가 없으므로 빠른 성능을 보여준다. 하지만 페이지 단위로 매핑 테이블을 유지하여야 하므로 큰 메모리 공간을 요구하는 단점이 있다.

이런 단점을 보안하기 위해 페이지 단위 매핑과 블록 단위 매핑을 함께 사용하는 하이브리드 매핑 기법이 제안되었다. 하이브리드 기법에서는 플래시 메모리 블록들을 데이터 블록과 로그 버퍼로 나누어 관리한다. 데이터 블록들은 일반적인 저장 공간으로 사용하고, 로그 버퍼는 덮어쓰기 작업을 위한 임시 저장 공간으로 사용된다. 그러므로, 데이터 블록에 이미 기록된 논리 페이지에 대한 업데이트 요청을 FTL이 받으면 새로운 데이터를 로그 블록에 기록하고 데이터 블록의 예전 데이터를 무효화 시킨다. 이때 해당 로그 블록과 데이터 블록이 연관되어 있다고 표현한다. 데이터 블록들은 블록 단위의 매핑 기법에 의해 관리되고 로그 블록들은 페이지 단위의 매핑 기법에 의해 관리된다. 로그 버퍼 내의 로그 블록은 전체 블록 중에 작은 일부만을 할당하므로 하이브리드 매핑 기법에서는 적은 매핑 테이블 크기를 가지면서도 효율적인 가비지 컬렉션을 수행할 수 있다는 장점을 가진다.

하지만 로그 버퍼에 더 이상 쓸 공간이 없을 경우에는 로그 블록의 데이터와 데이터 블록의 데이터를 모아서 새로운 데이터 블록에 기록하는 로그 블록 병합 연산을 통해 새로운 빈 공간을 확보해야만 한다. 병합 연산은 교체될 로그 블록을 선택하여 그 로그 블록의 유효한 페이지와 연관된 데이터 블록내의 유효한 페이지들을 빈 블록에 복사한 후 이들 블록들을 삭제하고 주소 매핑 테이블을 수정하는 과정으로 이루어진다. 일반적으로 복사 및 삭제 연산이 전반적인 병합 연산의 비용을 결정한다.

병합 연산은 교체 병합, 부분 병합, 완전 병합으로 분리된다[5]. 교체 병합은 병합될 로그 블록이 하나의 데이터 블록과 연관되어 있고, 블록 내에 모든 페이지들이 논리 오프셋과 동일한 물리 오프셋에 기록되어 있는 경우에만 수행 가능하며, 페이지 복사 연산 없이 한 번의 삭제 연산만을 요구하기 때문에 비용이 싸다는 장점을 가진다. 부분 병합은 병합될 로그 블록이 하나의 데이터 블록과 연관되어 있고, 로그 블록 내에 페이지들이 논리 오프셋과 동일한 물리 오프셋에 기록되어 있지만, 사용 가능한 페이지가 남아 있는 경우에 수행된다. 부분 병합은 교체 병합과 비슷하지만 추가적인 페이지 복사 연산이 발생한다. 완전 병합은 병합될 로그 블록이 다수의 데이터 블록들과 연관되어 있거나 로그 블록내의 페이지들이 논리 오프셋과 물리 오프셋이 다른 경우에 사용되며 복사 및 삭제 비용이 매우 높다. 따라서 하이브리드 기법에서는 로그 블록 병합을 최소화 하는 것과 값비싼 완전 병합 대신에 부분 병합이나 교체 병합을 수행하는 것이 성능 향상에 큰 영향을 미친다.

로그 블록의 병합 횟수와 평균 병합 비용은 데이터 블록과 로그 블록의 연관 구조(로그블록과 연관된 데이터 블록의 개수)에 따라서 크게 좌우되기 때문에 최적의 연관 구조를 결정하는 것은 FTL의 성능에 큰 영향을 미친다. 최근에 다양한 형태의 연관 구조들이 제안되어 있었지만, 기존 기법들은 워크로드의 가변성을 고려하지 않고 정적인 연관 구조를 사용하고 있다. 본 논문에서는 하이브리드 매핑 FTL에서 데이터 블록과 로그 블록 간의 연관 구조를 시간과 공간에 따라 변하는 워크로드에 최적화하여 동적으로 조절하는 새로운 FTL기법을 제안한다.

본 논문의 구성은 다음과 같다. 2장에서는 관련 연구로서 기존 하이브리드 매핑 기반의 FTL 기법들의 특징과 문제점에 대해 살펴보고 3장에서는 제안하는 기법에 대해 상세히 설명한다. 4장에서는 시뮬레이션을 통해 본 논문에서 제안한 기법의 성능을 보여주며, 마지막으로 5장에서는 결론에 대해 기술한다.

2. 관련연구

최근에 다양한 형태의 로그 버퍼 기반의 FTL 기법들이 제안되었다[5-9]. 이들은 데이터 블록과 로그 블록의 연관구조에 따라서 구분할 수 있다.

BAST(Block-Associative Sector Translation) 기법[5]에서는 하나의 데이터 블록이 하나의 로그 블록과 연관되어 있다. 어떤 데이터 블록의 페이지에 대한 업데이트 요청이 오면, 그 데이터 블록은 하나의 로그 블록을 할당받고 새로운 데이터는 해당 로그 블록에 순차적으로 기록된다. 전체 로그 블록의 개수에는 제한이 있으므로, 할당 가능한 로그 블록이 없는 경우에는 이미 할당된 로그 블록 중에 하나를 선택하여 병합연산을 통해 빈 로그 블록을 확보해야 한다. BAST 기법에서는 쓰기 접근 패턴이 대부분 순차적인 경우에 효율적인 가비지 컬렉션을 수행한다. 하지만, 쓰기 패턴이 랜덤한 경우에는 하나의 로그 블록이 오직 한 데이터 블록하고만 연관되기 때문에 로그 블록의 활용도가 낮아지는 단점이 있다. 작은 크기의 랜덤 쓰기가 많을 때 대부분의 로그 블록이 일부분만 사용 되어진 채로 가비지 컬렉션을 수행하기 때문이다. 이러한 문제를 로그 블록 쓰레싱(thrashing)이라고 한다.

BAST의 이러한 문제를 해결하기 위해 FAST (Fully-Associative Sector Translation) 기법[6]이 제안되었다. FAST에서는 하나의 로그 블록이 모든 데이터 블록에 의해 공유되고, 모든 쓰기 요청은 로그 블록들에 순차적으로 기록된다. 이것은 로그 블록의 활용도를 효율적으로 향상시키며, 로그 블록 병합 작업을 가능한 미룰 수 있게 한다. 그렇지만, 하나의 로그 블록이 여러 개의 데이터 블록의 페이지들을 포함하고 있기 때문에 BAST 기법보다 한 번의 로그 병합 당 비용이 크다는 단점이 있다.

BAST와 FAST 기법의 단점들을 보안하기 위해 이들 기법의 중간 형태인 BSFTL(Block-Set based Flash Translation Layer) 기법[7]과 SAST(Set-Associative Sector Translation) 기법[8]이 제안되었다.

BSFTL 기법은 BAST 기법에 기반을 두어 데이터 블록과 로그 블록 간에 1:1 매핑을 사용하지만 주기별로 데이터블록에 대한 업데이트 양을 측정하여 데이터 블록들을 핫 블록(hot block)과 콜드 블록(cold block)으로 구분하여 두 가지 연관 구조를 동적으로 제공한다. 핫 블록들은 1:1 매핑을 그대로 사용하는 반면, 콜드 블록들은 블록 그룹으로 묶어서 로그 블록을 공유하여 사용률을 높이도록 한다. 또한, 블록 그룹에서 콜드 블록이 핫 블록으로 변하게 되면 핫 블록을 분할하여 1:1 매핑을 사용하도록 하여 블록 그룹에서의 가비지컬렉션 발생 확률을 낮춰준다. 하지만, BSFTL의 경우 콜드 블록들을 그룹으로 묶어서 로그 블록을 공유하도록 하기 때문에 블록 그룹에서의 로그 병합 수행 시 로그 블록 병합 비용이 높으며 또한 랜덤하고 업데이트가 많은 특성의 워크로드에서는 핫 블록들이 많기 때문에 BAST처럼 동작하여 로그 블록 쓰레싱(thrashing) 문제가 발생된다. 또한, 단순히 콜드 블록들만을 묶어서 그룹을 생성하였기 때문에 워크로드의 특성을 고려하지 못하는 단점이 있다.

