HOT 데이터 블록 병합 지연을 이용한 효율적인 플래시 메모리 로그 버퍼 관리 기법

An Efficient Log Buffer Management Scheme of Flash Memory Through Delay of Merging Hot Data Blocks

김학철[*], 박용훈[*], 윤종현[**], 서동민[***], 송석일[****], 유재수[*]

충북대학교 정보통신공학과[*], 한국전자통신연구원 지식e러닝연구팀[**], 한국과학기술원 전자계산학과[***], 충주대학교 컴퓨터공학과[****]

Hak-Chul Kim(ageofjang@gmail.com)[*],Yong-Hun Park(yhpark@netdb.cbnu.ac.kr)[*], Jong-Hyeong Yun(jhyun@netdb.cbnu.ac.kr)[**], Dong-Min Seo(dmseo@cbnu.ac.kr)[***], Suk-Il Song(sisong@cjnu.ac.kr)[****], Jae-Soo Yoo(yjs@cbnu.ac.kr)[*]

요약

본 논문에서는 데이터의 접근성에 따른 병합 가치를 고려한 새로운 로그 버퍼 관리 기법을 제안한다. 제안하는 기법은 로그 블록의 병합 가치를 평가하여 빈번하게 갱신이 발생하지 않는 데이터에 대해서 데이터 블록과의 병합연산을 수행한다. 또한 빈번하게 갱신되는 데이터에 대해 데이터 블록과의 병합을 최대한 지연한다. 이를 통해 불필요한 데이터 블록의 병합 연산을 방지하여 플래시 메모리의 소거 연산 횟수를 크게 감소시켰고, 공간 활용을 극대화 하였다. 마지막으로, 로그 버퍼 관리 기법의 대표적인 기법인 BAST와 FAST와의 성능 비교를 통해 본 논문에서 제안하는 기법의 우수성을 증명하였다. 성능평가 결과 제안하는 기법이 BAST와 FAST에 비해 소거연산 측면에서 평균 25%와 65%의 성능 향상이 있었다.

■ 중심어 : | 플래시 메모리 | 로그 블록 | FTL |

Abstract

In this paper, we propose a new log buffer management scheme considering the accessibility of the data. Our proposed scheme evaluates the worth of the merge of log blocks. It conducts the merge operations between infrequently updated data and the data blocks and postpones as much as possible the merge operations between frequently updated data and the data blocks. As a result, the proposed method prevents the unnecessary merge operations, reduces the number of the erase operations, and improves the utilization of the flash memory storage. In order to show the superiority of the proposed scheme, we compare it with BAST and FAST. It is shown through performance evaluation that the proposed method achieves about 25% and 65% performance improvements over BAST and FAST on average in terms of the number of the erase operations.

■ keyword : | Flash memory | Log Blocks | FTL |

* 이 논문은 2009년 교육과학기술부(지역거점연구단육성사업/충북BIT연구중심대학육성사업단)와 한국연구재단의 지원을 받아 수행된 기초연구사업의 결과임.(No. 2009-0080279)

접수번호 : #090831-008

접수일자 : 2009년 08월 31일

심사완료일 : 2009년 12월 15일

교신저자 : 유재수, e-mail : yjs@cbnu.ac.kr

1. 서 론

최근 다양한 기기에서 NAND 플래시 메모리가 저장 장치로 널리 사용되고 있다. NAND 플래시 메모리는 비휘발성 기억장치로 하드디스크와 다르게 탐색시간이 빠르고 크기가 매우 작기 때문에 모바일 기기나 USB 드라이브와 같은 이동식 저장매체에서 광범위하게 사용되고 있다. 최근에는 하드디스크를 대체하는 새로운 저장매체로써 널리 이용되고 있다[1][9].

하지만, NAND 플래시 메모리는 데이터를 기록하기 전 해당 영역을 반드시 삭제해야 하는 "쓰기 전 삭제(erase before write)"라는 제약을 갖는다. 일반적으로 플래시 메모리에서의 한 섹터에 대한 읽기 연산은 25μs, 쓰기 연산은 300μs 그리고 삭제 연산은 1.5 ~ 2.0ms의 시간이 요구하고, 쓰기와 삭제의 단위가 다른 것과 같은 고유의 특성을 가진다[6].

그래서 하드디스크처럼 동일한 물리적 위치에 덮어쓰기를 수행할 경우 일반 연산에 비해 긴 시간을 요구하는 소거 연산에 따른 추가적인 비용이 요구된다.

이러한 특성으로 인한 제약을 극복하기 위해 주소 사상 소프트웨어인 FTL(Flash Translation Layer)이 필요하다. FTL은 논리적인 섹터 주소를 물리적인 데이터 위치로 변환하여 플래시 메모리가 디스크 같은 저장 장치로 보이도록 모방하는 소프트웨어 계층이다.

FTL은 파일 시스템으로 전환된 데이터의 논리 주소(logical address)와 플래시 메모리의 물리 주소(physical address)를 매핑하고 테이블로 메모리에 유지 시킨다. 플래시 메모리는 갱신된 데이터가 제자리에 기록될 수 없는 특성에 따라 페이지가 갱신되면 새로운 공간에 기록하고 기존 매핑 정보를 변경한다. 또한 FTL은 블록의 지우는 횟수를 평준화 시키는 마모도 평준화(wear-leveling)를 관리하는 기능을 가진다[2][3]. 플래시 메모리는 한 블록에 대한 지우기 연산이 일정 횟수를 넘어가면 그 블록이 손상되어 플래시 메모리가 손상될 경우가 있다.