SAST 기법은 연속적인 N개의 논리적 블록들을 위해 N개의 데이터 물리 블록과 K개의 로그 물리 블록을 사용하는 N:N+K 매핑 기법을 사용한다. N개의 데이터 블록들은 데이터 블록 그룹(Data Block Group, 이하 DBG)을 구성하고 K개의 로그 블록들은 로그 블록 그룹(Log Block Group, 이하 LBG)을 구성하며 하나의 DBG와 하나의 LBG가 연관되어 있다. LBG내의 하나의 로그 블록은 연관된 DBG내의 임의의 데이터 블록들과 연관될 수 있다.

그림 1은 4:4+2 매핑으로 구성된 SAST 기법의 예를 보여준다. 하나의 블록은 4개의 페이지들로 구성되어 있다고 가정했다. 그림 1에서는 하나의 데이터 블록 그룹이 연속적인 4개의 논리적 블록들로 이루어져 있고, 각 DBG을 위한 LBG가 존재하고 LBG에 대한 K값은 2이다. 예를 들어, DBG0에는 100, 101, 102, 103의 물리 블록 번호(PBN: physical block number)를 가진 물리 블록들이 데이터 블록으로 할당되어 있으며, PBN 301과 PBN 302가 LBG0를 구성하며 DBG0와 연관되어 있다. 또한, 로그블록이 부족할 때, 병합연산의 타겟이 될 희생 로그 블록을 선택하기 위해서 로그 블록 그룹 단위의 LRU(least recently used) 테이블을 관리하고 있다. 사용된 지 오래된 블록일수록 일반적으로 무효화된 페이지를 많이 가지고 있으며 추후에 업데이트 요청에 의해서 무효 페이지가 늘어날 가능성이 적기 때문에 LRU 테이블을 이용하여 이러한 블록을 희생 블록으로 선택하면 전체적인 병합 비용을 줄일 수 있다. 만약 SAST 기법이 1:1+1 매핑으로 구성된다면 BAST 기법과 같은 기법으로 볼 수 있다.

SAST기법에서는 연관도를 결정짓는 N과 K의 값을 선택하는 것이 중요한 문제이다. 여기서 연관도는 로그 블록에 연관된 데이터 블록 개수를 말하며 연관도에 따라 로그 병합 횟수와 평균 로그 병합 비용이 크게 좌우되기 때문에 최적의 연관 구조를 결정하는 것이 중요하다. 연관도는 로그 블록을 공유하는 데이터 블록 개수(N)에 의해 결정되며 로그 블록을 공유하는 데이터 블록이 많은 경우 연관도가 높아져 로그 병합 비용이 증가하지만 사용률은 높아져 로그 병합 횟수가 줄어들게

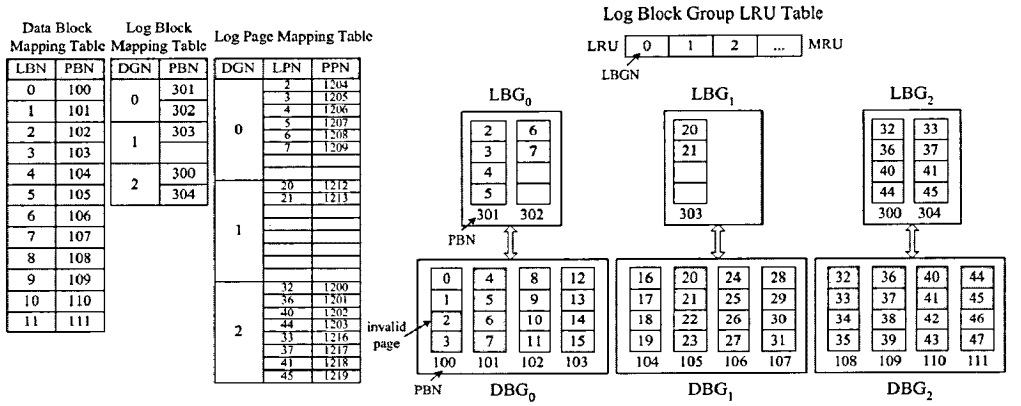


그림 1 SAST 기법 (N=4, K=2)

된다. 반면 연관도가 낮으면 로그 블록을 공유하는 데이터 블록이 적으므로 로그 병합 비용은 줄어들지만 사용률이 낮아져서 로그 병합이 자주 발생되게 된다. 예를 들어 그림 1에서 N값이 4이므로 최대 4개의 데이터 블록이 로그 블록을 공유할 수 있고 연관도는 최대 4가 될 수 있다. LBG0에서 PBN이 301인 로그 블록은 연관도가 2이며 두 개의 데이터 블록(PBN 100과 PBN 101)과 연관되어 있다.

SAST기법에서는 어플리케이션 마다 적절한 연관 구조를 제공하기 위해 어플리케이션의 특성을 고려하여 최적의 값(N, K)을 선택하여 사용하고 있다. 어플리케이션에 따라 스토리지 접근 패턴이 다르므로 최적의 N, K 값은 달라지기 때문에 각각의 어플리케이션에 대해 효율적으로 최적의 N과 K의 값을 찾을 수 있는 방법을 제공하지 않는다면 가능한 모든 조합을 사전 실험을 통하여 찾아야만 한다.

그래서, SAST 기법에서는 수학적 모델링을 통하여 각각의 워크로드를 분석하여 최적의 N과 K의 값을 결정하는 방법을 제공한다. 먼저, N값을 결정하는 방법은 워크로드의 전체 쓰기 요청을 특정 크기(한 블록의 페이지 개수)의 윈도우들로 분할을 하여 각각의 윈도우에서 논리 블록에 접근되는 쓰기 밀도를 조사하여 각 윈도우에 적합한 N값을 선정한다. 특정 윈도우에서 특정 블록에 대한 밀도가 높으면 공간적 지역성(spatial locality)이 높기 때문에 작은 N 값을 사용하고, 밀도가 낮으면 공간적 지역성이 낮기 때문에 큰 N 값을 사용한다. 이렇게 각각의 윈도우 별로 적절한 N값을 할당한 후, 전체 윈도우 내에서 가장 많이 할당 받은 N값들을 최적의 N값으로 사용하게 된다. 다음으로 K값은 시간적 지역성(temporal locality)에 의해 결정되므로 각각의 논리 블록에서 발생된 업데이트의 횟수를 측정하여 최적의 K값을 구한 후 가장 많은 수치를 기록한 K값을 최적의 K값으로 사용한다.

이렇게 최적의 값으로 선정된 N과 K값은 모든 주소공간에 동일하게 적용되며 실행시간 중에 변하지 않고 고정되어 있다. 하지만, 저장장치에 대한 I/O 패턴은 주소공간이나 시간에 따라서 다르기 때문에 이를 고려하지 않는다면 최적의 성능을 이끌어 낼 수 없다. SAST 기법의 경우 모델링을 통하여 워크로드를 분석할 때 시간적이나 공간적인 분석을 하였지만 최종적으로 가장 많은 수치를 기록한 대표 N, K값을 선정하여 정적으로 사용하기 때문에 모든 주소 공간과 시간적으로 변화되는 접근 패턴에 대해서는 성능이 떨어질 수밖에 없다. 또한, 워크로드의 특성을 설계시에 미리 알아내는데도 제약이 있다. 따라서 워크로드 패턴에 따라 적응적으로 DBG와 LBG의 크기인 N과 K를 변경하는 기법이 요구된다.

3. 적응적 SAST 기법

본 논문에서는 SAST 기법의 문제점을 개선하기 위해서 적응적 로그 블록 할당 기법을 사용하는 A-SAST 기법을 제안한다. A-SAST 기법에서는 크게 3가지 점에 있어서 개선을 했다. 첫째로는 희생 로그 블록 선택 방법을 그룹 단위에서 블록 단위로 개선을 하였고 둘째로는 고정된 DBG의 크기를 I/O 패턴에 따라 병합이나 분할을 통해 적절한 크기로 변경하도록 개선하였다. 셋째로는 LBG의 크기에 제한을 두지 않고 워크로드에 따라 희생 로그 블록 선택 기법이 자동적으로 최적의 LBG 크기를 선택하도록 하였다.

3.1 희생 로그 블록 선택

기존 SAST 기법에서는 로그 블록 그룹 LRU 테이블을 사용한다. 이 테이블은 로그 블록 그룹 단위로 LRU 정도에 따라 그룹 번호를 정렬해 두고, 희생 로그 블록이 필요할 때는 사용 된지 가장 오래된 그룹에서 하나의 로그 블록을 선택한다. 로그 그룹의 LRU 정도를 계

산하기 위해서 그룹 내의 블록에 대한 쓰기가 발생할 때마다 해당 그룹을 LRU 테이블의 맨 끝으로 이동시킴으로서 정렬된 테이블을 유지한다. 하지만 그룹 단위의 관리 기법은 정확성이 떨어질 수 있다.

예를 들어, 그림 1에서 보여진 SAST 기법의 경우, 각 페이지에 대한 쓰기 요청이 (32, 36, 40, 44, 2, 3, 4, 5, 6, 7, 20, 21, 33, 37, 41, 45) 순서로 발생할 때, 실제로 페이지 (32, 36, 40, 44)를 포함하는 PBN 300이 가장 오래된 로그 블록이지만 로그 블록 그룹 LRU의 특성으로 인해 페이지 (2, 3, 4, 5)를 포함하는 PBN 301 로그 블록이 희생 로그 블록으로 선택된다.

이러한 문제를 해결하기 위해서는 그룹 단위가 아닌 블록 단위 LRU 기법을 사용해야 한다. 또한 희생 로그 블록을 선택할 때, 병합 비용을 고려하여 병합 비용이 낮은 희생 로그 블록을 선택하는 것이 좋다. 그래서 A-SAST에서는 이 두 가지 요소를 함께 고려하여 희생 로그 블록을 선택한다. 그림 2는 위에서 설명한 두 가지 방법을 A-SAST 기법에 적용한 것을 보여준다. 그림에서 보듯이 로그 블록 단위로 LRU 테이블이 유지되며 희생 로그 블록 영역(victim log block region)을 두어 그 영역에 있는 로그 블록들 중 병합 비용이 가장 낮은 블록을 선택한다. 이때 한 개의 로그 블록 L의 전체 병합 비용은 로그 블록과 연관된 데이터 블록 개수를 사용하여 아래 수식에 의해 계산된다.

$$N \cdot A(L) \cdot C_{copy} + (A(L)+1) \cdot C_{erase}$$

위의 수식에서 N은 한 블록이 가진 페이지 개수, 연관도 $A(L)$은 로그 블록 L과 연관된 데이터 블록 개수, C_{copy}는 한 페이지의 복사 비용, C_{erase}는 한 블록의 삭제 비용을 나타낸다.

그림 2에서 각 페이지에 대한 쓰기 요청이 (32, 36, 40, 44, 2, 3, 4, 5, 6, 7, 20, 21, 33, 37, 41, 45) 순서로 발생할 때, 실제로 PBN 300번이 가장 오래된 로그 블록 이지만 희생 로그 블록 영역에서 병합 비용이 가장 적은 PBN 302가 희생 로그 블록으로 선택된다. PBN 300의 경우 $A(L)$이 4이므로 16개의 페이지 복사와 5개의 블록 삭제 연산이 필요하지만, 희생 로그블록으로 선택된 PBN 302의 경우 $A(L)$이 1이므로 4개의 페이지 복사와 2개의 블록 삭제 연산만 필요하다.

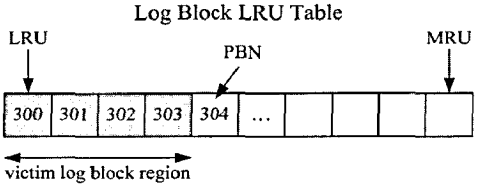


그림 2 A-SAST 기법에서의 로그 블록 LRU 테이블

이처럼 블록 수준의 LRU 기법을 사용하고, 추가적으로 병합 비용을 고려함으로써 앞으로 업데이트가 될 가능성이 낮으면서도 병합 비용이 적은 로그 블록을 병합의 대상으로 선택할 수 있다. 이렇게 블록 단위의 LRU 기법을 사용하면 그룹 단위보다 메모리 요구량이 증가할 수 있다. 블록단위 LRU 리스트 관리를 위한 메모리 요구량은 전체 로그 블록 개수에 비례하여 증가하게 된다. 하지만, 각 로그블록의 정보가 12 bytes로 표현된다고 할 때, 256개의 로그블록을 사용한다고 가정하면, 블록 단위 LRU를 유지하기 위해 3KB의 메모리가 요구되므로 그리 크지 않다고 볼 수 있다. 그러므로, 시스템의 메모리 환경을 고려하여 로그 블록 개수를 적절히 설계하는 것이 중요하다. 또한, 스토리지 용량이 매우 큰 경우에도 로그 블록 개수는 스토리지 크기에 비례하지 않고 워크로드의 지역성(업데이트 빈도)에 비례하여 할당하게 되므로 메모리 오버헤드가 큰 문제가 되지는 않는다.

3.2 데이터 블록 그룹 병합 및 분할

기존 SAST기법에서는 정적인 N:N+K 매핑 기법을 사용하여 N과 K가 모든 주소공간에 동일하게 적용되며 실행 시간동안에도 변하지 않고 고정되어 있다. 하지만, A-SAST 기법에서는 주소 공간이나 시간에 따라 변하는 I/O 패턴을 고려하여 각각의 DBG의 크기를 적합하게 조절한다. 각각의 DBG의 크기는 데이터 그룹을 병합하거나 분할하여서 조절된다.

그룹 병합은 로그 병합 비용이 낮으면서 로그 블록의 사용률이 낮은 연속적인 두 개의 데이터 블록 그룹을 병합하여 적당한 로그 블록 병합 비용을 보장하면서 로그 블록의 사용률을 최대로 높이기 위해서 수행된다. 이렇게 사용률이 높아지면 잦은 로그블록 병합이나 낮은 사용률로 인한 로그 블록 쓰레싱 문제를 해결할 수 있기 때문에 한정된 로그 블록을 효율적으로 사용할 있게 된다. 그룹 병합의 수행은 가비지 컬렉션 수행 시 희생 로그 블록으로 선택된 로그 블록과 연관된 DBG와 이웃한 DBG들중 하나가 병합 조건(낮은 로그 병합 비용과 로그 블록 사용률)을 만족하는 경우에 수행된다.

반면, 그룹 분할은 로그 블록의 연관도가 높아서 로그 블록 병합 비용이 높은 데이터 블록 그룹을 분할하여 연관도를 낮춤으로서 로그 블록 병합 비용을 줄이기 위해서 수행된다. 그룹 분할의 수행은 데이터 블록 그룹이 새로운 로그 블록을 할당 받을 때 이 그룹의 로그 블록 병합 비용을 검사하여 비용이 기준 값보다 높은 경우에 수행된다.

그룹 병합과 분할의 수행은 물리적인 유효 페이지 복사, 블록 삭제의 연산을 수행하는 것이 아니라 자료구조 관리 단위의 매핑 정보를 수정하는 작업으로 그림 1의 매핑 테이블 정보(Log Block, Log Page Mapping Table)

를 그림 3의 매핑 테이블 정보로 수정하게 된다.