FTL의 한 기법으로 플래시 메모리 내의 블록 일부를 로그 블록으로 지정하여 사용하는 기법들이 소개되었다[4][5][8]. 만약 데이터 블록에서 갱신이 발생하면 임시적으로 그 데이터 블록의 변경된 섹터의 데이터를 미리 지정된 임의 로그 블록에 저장한다. 만약 로그 블록이 가득 차면, 로그 블록에 기록된 각 갱신 섹터들과 갱신 전 데이터가 저장된 데이터 블록 대해 병합(merge) 연산을 수행한다. 병합 연산은 새로운 데이터 블록을 할당 받고 로그 블록과 데이터 블록의 가장 최신 데이터를 찾아내어 그 새로운 데이터 블록에 기록한다. 그리고 사용된 로그 블록과 이전의 데이터 블록에 대해서 소거 연산을 수행한다.

기존의 기법들은 병합 연산을 위한 로그 블록의 선택 시, 연관된 데이터 블록의 병합 가치에 대해서는 고려하지 않고 있다. 데이터 블록의 병합 가치는 추가 갱신 가능성을 나타낸다. 그래서 병합 가치가 높은 데이터 블록은 추가 갱신 가능성이 낮은 것을 의미하고, 병합이 되더라도 다시 로그에 기록될 가능성이 낮다. 병합 가치가 낮은 데이터 블록은 추가 갱신 가능성이 높은 것을 의미하고, 병합이 되더라도 다시 로그에 기록될 가능성이 높다. 즉, 병합 가치가 낮은 데이터 블록을 병합하게 되면 다시 로그 블록에 기록될 가능성이 높기 때문에 추가 병합이 요구된다.

본 논문에서는 추가 갱신 가능성을 고려한 새로운 로그 버퍼 관리 기법인 JBB(Join Between log Block)을 제안한다. JBB는 로그 블록이 포함하는 데이터 블록들의 가치를 평가하여 병합할 로그 블록을 선택하여 병합 연산을 수행한다. 그리고 로그 블록에 포함되는 모든 데이터 블록을 강제 병합하지 않고 데이터 블록의 병합 가치에 따라 로그 블록 상에 남겨 놓고 최대한 병합을 지연시킨다. 또한 한 로그 블록이 많은 데이터 블록과 연관이 되어 있는 경우 로그 블록들의 병합 가치를 명확하게 비교하기 위해 데이터 섹터의 지역성을 고려한 로그 블록 할당 기법을 제안한다. 그래서 제안하는 기법은 불필요한 데이터 블록의 병합 연산을 방지하여 플래시 메모리의 소거 횟수를 크게 감소시켰고, 공간 활용을 극대화 하였다.

본 논문의 구성은 다음과 같다. 2장에서는 기존 로그 버퍼 관리 기법을 분석한다. 3장에서는 제안하는 로그 블록 관리 기법을 기술한다. 4장에서는 제안하는 기법과 타 기법의 성능을 비교한다. 5장에서는 제안하는 기

법을 정리하고 결론을 맺는다.

2. 기존 로그 버퍼 관리 기법

2.1 BAST

BAST(Block-Associative Sector Translation)[4]는 데이터의 로그 블록과 데이터 블록을 일 대 일로 관리한다. 그래서 한 데이터 블록에서 갱신 연산이 발생할 경우 그 데이터 블록을 위한 로그 블록이 할당된다. 만약 데이터 블록을 위해 할당할 로그 블록이 없다면 로그 블록 중 하나를 희생 로그 블록으로 선정하여 병합 연산을 수행한다. 그래서 블록 매핑 기법과는 달리 일부 페이지가 자주 갱신된다고 해도 해당 로그 블록의 페이지를 모두 사용할 때까지 나머지 페이지에 대한 복사를 수행하지 않아도 된다. 하지만 로그 블록을 데이터 블록과 일 대 일로 할당하기 때문에 병합 연산 시, 최대 두 번의 소거와 한 블록에 수용 가능한 섹터 수만큼의 읽기/쓰기 작업이 발생한다. 또한 순차적인 데이터가 들어올 경우 매핑 테이블에 기록되어 있는 데이터 블록의 번호를 로그 블록과 스위치 시켜 병합 연산을 피할 수 있다. 하지만 BAST는 쓰기 연산의 패턴이 임의(random) 쓰기인 경우 잦은 병합 연산을 발생시켜, 연산 속도를 저하시키고 로그 블록의 활용률을 낮게 하는 단점이 있다.

[그림 1]은 BAST 기법에서 로그 블록에 데이터를 쓰는 과정을 보여준다. 5개의 데이터 블록 (B0~B5)과 최대 할당할 수 있는 로그 블록(L0~L3)이 4개라고 가정한다. P1, P5, P9, P13에 대한 갱신이 발생할 경우, 해당 데이터 블록의 갱신을 위한 로그 블록이 지정된다. 현재 섹터 갱신을 위한 매핑 테이블(B0:L0, B1:L1, B2:L2, B3:L3)이 구성되어 해당 블록에 대한 페이지 갱신은 지정된 로그 블록에서 수행한다. 이후 P16에 대한 변경을 요청한다면 기존 로그 블록 중 하나를 병합하고 새로운 로그 블록을 할당할 수 밖에 없다. 따라서 총 4개의 페이지 중 1개만 사용한 후 병합 연산을 수행해야 한다. 결국 해당 로그 블록은 25%의 낮은 이용률을 갖게 된다.