병합된 그룹의 매핑 테이블 관리의 예를 보면, 그림 3에서 병합된 DBG0과 DBG1은 DBG0이 가비지 컬렉션 수행 시 선택된 로그 블록과 연관된 DBG로 병합의 주체가 되므로 병합된 그룹의 정보를 관리한다. 분할된 그룹의 매핑 테이블은 분할된 그룹별로 매핑 테이블 정보를 관리한다. 이처럼 그룹 병합 및 분할을 수행하기 위해 물리적인 연산 없이 간단한 매핑 정보의 수정 작업만 요구되고 또한 워크로드를 실시간으로 분석할 때 로그 블록의 사용률과 로그 블록 병합 비용(연관도)만을 계산하기 때문에 계산량과 관리 비용이 크지 않다. 따라서, A-SAST 기법의 그룹 병합 및 분할 작업은 그룹 크기를 동적으로 워크로드에 적합하도록 조절해줄 때 작은 수행 부하를 발생시킨다. 반면에 LBG의 크기에 대해서는 특별한 조정보다는 크기에 제한을 두지 않고 워크로드에 따라 희생 로그 블록 선택 기법이 자동적으로 최적의 LBG 크기를 선택하도록 하였다.

그림 1에서 보면 LBG_0과 LBG_1은 사용하지 않은 페이지를 많이 가지고 있어 로그 블록의 활용도가 낮다. 이것은 연관된 데이터 블록 그룹에 쓰기 요청이 많지 않기 때문이다. 따라서 그림 1의 DBG_0과 DBG_1과 같이 업데이트가 적게 발생하고 업데이트가 비교적 순차적이어서 병합 비용이 적은 블록들로 이루어진 데이터 블록 그룹들을 하나로 합쳐 새로운 데이터 블록 그룹으로 만들면 병합 비용은 크게 변화가 없고, 로그 블록의 활용도는 높아진다.

반면에 그림 1의 LBG_2의 경우에는 로그 블록의 사용률은 높지만 각 로그 블록의 연관도가 크기 때문에 병합 비용이 크다. 이것은 DBG_2에 대한 쓰기 패턴이 매우 랜덤한 것을 의미한다. 따라서, DBG_2를 두 개의 데이터 그룹으로 분할하면 각 로그 블록에 대한 병합 비용을 줄일 수 있다.

그림 3의 왼쪽 데이터 블록 그룹(DGB_{0+1})은 그림 1의 DBG_0과 DBG_1을 병합한 예를 보여준다. DBG_0와 DBG_1을 병합하여 8개의 데이터 블록으로 구성된 새로운 DBG가 생성 되었다. 그룹 병합 결과, 로그 블록의 활용도를 높여 로그 블록을 1개 절약할 수 있다. 데이터 블록 그룹의 병합은 두 개의 연속적인 DBG 들인, DBG_i와 DBG_j가 아래의 조건을 만족하는 경우에 수행한다.

$$\frac{P_{used}(\phi(DBG_i))}{P_{total}(DBG_i)} < \alpha \quad \text{and} \quad \frac{P_{used}(\phi(DBG_j))}{P_{total}(DBG_j)} < \alpha \quad \text{and}$$

$$\forall L, L \in \phi(DBG_i) \cup \phi(DBG_j), \quad A(L) < \beta$$

위의 수식에서 P_{used}는 LBG에서 사용 중인 페이지의 개수, P_{total}은 DBG에 할당된 전체 페이지 개수, $\Phi(g)$는 g에 할당된 로그 블록 그룹을 나타낸다. α, β는 각각 그룹에서의 로그 블록 사용률과 로그 병합 비용의 기준 값이며 기준을 만족하는 로그 블록 사용률이 낮으면서 로그 병합 비용이 낮은 그룹들을 병합하기 위해 사용된다.

그림 3의 오른쪽 데이터 블록 그룹들($DBG_{2.1}$과 $DBG_{2.2}$)은 그림 1의 DBG_2를 분할한 예를 보여준다. 이 경우에 각 페이지에 대한 쓰기 요청이 (32, 36, 40, 44, 33, 37, 41, 45) 순서로 발생할 때, DBG_2를 2개의 DBG로 분할하였기 때문에 각 로그 블록들의 최대 병합 비용이 반으로 줄어드는 것을 볼 수 있다.

이처럼 특정 그룹에서 로그 블록에 연관도가 높은 경우에 쓰기 패턴이 랜덤한 데이터의 업데이트가 많다고 추론할 수 있고, 그룹을 분할하면 병합 비용을 줄여 성능을 높일 수 있다. DBG의 분할은 새로운 로그 블록을 할당 받을 때 해당 로그 블록 그룹에서 마지막으로 할당된 로그 블록 L의 데이터 블록에 대한 연관도를 분할 기준 값(γ)과 비교 후 아래 조건을 만족할 때 발생한다.

$$A(L) > \gamma$$

γ는 그룹 분할시 로그 병합 비용의 기준 값이며 기준을 만족하는 로그 병합 비용이 높은 그룹을 분할하기 위해 사용된다.

α, β, γ의 값은 실험을 통해서 적절한 값을 선택하

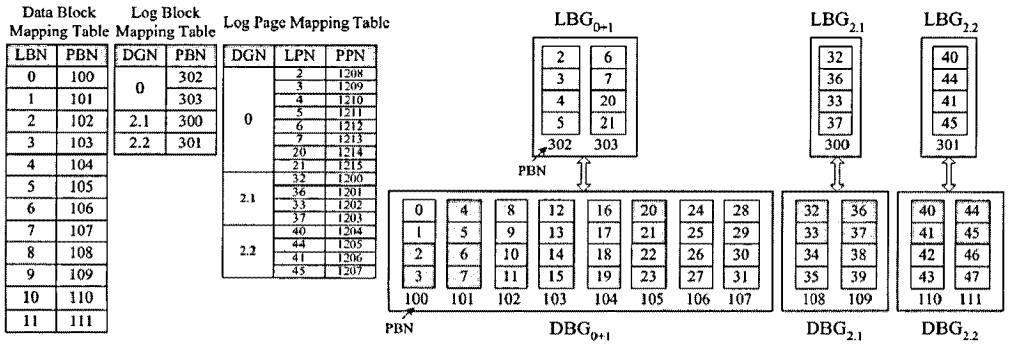


그림 3 A-SAST 기법의 그룹 병합 및 분할 예

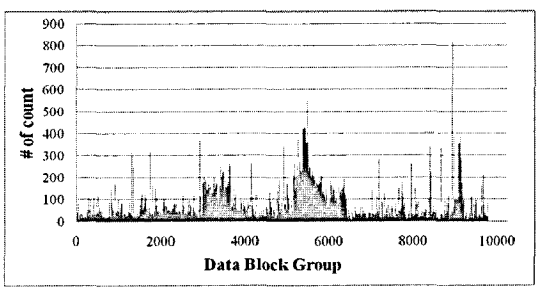
여 사용한다. 물론 알고리즘을 적응적으로 설계하여 실행 시간에 최적의 α, β, γ 값을 찾아서 사용할 수도 있겠지만, 추가 오버헤드가 커지게 되므로 제안된 기법에서는 몇 가지 벤치마크를 선택하여 대부분의 워크로드에서 공통적인 최적의 값을 찾아 고정된 값으로 사용한다. 실험 결과, 최적의 값들은 워크로드에 독립적임을 확인할 수 있었다. 분석적 기법을 통해서 최적의 알고리즘 기준 값이 워크로드에 독립적임을 보여야 하지만 이것은 본 논문의 범위를 벗어나므로 경험적 실험을 통한 증명으로 대체하였다.

4. 성능 평가

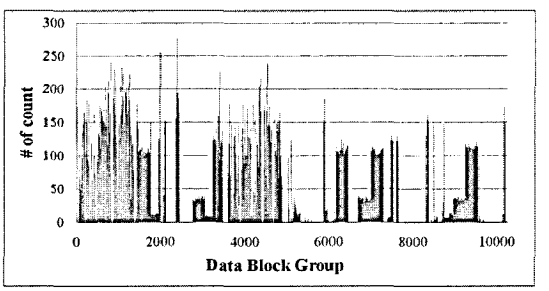
본 논문에서 제안된 기법을 시뮬레이션을 통해 평가하였다. 시뮬레이션 평가를 위하여 마이크로소프트 윈도우XP를 기반으로 한 데스크탑 컴퓨터에서 워드프로세서, 동영상 재생, 웹브라우징, 게임 등의 작업을 실행하며 일반적인 PC 사용에 대한 입출력 정보를 수집한 워크로드(PCtrace)와 서로 다른 크기의 파일들을 여러개 생성하여 랜덤한 위치에 시간에 따라 다른 양의 쓰기 작업을 수행하면서 수집한 워크로드(RandomFile)를 사용하였다. 또한, Iozone 벤치마크[10]를 사용하여 불규칙적으로 다수의 파일에 접근하는 입출력 정보를 수집한 워크로드(Iozone-4, Iozone-80)도 실험에 사용하였다. Iozone-4는 파일에 1~4KB크기의 작은 데이터를 기록하는 랜덤성이 강한 워크로드이며, Iozone-80은 파일에 2~80KB 크기의 쓰기 작업을 수행하여 랜덤한 데이터와 순차적인 데이터가 섞여있는 워크로드이다.