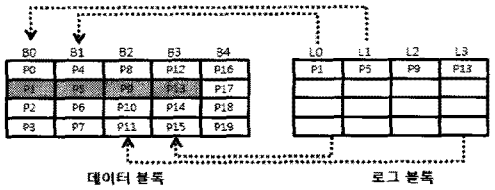


그림 1. BAST 기법

2.2 FAST

FAST(Fully Associative Sector Translation)[5]는 BAST의 로그 공간 활용률과 임의 쓰기에서의 단점을 보완하기 위해 제안되었다. FAST는 로그 블록을 하나의 순차 로그 블록과 다수의 임의 로그 블록으로 구분한다. 순차 로그 블록에는 순차 쓰기 연산으로 판단되는 논리 섹터가 기록된다. FAST는 임의 쓰기 연산 시, 데이터 블록에 대한 갱신을 해당 로그 블록에서 수행하는 것이 아니라 첫 번째 로그 블록부터 페이지를 채우게 된다. BAST에 비해 로그 블록의 공간 활용률을 극대화하였다. 그림 2는 FAST 기법에서 로그 블록에 데이터를 쓰는 과정으로 P1, P5, P9, P13, P17, P2, P6, P10의 순서로 갱신이 발생하였다. BAST 기법과는 달리 데이터 블록과 로그 블록간의 매핑을 수행하지 않고 첫 번째 로그 블록부터 데이터를 채워 간다. 이후 모든 로그 블록에 데이터가 가득 차게 되면 첫 번째 로그 블록부터 연관된 데이터 블록과 병합 연산을 수행한다. 하지만 다수의 데이터 블록에 속한 갱신된 섹터들이 하나의 로그 블록에 매핑되기 때문에 로그 블록에 대한 병합 연산 시, 많은 데이터 블록이 연관되는 문제가 발생한다. 즉 FAST는 로그 블록에 의해 참조되는 논리 주소의 지역성을 고려하지 않았다. 또한 자주 갱신되는 데이터에 의해 빈번하게 발생되는 병합연산을 고려하지 않았다.

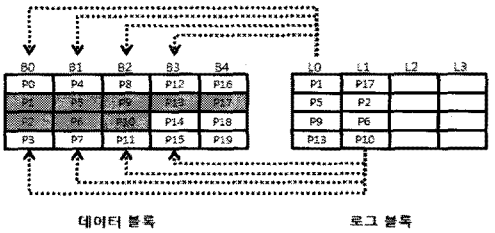


그림 2. FAST 기법

3. 제안하는 로그 버퍼 관리 기법

LRU(Least Recently Used) 정책에 따르면 최근 접근된 데이터는 다시 접근될 가능성이 높다. 만약 이러한 데이터에 대해 잦은 병합이 일어날 경우 그 데이터는 다시 갱신이 되고 그 데이터를 포함하는 데이터 블록이 다시 병합 연산을 수행하는 과정이 자주 일어나게 된다. 본 논문에서 제안하는 기법은 이러한 현상을 방지하기 위해 병합되는 로그 블록들 내에 병합 가치가 낮은 데이터 블록에 포함되는 섹터에 대해서는 로그 블록 간 병합 기법을 이용하여 해당 데이터 블록에 대한 병합 연산을 지연 시키는 기법을 제안한다.

3.1 Hot 데이터와 Cold 데이터의 구분

이 논문에서 제안하는 로그 관리 기법은 데이터의 갱신 빈도에 따라 데이터를 관리 한다. 데이터는 갱신 빈도에 따라 hot 데이터(자주 갱신되는 데이터)와 cold 데이터(자주 갱신되지 않는 데이터)로 구분 한다. 또한, 데이터는 쓰기 연산 패턴에 따라 구분 된다. hot 데이터는 어떤 임의 쓰기 패턴에서 빈번하게 갱신 되는 데이터를 말한다. cold 데이터는 빈번하게 갱신이 일어 나지 않는 데이터를 말한다. hot 데이터는 빈번하게 갱신되기 때문에, 다시 갱신되어 로그 블록에 기록될 가능성이 크다. 그래서 로그 블록과 데이터 블록 간 불필요한 병합 연산이 자주 발생한다. 이러한 이유로 hot 데이터는 최대한 로그 블록에 유지 시켜 불필요하게 발생하는 데이터 블록 간의 병합 연산을 줄여야 한다. 또한 cold 데이터는 빈번하게 갱신되지 않는 데이터이기 때문에 짧은 시간 안에 다시 로그 블록에 기록될 가능성이 낮다. 그래서 로그 블록에 유지 시키지 않고 데이터 블록 간 병합 연산을 해야 한다.