실험에서 가정한 낸드 플래시 메모리는 한 블록이 2KB 크기의 페이지 64개로 구성되어 있으며, 전체 블록 개수는 40960(5GB)개 이고, 로그블록 개수는 256개이다. 낸드 플래시 메모리의 연산에 대한 시간 변수는 표 1[11]과 같으며 I/O 비용이 CPU 비용 보다 훨씬 크기 때문에 시뮬레이션 결과는 플래시 메모리의 시간 변수만 사용하여 성능을 측정하였고 CPU 연산은 고려하지 않았다. 그룹 병합 및 분할 작업을 위한 기준 값은 실험(그림 6, 7)을 통하여 적당한 값(α : 0.2, β : 6, γ : 10)을 선정하였다. 희생 로그 블록 선택 기법에서 블록 단위 LRU 테이블의 희생 로그 블록 영역은 실험(그림 8)을 통하여 LRU 테이블의 20%로 선정하였다. 또한 SAST 기법의 모든 실험에서 K 값은 너무 작게 잡지 않는 이상 성능에 큰 영향이 없으므로 N/2 로 설정하였다. 반면 A-SAST에서는 K 값에 대한 제한이 없으므로 특정 값을 사용하지 않았다.

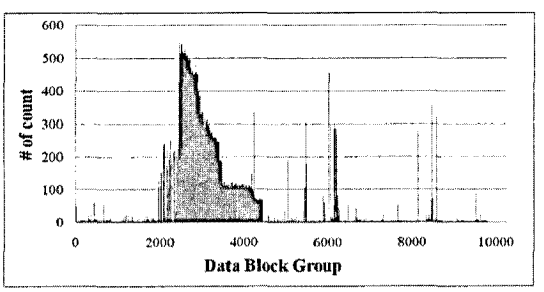
우선 기존 SAST 기법에서 발생되는 로그 블록 병합의 불규칙한 특징을 관찰하였다. 그림 4는 기존 SAST 기법을 수행하였을 때 각각의 DBG에서 발생되는 로그 병합 횟수를 보여준다. 그림 4를 통하여 워크로드에 따라서 특정 주소공간에 접근이 많이 발생되는 것을 확인할 수 있다. 그림 5는 시간이 지남에 따라 발생되는 로그 블록 병합 횟수를 보여 주는데 시간에 따라 저장장



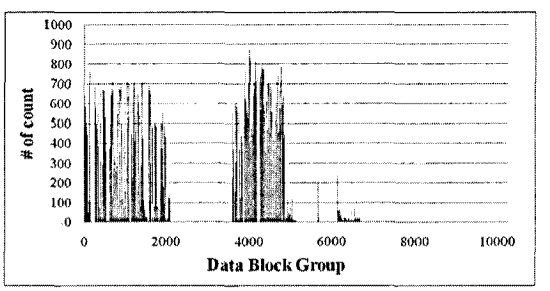
(a) PCtrace



(b) RandomFile

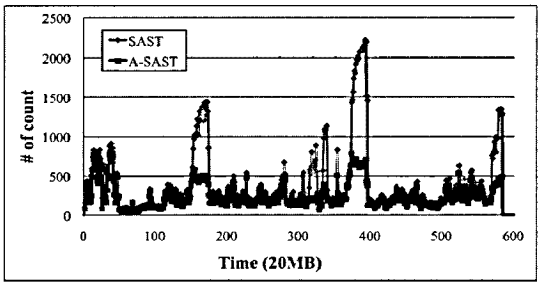


(c) Iozone-4

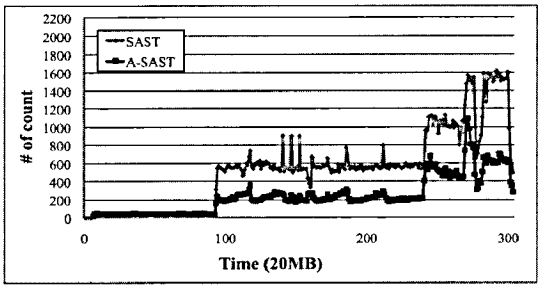


(d) Iozone-80

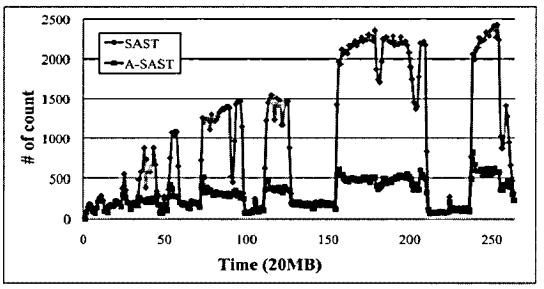
그림 4 SAST 기법의 로그 블록 병합 횟수 (N=4, K=2)



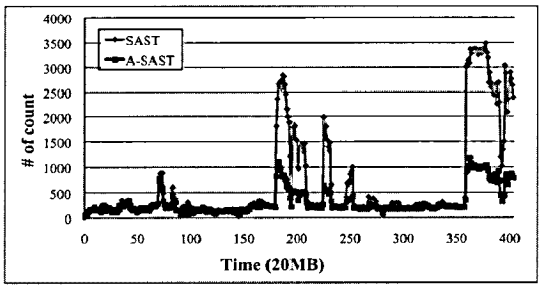
(a) PCtrace



(b) RandomFile



(c) Iozone-4



(d) Iozone-80

그림 5 SAST 기법의 로그 블록 병합 횟수 (N=4, K=2)

표 1 실험에 사용한 낸드 플래시 메모리의 시간 변수

Operation	Time
Page read	25 us
Page program	300 us
Block erase	2 ms
Bus speed	40 MB/s (25ns/byte)

치에 대한 I/O 패턴이 다르기 때문에 시간 별로 발생되는 로그 블록 병합 횟수가 상이한 것을 보여준다. 기존 SAST 기법에서는 변동 폭이 크며 로그 블록 병합이 많이 발생되는 것을 볼 수 있다.

그림 4와 5를 통하여 워크로드의 특징에 따라 접근빈도가 주소 공간과 시간에 따라 따르기 때문에 정적인 LBG 크기를 가진 SAST 기법에서는 최적의 성능을 이끌어 낼 수 없다는 것을 알 수 있다. 그림 5에서 주소 공간이나 시간에 따라 변하는 워크로드의 특징을 동적으로 고려하는 A-SAST 기법의 경우 시간에 따라 발생되는 로그 블록 병합의 변동 폭과 발생 횟수가 SAST 기법에 비하여 현저히 줄어든 것을 볼 수 있다.

다음으로 A-SAST 기법에서 그룹 병합 및 분할을 수행하기 위해 수식에서 사용되는 기준 값(α, β, γ)에 대한 실험을 하였다. 그림 6은 α가 0.2인 경우 β, γ 값에 따른 전체 성능의 변화를 보여준다. 실험결과를 통하여 β 값이 너무 작으면 그룹 병합이 적게 발생되어 작은 크기의 DBG가 많아지며, 반면에 너무 커지면 그룹 병합이 많이 발생되어 큰 크기의 DBG가 많아져 성능이 저하되는 것을 볼 수 있다. γ 값의 경우 너무 작으면 그룹 분할이 자주 발생되며 너무 커지면 분할이 적게 발생되어 성능이 약화되는 것을 확인할 수 있다. 이러한 특성으로 인하여 실험결과에서 보듯이 서로 다른 특성의 워크로드나 초기에 설정된 데이터 그룹의 크기(N)가 다른 경우에도 공통인 최적의 기준 값(β, γ)이 존재하는 것을 볼 수 있다. 실험에서 모든 워크로드에 대해 β 값은 4~8, γ 값은 4~12인 경우 최적의 성능을 보인다.

그림 7은 β, γ 값이 각각 6, 10으로 고정되어 있을 때 α 값을 0.0~1.0까지 변경하며 수행시간을 비교한 그래프이다. α 값이 0.0인 경우 그룹 병합이 발생되지 않으므로 작은 크기의 DBG가 많아져 성능이 가장 나쁘며, 0.2일 때 최적의 성능을 보이고 α 값이 증가될수록 그룹 병합이 자주 발생되어 너무 큰 크기의 DBG가 많아지기 때문에 성능 저하를 보인다. 그림 6과 7의 실험결과를 통하여 A-SAST 기법에서 그룹 병합 및 분할 수행 시 사용되는 적절한 기준 값을 선택할 수 있다.

다음으로 희생 로그 블록 영역의 크기에 따른 성능 변화를 측정하였다. 그림 8은 SAST 기법에 블록 단위 LRU 기법만 적용한 경우(SAST-V)에 대해 희생 로그 블록 영역의 크기를 변경해 가며 전체 수행 시간을 측정한 그래프이다. 희생 로그 블록 영역이 0%인 경우 LRU만 사용하므로 성능이 나쁘고 LRU 테이블의 20%

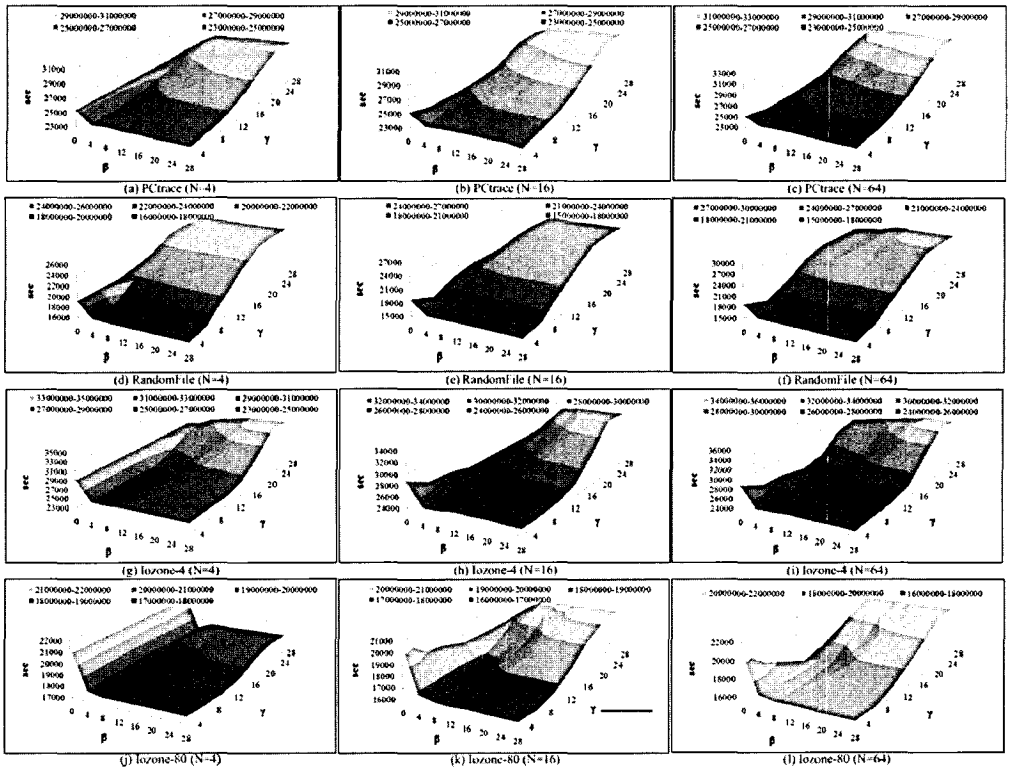


그림 6 A-SAST 기법의 매개변수별 I/O 성능(β, γ)

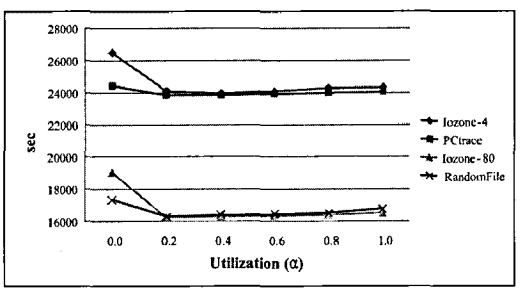


그림 7 A-SAST 기법의 매개변수별 I/O 성능 (N=16)

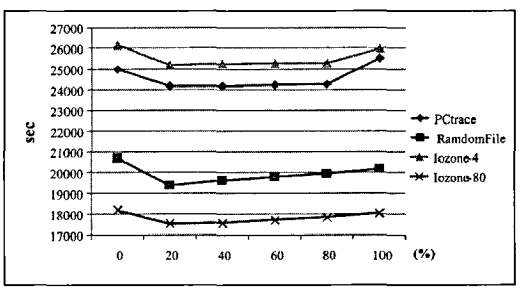
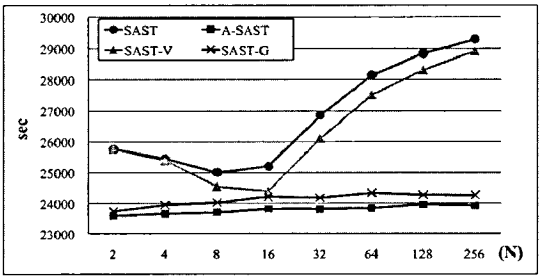


그림 8 SAST-V 기법의 회생 로그 블록 영역 크기별 수행시간 (N=16, K=8)

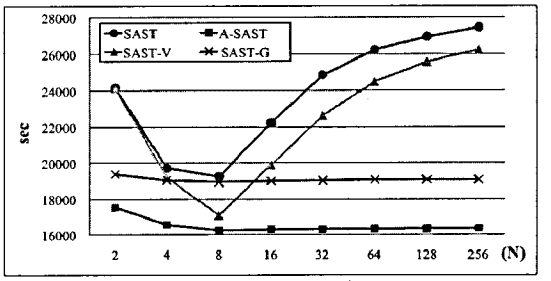
인 경우에는 LRU의 특성을 유지하는 범위에서 비용이 가장 낮은 희생 로그 블록을 선택하므로 0%인 경우보다 약 3~6%정도의 성능 향상이 있으며 100%로 갈수록 LRU의 특성이 무시되어 성능이 악화되는 것을 볼 수 있다.

다음으로 SAST 기법과 A-SAST 기법의 성능을 비교하였다. 성능 측정은 SAST 기법과 본 논문에서 제안한 블록 단위 LRU 기법(SAST-V)과 그룹 병합 및 분할(SAST-G)을 SAST에 각각 적용한 기법, 그리고 제안한 기법을 모두 적용한 A-SAST 기법의 성능을 비교 하였다. 그림 9는 전체 수행시간을 보여준다. 실험에서 SAST 기법의 경우 그룹 크기(N)가 작을 때나 너무 큰 경우 성능이 악화되고 8이나 16의 그룹 크기에서 최적의 성능을 보인다.

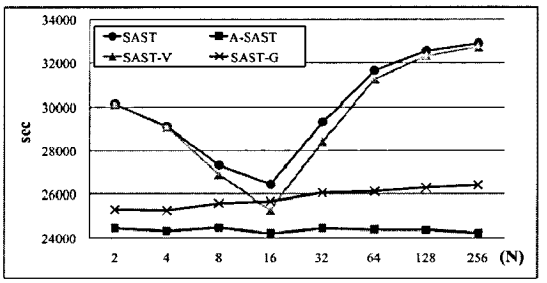
SAST 기법에 블록 단위 LRU 기법만 적용한 경우(SAST-V)가 그룹 단위의 LRU 정책을 사용하는 SAST 기법에 비해 LRU의 정확성이 높아지기 때문에 각각의 워크로드에서 평균적으로 약 1.5~6%의 성능 향상을 보인다. 그룹 크기가 작은 경우에는 그룹 단위나 블록 단위 LRU의 차이가 거의 없기 때문에 비슷한 성능을 보이며 그룹 크기가 커지면 블록 단위 LRU의 정확성이 높아지기 때문에 성능 향상을 보인다.