3.2 데이터 블록 병합 가치 평가

제안하는 기법은 데이터 블록의 병합 가치를 평가하기 위해 갱신 시간을 나타내는 기록 순서 값 SO(Sequence Order)를 유지한다. SO는 전역으로 관리되고 갱신 요청이 발생할 때마다 1씩 증가하는 값이다. 로그 블록에 데이터 섹터가 갱신되어 기록될 때 그 섹터 정보와 함께 SO 값을 기록하고 1을 증가 시킨다. 이렇게 표현된 시간 정보를 이용하여 로그 블록 상의 각 데이터 섹터들이 다른 섹터들과의 상대적인 갱신 시간을 알아낸다. 그리고 로그 블록 상에서 동일한 데이터 블록과 연관된 데이터 섹터들의 평균 접근 시간을 이용하여 상대적인 병합 가치를 평가한다. 식 1은 임의의 데이터 블록 DB의 병합 가치를 평가 하기위한 평균 접근 시간 APU(DB)의 계산을 나타낸다. LP는 로그 블록 상의 로그 섹터, SUP(DB)는 DB와 연관된 로그 섹터들의 셋, SO(LP)는 로그 섹터 LP의 기록 순서 값, SOcur은 기록 순서의 현재 값, 그리고 NUP(DB)는 데이터블록 DB와 연관된 로그 섹터들의 수이다.

$$APU(DB) = \sum_{LP_i \in SUP(DB)} \left(\frac{SO_{cur} - SO(LP_i)}{NUP(DB)} \right) \quad (1)$$

LP : 로그 블록 상의 로그 섹터
SUP(DB) : 데이터 블록과 연관된 로그 섹터들의 셋
SO(LP) : 로그 섹터의 기록 순서 값
SOcur : 기록 순서의 현재 값
NUP(DB) : 데이터 블록과 연관된 로그섹터들의 수

$$MSDB(DB) = \frac{1}{\log(AUP(DB))} \quad (2)$$

최근 SO 값의 상대적인 차이는 병합 가치에 큰 영향을 미치지만 오래된 SO 값의 상대적인 차이는 병합 가치에 큰 영향을 미치지 않는다. 이를 반영하기 위해 로그를 이용하여 데이터 블록에 병합 가치를 부여한다. 식 2는 데이터 블록의 최종 병합 가치를 나타낸다. 수식에 따라 데이터 블록들의 MSDB 값을 비교하여 상대적으로 작은 값을 가지는 데이터 블록이 다른 데이터 블록보다 높은 병합 가치를 가진다.

[그림 3]은 데이터 블록의 병합 가치를 평가하여 병합 가치가 높은 데이터 블록을 선정 하는 과정을 예로 보여 준다. DB0 와 DB1 의 병합 가치를 평가 하는 경우 DB0 의 NUP(DB)는 3이 되고, DB1의 NUP(DB)는 1이 되고, SOcur 는 6이 된다. DB0 와 DB1 의 MSDB는 수식 1과 수식 2에 의해 각각 8.07, 3.32가 된다. 병합될 블록 선정 기준은 상대적으로 MSDB가 작은 값을 가지는 데이터 블록이 선택되기 때문에 [그림 3]에서는 DB1이 병합 가치가 높은 블록으로 선정 된다.

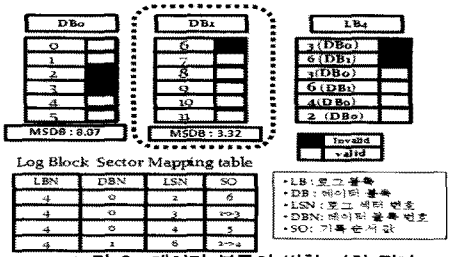


그림 3. 데이터 블록의 병합 가치 평가

3.3 로그 블록의 병합 가치 평가

모든 로그 블록이 가득 채워진 경우 로그 블록 중 병합될 로그 블록을 선정해야 한다. 이때 로그 블록의 병합 가치 평가를 통해 병합할 로그 블록을 선택한다. 로그 블록의 병합 가치는 그 블록이 가지는 각 섹터들의 데이터 블록 병합 가치와 이전에 기록되었지만 다시 갱신되어 무효화된 섹터들의 수를 이용하여 계산한다.

$$MSLB(LB)\sum_{i=0}^{N_{sectors}}(MSDB(ADB(LP_i))+\alpha \cdot N_{invalid}(LB) \quad (3)$$

식3에서 로그 블록 LB의 병합 가치 MSLB (LB)를 계산한다. Nsectors는 한 블록이 가지는 섹터의 개수, Ninvalid(LB)는 로그 블록 LB의 무효화된 섹터의 개수, 그리고 ADB(LP)는 로그섹터LP에 연관된 데이터 블록을 나타낸다. 만약 LPi가 무효화된 로그 섹터일 경우 MSDB(ADB(LPi))의 값은 0이다. α는 무효화된 로그 섹터가 병합 가치에 미치는 영향을 표현하기 위한 상수이다. 4장에서 α값의 변화에 따른 성능평가를 수행하였다.

[그림 4]는 로그 블록의 병합 가치를 평가하여 로그 블록의 병합 가치가 높은 블록 2개를 선정 하는 과정이다. 병합될 로그 블록을 선정하는 과정이다. 각 로그 블록의 병합 가치가 $LB_1=5$, $LB_2=4$, $LB_3=0$, $LB_4=3$, $LB_5=3$일 경우 병합 가치가 가장 높은 LB_1, LB_2가 병합될 로그 블록으로 선정된다.