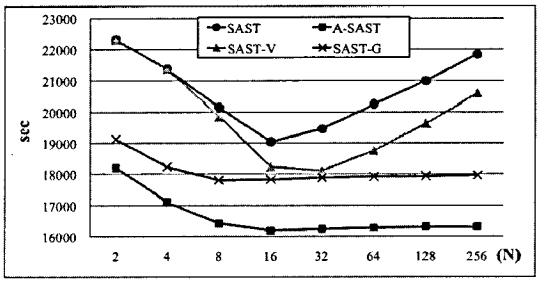
(a) PCtrace



(b) RandomFile



(c) Iozone-4



(d) Iozone-80

그림 9 각 기법별 전체 수행시간

SAST 기법에 그룹 병합 및 분할을 적용한 경우(SAST-G)는 주소 공간에 따라 다른 접근 패턴과 시간에 따라 변경되는 접근 패턴을 고려하여 동적으로 적절한 그룹 크기를 구성하기 때문에 최적의 정적인 그룹크기를 사용한 경우보다 약 2~6.5%의 성능 향상을 보인다.

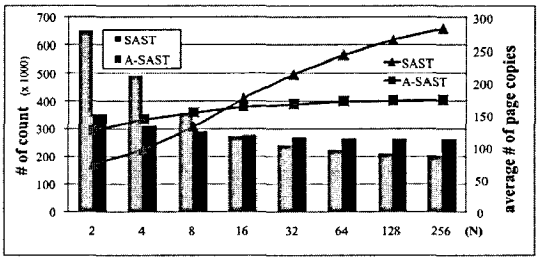
또한 두 가지 기법을 모두 적용한 A-SAST 기법의 경우 블록 단위 LRU 기법이 그룹 단위 LRU 기법보다 정확성이 높기 때문에 그룹 병합 수행 시 고려 조건인 사용률이 적은 그룹 판별에 있어 정확성이 높아진다. 따라서, 두 가지 기법을 모두 적용한 A-SAST 기법의 경우 SAST 기법에 비해 성능이 가장 좋은 경우 보다 약 5~16%의 성능 향상을 보인다.

다음으로 SAST 기법과 A-SAST 기법의 성능 차이 원인을 분석하기 위해 전체 로그 병합 횟수와 로그 블록 병합 시 평균 병합 비용(로그 블록 병합 수행시 유효 페이지 복사 개수)에 대해 실험을 하였다.

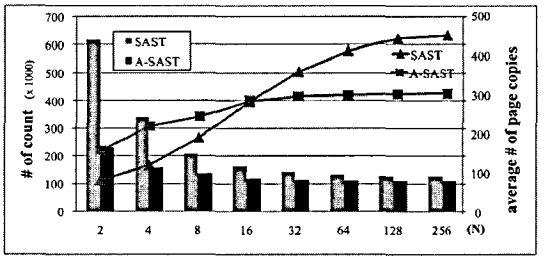
그림 10에서 막대 그래프는 전체 로그 블록 병합 횟수를 보여주고 실선그래프는 평균 병합 비용을 나타낸다. 실험 결과를 통하여 SAST 기법의 경우 그룹 크기가 작을 때는 사용률이 낮기 때문에 로그 블록 병합이 많이 발생되지만 그룹 크기가 커지면서 사용률이 높아지기 때문에 로그 블록 병합 횟수가 감소되는 것을 알 수 있다. A-SAST 기법의 경우 그룹 크기가 작은 경우 사용률이 낮은 그룹들을 병합하여 사용률을 높여주기 때문에 그룹 크기가 작은 경우에도 로그 블록 병합 횟수가 낮은 것을 볼 수 있고 그룹 크기가 큰 경우에는 로그 블록 병합 비용이 높은 그룹들을 분할하기 때문에 SAST 기법에 비해 로그 블록 병합 횟수가 증가되는 경우를 볼 수 있다.

SAST 기법의 경우 그룹 크기가 작은 경우에는 로그 블록의 데이터 블록에 대한 연관도(A(L))가 작기 때문에 로그 블록 병합 수행 시 병합 비용이 낮지만 그룹 크기가 커지면서 A(L)이 커지므로 병합 비용이 증가되는 것을 볼 수 있다. 따라서 그림 9에서처럼 SAST 기법의 경우 그룹 크기가 작을 때는 병합 비용은 적지만 사용률이 낮아서 성능이 저하되고 그룹 크기가 큰 경우에는 사용률은 높지만 병합 비용이 크기 때문에 성능이 저하되는 것을 알 수 있다. A-SAST 기법의 경우 그룹 크기가 작을 때는 그룹 병합을 수행하므로 병합 비용이 증가 되고 그룹 크기가 큰 경우에는 그룹 분할을 통하여 병합 비용을 낮추는 것을 볼 수 있다.

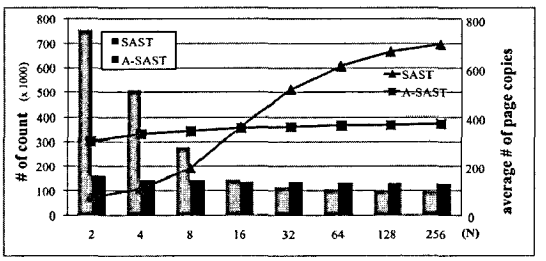
다음으로 제안된 A-SAST 기법이 시간이 지남에 따라 각각의 주소 공간에 적절한 그룹 크기를 조절해 나가는 것을 실험을 통해 확인하였다. 그림 11은 시간에 따라 변동되는 그룹 크기 분포의 변화를 보여 준다. 실험에서 초기 그룹 크기(N)가 4(초기 그룹 개수 10240)인 경우에 주로 그룹 병합이 발생되며 시간이 지남에 따라 적절할 그룹 크기를 찾아간다. 반면 초기 그룹 크기가 64(초기 그룹 개수 640)인 경우 주로 그룹 분할이 발생되며 시간에 따라 적절한 그룹 크기를 찾아간다. 그



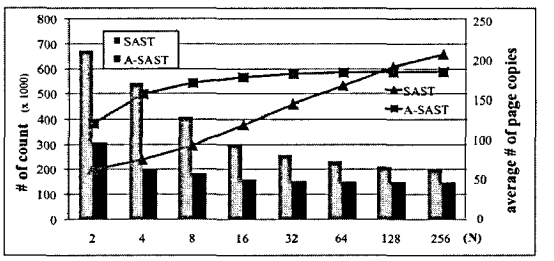
(a) PCtrace



(b) RandomFile

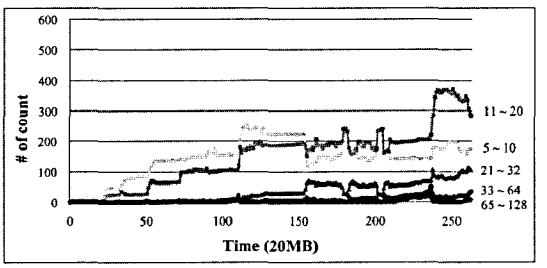


(c) Iozone-4

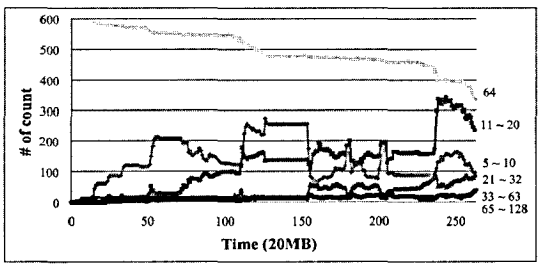


(d) Iozone-80

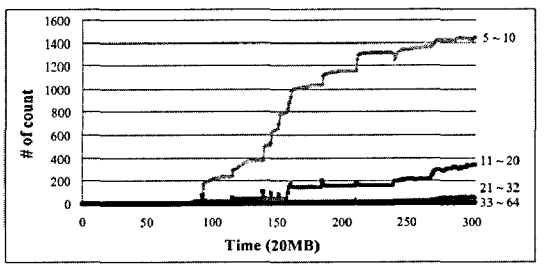
그림 10 SAST와 A-SAST의 로그 병합 횟수 및 평균 병합 비용 비교