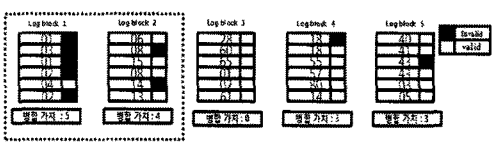


그림 4. 로그 블록의 병합 가치 평가

3.4 병합 가치를 고려한 로그 블록 간 병합 기법

3.3장에서의 상대적인 병합 가치 비교를 통해 N개의 병합할 로그 블록을 선정하게 되면 그 로그 블록과 연관된 데이터 블록과 병합 연산을 수행 한다. 이때 모든 연관된 데이터 블록과 병합 연산을 수행하지 않고 상대적으로 병합 가치가 높은 데이터 블록들과 병합 연산을 수행한다. 연관된 데이터 블록의 병합 가치가 낮아 병합을 수행하지 못한 로그 섹터를 가지는 로그 블록들은 모두 병합되고 다시 로그 블록으로 쓰여 진다. 이로 인해서 병합 가치가 낮은 데이터 블록과 연관된 로그 섹터들의 병합 시기가 연장되고 불필요한 데이터 블록과의 연산 비용을 줄이고 블록 소거 비용을 줄인다. 다시 말하면 로그 블록 가치 평가에 의한 병합 로그 블록 선정과 데이터 블록 가치 평가에 의한 데이터 블록의 병합 연장을 통해 자주 갱신이 일어나는 데이터 블록에 대한 불필요한 병합을 방지한다.

[그림 5]는 로그 블록들 간의 병합을 통해 병합이 연장된 로그 섹터들과 병합이 수행된 로그 섹터들을 보여준다. 3.2장과 3.3장을 통해 로그 블록 1과 2가 병합될 로그 블록으로서 선택되었다고 가정한다. 그리고 해당 로그 블록과 연관된 데이터 블록이 로그 블록 아래 표시되어 있다. 선정된 이 두 로그 블록은 데이터 블록 0, 1, 2와 연관되어 있다. 연관된 데이터 블록들의 가치 비교를 통해 병합할 데이터 블록을 선택한다. 예제에서는 데이터 블록 1이 다른 데이터 블록 0과 2에 비해 병합 가치가 높다고 가정한다. 그래서 데이터 블록 1과 로그 블록 1과 2의 연관된 로그 섹터들과 병합을 수행한다. 그리고 로그 블록 1과 2를 병합하여 로그 블록 6을 새롭게 생성한다. 로그 블록 6에는 병합이 연장된 데이터 블록 0과 2에 연관된 로그 섹터가 기록된다.

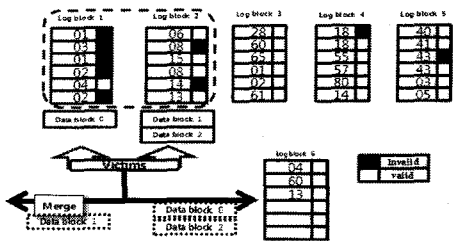


그림 5. 로그 블록 병합 기법

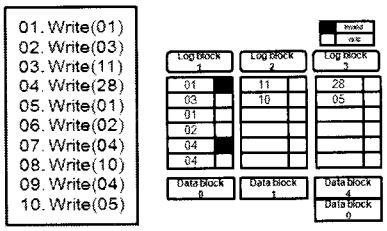
3.5 로그 블록의 지역성을 위한 전략

다수의 데이터 블록의 데이터 로그들이 하나의 로그 블록에 기록 된다면, 로그 블록에 대한 병합 연산 시, 불필요하게 많은 데이터 블록이 병합 연산에 연관되는 문제가 발생한다. 이를 해결하기 위해 본 논문에서는 로그 블록의 지역성을 위한 전략을 제안한다.

제안하는 전략은 기본적으로 한 데이터 블록에 관련된 로그 데이터들을 한 로그 블록에 기록하겠다는 것이다. 그래서 한 데이터 블록에 데이터들이 다수의 로그 블록에 분산되지 않도록 한다. 이를 로그 블록의 관점에서 보면 한 로그 블록이 다수의 데이터 블록과 연관되지 않고 단지 몇몇 데이터 블록들과 연관 되어 지기 때문에 병합 연산 수행 시, 불필요하게 많은 데이터 블록이 병합에 참여하는 상황을 방지할 수 있다.

일단 한 데이터 블록의 로그 데이터가 발생하면 그 후로 발생하는 로그 데이터에 대해 최대한 그 로그 블록에만 기록되게 한다. 만약 로그 블록이 가득 차면 병합 연산을 최대한 지연시키기 위해 추가로 다른 로그 블록을 선정하여 기록 하게 된다. 새로운 데이터 블록의 로그 데이터가 발생하게 되면 로그 블록들 간에 남은 공간의 가치 평가를 수행하여 가장 여유가 많은 로그 블록을 할당한다. 이러한 공간 가치 평가는 로그 블록 내 사용되지 않은 빈 섹터 수를 로그 블록 내 할당 받은 데이터 블록 수로 나누어 그 값이 가장 큰 로그 블록을 선정하게 된다. 이는 해당 로그 블록 내에 한 데이터 블록 당 남은 빈 섹터 수를 계산하기 위한 방식이다.

[그림 6]은 로그 블록의 공간 가치 평가를 통해 로그 블록에 데이터를 기록하는 예이다. [그림 2]에서 각 블록은 총 6개 섹터를 가지며, 논리 주소는 1부터 시작 한다. 로그 블록의 수는 3개로 제한한다. 덮어쓰기 연산에서 오른쪽 괄호 안 숫자는 연산이 요청된 논리 섹터의 주소를 의미하며 왼쪽의 숫자는 덮어쓰기 연산 순서를 의미한다. 로그 블록의 섹터에 기록된 숫자는 덮어쓰기 순서를 의미한다. 데이터 블록은 논리 섹터를 한 블록 당 섹터 수로 나누어 얻어진 몫으로 결정한다. 10번의 갱신 연산이 발생한 경우 로그 블록에 기록하는 9번째 연산 후, 1번 로그 블록이 가득 차게 된다. 10번째 갱신 연산은 데이터 블록 0번에 해당하고, 데이터 블록 0번의 데이터들이 기록될 로그 블록이 가득 채워졌기 때문에 새로운 로그 블록을 할당해야 한다. 새로운 로그 블록은 로그 블록들의 남는 공간의 가치 평가를 통해 선정 된다.



(a) 덮어쓰기 연산 (b) (a)연산 후 로그 블록 상태

그림 6. 공간 가치 평가를 통한 로그 블록 할당

로그 블록 2의 공간 가치 평가 결과 값은 4가 되고 로그 블록 3의 공간 가치 평가 결과 값은 5가 된다. 가치 평가 값이 큰 로그 블록 3이 0번 데이터 블록에 해당하는 로그 블록으로 선정 되고, 0번 데이터 블록의 갱신이 있는 경우 새로 할당 받은 로그 블록 3이 가득 채워질 때까지 기록하게 된다. 이 같은 기법을 사용하여 모든 로그 블록들을 가득 채운다.

4. 성능 평가

제안 하는 기법의 우수성을 보이기 위해 금융 시스템의 IO 연산을 Trace한 데이터[10]를 바탕으로 다른 기법과 비교 평가를 수행 하였다. 실험은 Pentium-4 PC시스템과 2GB RAM 그리고 120G HDD에서 수행 하였다.

실험은 한 블록 당 64개의 섹터를 갖고 각 섹터가 2K의 크기를 갖는 플래시 메모리 파일 시스템을 가정하여 수행하였다[7]. 실험에서 각 로그 블록은 고정된 크기를 할당 받아 사용한다고 가정하였다. 실험에서는 BAST와 FAST를 비교 평가하였다. 본 논문에서 제안하는 기법은 JBB로 표기하였다.

BAST와 FAST와 성능을 비교하기 위해 로그 블록 수를 변경하면서 평가를 수행하였다. 또한 본 논문에서 제안하는 기법의 특징을 보여주기 위해 한 번에 병합하

는 로그 블록의 개수와 병합 연기 데이터 블록 비율을 변경하면서 평가를 수행하였다. 기본적으로 로그 블록의 수는 128개로 설정하여 로그 블록 병합 시, 병합 가치가 가장 높은 로그 블록 6개를 선정하여 병합 연산을 수행한다. 또한 병합 연기 데이터 블록 비율은 30%로 설정하여 로그 블록 병합 시 연관된 데이터 블록들의 중에서 블록 가치가 가장 낮은 30%의 데이터 블록을 선택하여 그 데이터 블록에 해당하는 로그 데이터를 데이터 블록과 병합하지 않고 새로운 로그 블록을 할당하여 남겨 놓는다.

식 3에서 적절한 α값을 결정하기 위해 α값의 변화에 따른 성능 평가를 수행하였다. [그림 7]은 성능 평가의 결과를 나타낸다. α값은 무효화된 로그 섹터를 로그 블록의 병합가치 평가 시에 어느 정도의 비중을 줄 것인지 결정하는 값이다. 만약 이 값이 커지면 무효한 로그 섹터가 그 로그섹터를 포함하는 로그 블록의 병합 가치를 감소시키기 때문에 성능이 급격하게 떨어진다. 이는 [그림 7]에서 α값이 0.1보다 커지면서 소거횟수가 급격하게 증가하는 것으로서 보여준다. 또한 이 값이 너무 작아지게 되면 α값이 병합 가치에 미치는 영향이 너무 커서 무효한 로그 섹터의 수가 로그 블록의 병합가치를 평가하는 유일한 값이 된다. 실험 결과 이러한 경우에도 성능이 저하되는 현상이 발생하였고 α값이 -0.1보다 작은 값을 갖게 될 경우에 이러한 현상이 나타난다. 실험 결과 α값이 -0.01일 경우 가장 적은 소거 연산이 발생하였다. 그래서 성능 평가에서 α값을 -0.01로 지정하였다.