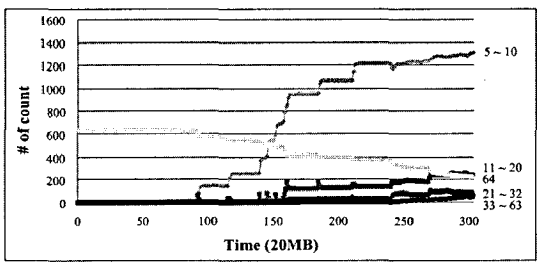
(a) Iozone-4 (N=4)



(b) Iozone-4 (N=64)



(c) RandomFile (N=4)

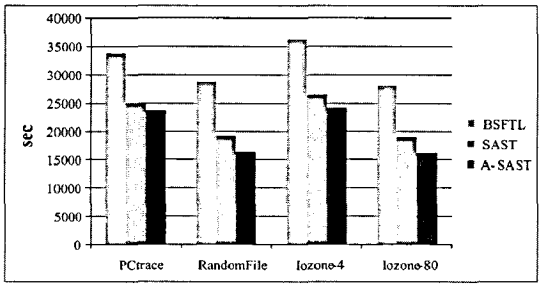


(d) RandomFile (N=64)

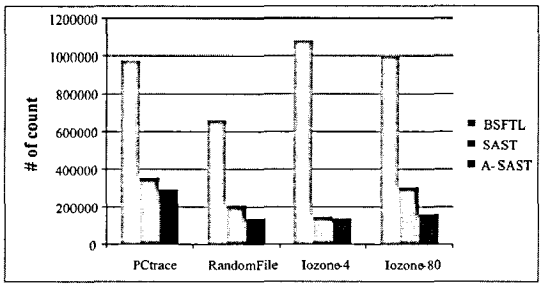
그림 11 A-SAST 기법의 DBG 크기 분포의 시간별 변화

결과, 초기 그룹크기가 달라도 그룹 병합과 분할을 통해서 비슷한 형태의 분포를 가짐을 알 수 있다.

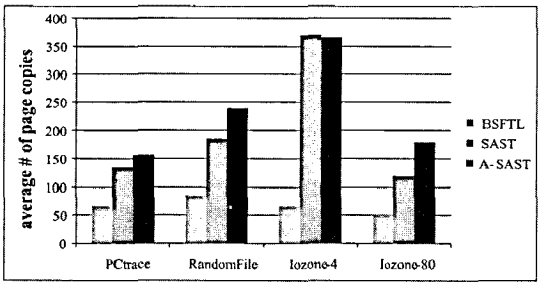
마지막으로 제안된 A-SAST 기법과 기본 개념이 비슷한 BSFTL 기법의 성능을 비교하여 차이점을 분석하였다. 실험에서 SAST와 A-SAST 기법의 N값은 그림 9에서 최적인 경우의 값(PCtrace와 RandomFile은 8, Iozone-4와 8은 16)을 사용 하였다. 그림 12(a)는 전체 수행시간을 나타내며 실험에서 SAST 기법은 BSFTL 기법 대비 25~33%의 성능 향상을 보이며, A-SAST 기법의 경우 30~43%의 성능 향상을 보인다. 그림 12(b), (c)를 보면 BSFTL은 SAST 기법과 A-SAST 기법에 비해 평균 로그 병합 비용은 낮지만 전체 로그



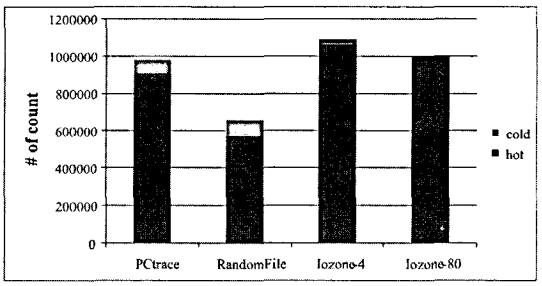
(a) 전체 수행 시간



(b) 전체 로그 병합 횟수



(c) 전체 평균 병합 비용



(d) 전체 로그 병합 hot/cold 비율

그림 12 성능 비교 - BSFTL, SAST, A-SAST

병합 횟수가 높은 것을 알 수 있는데 이것은 그림 12(d)에서 보듯이 BSFTL에서 전체 로그 병합의 대부분이 1:1 매핑을 사용하는 핫 블록에서 이루어지기 때문이다. 이처럼 BSFTL 기법은 BAST 기법에 기반을 두고 있기 때문에 랜덤하고 업데이트가 많은 특성의 워크로드에서는 핫 블록들이 많기 때문에 BAST처럼 동작하여 로그 블록 쓰레싱(thrashing)으로 인한 로그 병합이 빈번히 발생된다. 또한, 단순히 콜드 블록들만을 묶어서 그룹을 생성하였기 때문에 워크로드의 특성을 고려하지 못하는 단점이 있다.

반면 제안된 A-SAST 기법은 SAST 기법을 기반으로 하며 로그 블록 사용률이 낮으면서 로그 블록 병합 비용이 낮은 그룹들만을 병합하기 때문에 병합 비용은 낮으면서 로그 블록의 사용률을 최대로 높여주어 로그 블록 쓰레싱 문제를 해결하며 병합 비용이 높은 그룹에 대해서는 그룹 분할을 통하여 로그 병합 비용을 낮춰준다. 또한, 연속적인 논리 블록들을 병합/분할하기 때문에 각각의 주소 공간별로 워크로드에 최적인 서로 다른 그룹 크기를 가지게 된다.

5. 결 론

본 논문에서는 낸드 플래시 메모리 기반의 다양한 어플리케이션을 효율적으로 다루기 위해 새로운 FTL 설계를 제시하였다. 제안된 A-SAST 기법에서는 데이터 블록 그룹과 로그 블록 그룹의 연관관계를 정적으로 고정하여 사용하는 SAST 기법의 문제점을 해결하기 위해 프로그램 수행도중 현재 접근되는 쓰기 패턴에 따라 연관도를 동적으로 조정하여 최적의 값을 찾도록 하여 SAST 기법의 성능을 더욱 향상시켰다.

참 고 문 헌

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

[2] CompactFlash Association. http://www.compactflash.org

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

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

[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*, vol.48, no.2, pp.366-375, 2002.

[6] 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 Transactions on Embedded Computing Systems*, vol.6, no.3, 2007.

[7] Z. Z. Liu, L. H. Yue, P. Wei, P. Q. Jin, and X. O. Xiang, "An Adaptive Block-Set Based Management for Large-Scale Flash Memory," In *Proc. of the 2009 ACM Symp. on Applied Computing*

(SAC 2009), ACM Press, Hawaii, U.S.A., 2009.
[8] 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 Transactions on Embedded Computing Systems*, vol.7, no.4, 2008.
[9] J. U. Kang, H. Jo, J. S. Kim, and J. Lee, "A superblock-based flash translation layer for NAND flash memory," in *Proc. International Conference on Embedded Software*, pp.161-170, 2006.
[10] Iozone Filesystem Benchmark, http://www.iozone.org
[11] Samsung Electronics, NAND Flash Memory Datasheet, http://www.samsung.com/Products/Semiconductor/NANDFlash/SLC_LargeBlock/16Gbit/K9KAG08U0M/ds_k9xxg08uxm_rev10.pdf.

구 덕 회
2008년 한국산업기술대학교 전자공학과(학사). 2008년~현재 성균관대학교 전자전기컴퓨터공학과(석사과정). 관심분야는 임베디드 시스템, 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