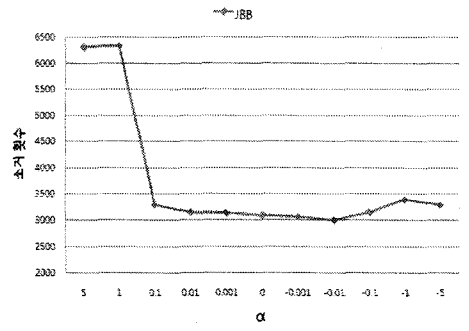


그림 7. 병합 로그 블록 수에 따른 성능 비교

FAST와 BAST의 성능을 비교하기 위해 로그 블록의 수를 32에서 512로 변경하여 비교를 수행하였다. JBB는 BAST에 40%에서 10%, FAST에 비해 90%에서 40%의 소거 연산을 수행하였다. 그림 8은 로그 블록 수에 따른 비교를 나타낸다. α값은 -0.01, 연기 데이터 블록 비율은 30%, 병합 로그 블록 수는 6개로 설정하여 실험 하였다. 로그 블록을 32개로 설정 하였을 때, 가장 많은 소거 연산이 발생 하였고, 로그 블록 수가 커질수록 그 소거 연산이 줄어드는 것을 볼 수 있다. 로그 블록을 많이 설정하면 그 만큼 갱신 데이터를 로그 블록에 많이 적을 수 있기 때문에 병합 연산을 지연 시킬 수 있다. 병합 연산의 감소로 인해 블록의 삭제 연산 또한 줄어드는 결과를 가져온다. 하지만 갱신 데이터를 위해 로그 블록을 너무 많이 늘리는 경우 그 만큼 기록 할 수 있는 데이터 공간이 작아지는 결과를 가져 올 수 있다.

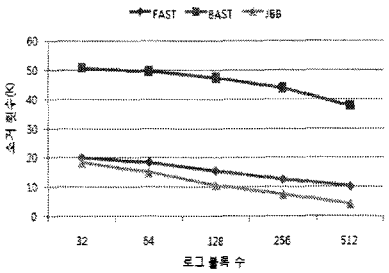


그림 8. 로그 블록 수에 따른 성능 평가 비교

[그림 9]는 병합 로그 블록 수에 따른 성능을 나타낸다. 병합 로그 블록의 수가 너무 작거나 또는 너무 큰 경우 상대적으로 성능이 좋지 않았다. 이 실험에서는 병합 블록에 대한 지연 비율을 35%로 설정하고 실험을 하였다. 병합 대상 로그 블록의 수가 너무 작거나 큰 경우에도 지연을 35% 비율로 시키기 때문에 hot data의 병합을 지연 시키지 못하거나, cold data를 병합 시키지 않고 지연 시키는 경우가 생긴다. 그래서 병합 로그 블록의 수에 따라 'U' 형태의 소거 횟수 패턴을 보인다. 결론적으로 병합 로그 블록의 수는 너무 작거나 크지 않도록 중간 정도의 병합 로그 블록 수를 가져야 한다.

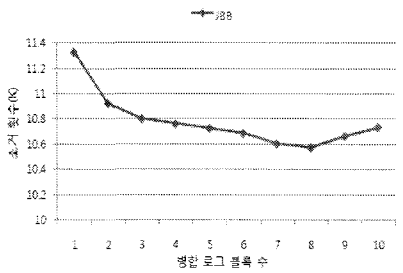


그림 9. 병합 로그 블록 수에 따른 성능 비교

로그 데이터의 병합 지연 데이터 블록 비율에 따른 블록 소거 횟수 비교를 수행 하였다. 로그 블록의 병합 연산 수행 시 해당 로그 블록 내에 포함된 데이터 블록들 중 병합 가치가 낮은 데이터 블록의 로그 데이터는 로그 블록에 다시 기록 하여 병합을 수행하지 않는다. 이때 병합이 지연된 데이터 블록을 병합 지연 데이터 블록이라고 한다. [그림 10]은 병합 지연 데이터 블록의 비율에 따른 성능 비교를 수행한 결과를 나타낸다. 만약 병합 지연 비율이 적으면 hot 데이터에 속하는 데이터 블록의 병합이 수행 되기 때문에 소거 횟수가 상대적으로 많아진다. 반대로 병합 지연 비율이 너무 크면 cold 데이터 임에도 불구하고 로그 블록에 여전히 남기기 때문에 로그 블록 공간 활용의 저하로 인해 소서 횟수가 증가 한다. [그림 10]에서는 0%~30% 구간에서는 30%~70% 구간에 비해 상대적으로 소거 횟수가 큰 것을 보여 준다. 또한 75% 이후부터는 소거 횟수가 급격하게 증가 하는 것을 보여준다.

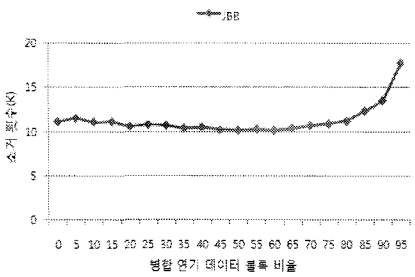


그림 10. 병합 연기 데이터 블록 비율에 따른 성능 비교

5. 결 론

본 논문에서는 로그 블록이 포함하는 데이터 블록들의 가치를 평가하여 병합할 로그 블록을 선택하여 병합 연산을 수행하는 효과 적인 로그 버퍼 관리 기법을 제안하였다. 제안하는 기법은 로그 블록에 포함되는 모든 데이터 블록을 강제 병합하지 않고 데이터 블록의 병합 가치에 따라 로그 블록 상에 남겨 놓고 최대한 병합을 지연 시킨다. 이를 통해 BAST에 비해 40%에서 10%, FAST에 비해 90%에서 40%의 소거 연산을 수행 하였다. 제안 하는 기법으로 인해 불필요한 데이터 블록의 병합 연산을 방지하여 플래시 메모리의 소거 횟수를 크게 감소시켰고, 공간 활용을 극대화 하였다. 향후 연구는 다양한 저장 장치 접근 패턴에 따른 성능 평가를 수행할 것이고 데이터베이스와 같은 특수 프로그램을 위한 플래시 메모리 관리 기법을 연구할 것이다.

참 고 문 헌

[1] E. Gal and S. Toledo "Algorithms and data structures for flash memories," ACM Comput. Surv, Vol.37, No.2. pp.138-16, 2005.

[2] S. E. Wells and C. H. Calif, "Method for wear leveling in a flash EEPROM memory," United States Patent, No.5,341,339, 1994.

[3] S. W. Han, ""Flash memory wear leveling system and method," United States Patent, No. 6,016,275, 2000.

[4] J. S Kim, J. M. Kim, S. H. Noh, S. L. Min, and Y. K. Cho, "A space-efficient flash translation layer for Compact Flash systems," IEEE Transactions on Consumer Electronics, pp.366-375, 2002(5).

[5] S. W. Lee, D. J. Park, S. W. Lee, T. S. Chung, 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 Issue 3, 2007(7).

[6] SAMSUNG ELECTRONICS. Nand flash memory & smartmedia data block, 2005.

[7] SAMSUNG ELECTRONICS, Nand Flash memory, K9F2G08R0B data book, 2007.

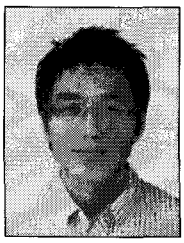
[8] B. S. KIM and G. Y. LEE, "Method of driving remapping in flash memory and flash memory architecture suitable therefore," United States Patent, No.6,381,176, 2002(4).

[9] C. Park, J. Seo, D. Seo, S. Kim, and B. Kim, "Cost-efficient memory architecture design of nand flash memory embedded systems," In Proccedings of the 21st International Conference on Computer Design(ICCD '03), pp.474-480, 2003(10).

[10] http://traces.cs.umass.edu/index.php/Storage/Storage.

저 자 소 개

김 학 철(Hak-Chul Kim) 준회원



- 2008년 2월 : 충북대학교 정보통신공학과(공학사)
- 2008년 3월 ~ 현재 : 충북대학교 정보통신공학과 석사과정

<관심분야> : DB 시스템, 센서 네트워크, 저장시스템, 파일시스템

박 용 훈(Yong-Hun Park) 준회원

- 2005년 2월 : 호원대학교 정보통신공학과(공학사)
- 2007년 2월 : 충북대학교 정보통신공학과(공학석사)
- 2009년 2월 ~ 현재 : 충북대학교 정보통신공학과 박사과정

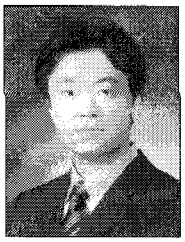
<관심분야> : DB 시스템, 정보검색, 시공간 데이터베이스, 파일시스템, 위치기반서비스

윤 종 현(Jong-Hyeong Yun) 정회원

- 2003년 2월 : 충북대학교 정보통신공학과(공학사)
- 2005년 2월 : 충북대학교 정보통신공학과(공학석사)
- 2009년 2월 : 충북대학교 정보통신공학과(공학박사)
- 2009년 3월 ~ 현재 : 한국전자통신연구원 지식e러닝 연구팀

<관심분야> : DBMS, 저장시스템, 시공간색인구조, 이동객체, 센서 네트워크

서 동 민(Dong-Min Seo) 정회원



- 2002년 2월 : 충북대학교 정보통신공학과(공학사)
- 2004년 2월 : 충북대학교 정보통신공학과(공학석사)
- 2008년 2월 : 충북대학교 정보통신공학과(공학박사)
- 2008년 3월~현재 : 한국과학기술원 전산학과 연수연구원.

<관심분야> : 데이터베이스 시스템, 에이전트 시스템, XML, 이동 객체 데이터베이스, 시공간 색인 구조, 센서 네트워크

송 석 일(Suk-Il Song) 정회원

- 1998년 2월 : 충북대학교 정보통신공학과(공학사)
- 2000년 2월 : 충북대학교 정보통신공학과(공학석사)
- 2003년 2월 : 충북대학교 정보통신공학과(공학박사)
- 2003년 8월 ~ 현재 : 충주대학교 컴퓨터공학과 교수

<관심분야> : 데이터베이스 시스템, 센서 네트워크, 이동 객체 데이터베이스, 저장 관리 시스템, XML

유 재 수(Jae-Soo Yoo) 종신회원

- 1989년 2월 : 전북대학교컴퓨터공학과(공학사)
- 1991년 2월 : 한국과학기술원 전산학과(공학석사)
- 1995년 2월 : 한국과학기술원 전산학과(공학박사)
- 1995년 3월 ~ 1996년 8월 : 목포대학교 전산통계학과(전임강사)
- 1996년 8월 ~ 현재 : 충북대학교 전기전자컴퓨터공학부 및 컴퓨터정보통신연구소 교수

<관심분야> : 데이터베이스시스템 정보검색 센서네트워크 및 RFID, 멀티미디어데이터베이스, 분산객체컴퓨팅