로그 블록 간 병합을 이용한 효율적인 로그 버퍼 관리

An Efficient Log Buffer Management Through Join between Log Blocks

김학철, 박용훈, 윤종현*, 서동민**, 송석일***, 유재수
충북대학교, 한국전자통신연구원*,
한국과학기술원**, 충주대학교***

Kim hak-cheol, Park youg-hun, Yun jong-hyeon*,
Seo dong-min**, Song seok-il***, Yoo jae-soo
Chungbuk National University,
Electronic and Telecommunications
Research Institute*,
Korea Advanced Institute of Science and
Technology**,
Chungju National University***

요약

플래시 메모리는 이미 데이터가 기록된 섹터에 대해 덮어 쓰기 연산이 되지 않는 특징이 있다. 이러한 플래시 메모리의 특징을 극복하기 위해 로그 버퍼 관리 기법이 소개 되었다. 그러나 현재 까지 연구된 로그 버퍼 관리 기법들 중 BAST 기법은 쓰기 연산의 패턴이 임의 쓰기인 경우 잦은 병합 연산을 발생시키는 문제가 있으며, 이를 개선한 FAST 기법은 자주 갱신되는 데이터에 의해 빈번하게 발생되는 병합 연산을 고려하지 않았다. 본 논문에서는 새로운 로그 버퍼 관리 기법인 JBB를 제안한다. 제안하는 기법은 로그 블록의 병합 가치를 평가하여 빈번하게 갱신이 발생하지 않는 데이터에 대해서 데이터 블록과의 병합연산을 수행하고, 빈번하게 갱신되는 데이터에 대해 데이터 블록과의 병합을 최대한 지연한다. 이를 통해 불필요한 데이터 블록의 병합 연산을 방지하여 플래시 메모리의 소거 횟수를 크게 감소시켰고, 공간 활용을 극대화 하였다. 로그 버퍼 관리 기법의 대표적인 기법인 BAST와 FAST와의 성능 비교를 통해 본 논문에서 제안하는 기법의 우수성을 증명하였다.

Abstract

Flash memory has rapidly deployed as data storage. However, the flash memory has a major disadvantage that recorded data cannot be dynamically overwritten. In order to solve this "erase-before-write" problem, the log block buffer scheme used Flash memory file system. however, the current managements of the log buffer, in case random write pattern, BAST technique have problem of frequent merge operation, but FAST technique don't consider merge operation by frequently updated data. Previous methods not consider merge operation cost and frequently updated data. In this paper, we propose a new log buffer management scheme, called JBB. Our proposed method evaluates the worth of the merge of log blocks, so we conducts the merge operation between infrequently updated data and its data blocks, and postpone the merge operation between frequently updated data and its data blocks. Through the method, we prevent the unnecessary merge operations, reduce the number of the erase operation, and improve the utilization of the flash memory storage. We show the superiority of our proposed method through the performance evaluation with BAST and FAST.

Ⅰ. 서론

최근 다양한 기기에서 NAND 플래시 메모리가 저장 장치로 널리 사용되고 있다. NAND 플래시 메모리는 비

이 논문은 2009년 교육과학기술부(지역거점연구단육성사업/충북BIT연구중심대학육성사업단)와지식경제부의 특구연구개발사업으로부터 지원받아 수행된 연구임.

휘발성 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].

하지만, NAND 플래시 메모리는 데이터를 기록하기 전 해당 영역을 반드시 삭제해야 하는 "쓰기 전 소거(erase before write)" 라는 특징을 갖는다. 일반적으로 플래시 메모리에서의 한 섹터에 대한 읽기 연산은 25 ㎲, 쓰기 연산은 300㎲ 그리고 소거 연산은 1.5 ~ 2.0 ㎳의 시간이 요구된다.

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

NAND 플래시 메모리를 저장매체로 사용하는 기기들은 기존 파일 시스템과 NAND 플래시 메모리 사이에 FTL(Flash Translation Layer)이라는 특수한 소프트웨어 계층을 사용한다. FTL의 한 기법으로 플래시 메모리 내의 블록 일부를 로그 블록으로 지정하여 사용하는 기법들이 소개되었다[2,3]. 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 기법들[4][5]은 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. 다시 말해서 병합가치가 낮은 데이터 블록을 병합하게 되면 다시 로그 블록에 기록될 가능성이 높기 때문에 불필요한 병합이다.

그래서 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 포함하는 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 본 논문의 대상인 로그 블록 기법과 기존 관련 기법을 설명한다. 3장에서는 제안하는 로그 블록 관리 기법을 기술한다. 4장에서는 제안하는 기법과 타 기법의 성능을 비교한다. 5장에서는 제안하는 기법을 정리하고 결론을 맺는다.

Ⅱ. 관련연구

효과적인 로그 블록의 관리를 위해 BAST[4]와 FAST[5]가 제안되었다. BAST(Block Associative Sector Translation)는 로그 블록과 데이터 블록을 일대 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. 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(switch)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.

FAST(Fully Associative Sector Translation)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 BAST에 비해 임의 쓰기 연산 시, 로그 블록의 공간 활용률을 극대화하였다. 하지만 다수의 데이터 블록에 속한 갱신된 섹터들이 하나의 로그 블록에 매핑 되기 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.

Ⅲ. 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 병합연산을 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.

본 장에서는 제안하는 로그 버퍼 관리 기법에 대해 설명한다. 우선 데이터 블록의 병합 가치 평가에 대해서 기술하고, 로그 블록의 병합 가치 평가에 대해서 기술한다. 그리고 병합 가치를 이용한 로그 블록 간 병합 기법을 설명한다. 마지막으로 데이터 로그의 지역성을 고려한 로그 블록 할당 기법에 대해서 기술한다.

1. 데이터 블록 병합 가치 평가

데이터 블록의 병합 가치를 평가하기 위해 갱신 시간을 나타내는 기록 순서 값 *SO*(Sequence Order)를 유지한다. *SO*는 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 *DB*의 병합 가치를 평가하기 위한 평균 접근 시간 *APU(DB)*의 계산을 나타낸다. *LP*는 로그 블록 상의 로그 섹터, *SUP(DB)*는 데이터 블록 *DB*와 연관된 로그 섹터들의 셋, *SO(LP)*는 로그 섹터 *LP*의 기록 순서 값, SO_{cur}은 기록 순서의 현재 값, 그리고 *NUP(DB)*는 데이터 블록 *DB*와 연관된 로그 섹터들의 셋이다.

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

최근 *SO* 값의 상대적인 차이는 병합 가치에 큰 영향을 미치지만 오래된 *SO* 값의 상대적인 차이는 병합 가치에 큰 영향을 미치지 않는다. 이를 반영하기 위해 log를 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.

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

2. 로그 블록 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. 식 3에서 로그 블록 *LB*의 병합 가치 *MSLB(LB)*를 계산한다. $N_{soctors}$는 한 블록이 가지는 섹터의 개수, $N_{invalid}(LB)$는 로그 블록 *LB*의 무효화된 섹터의 개수, 그리고 *ADB(LP)*는 로그 섹터 *LP*에 연관된 데이터 블록을 나타낸다. 만약 LP_i가 무효화된 로그 섹터 일 경우 $MSDB(ADB(LP_i))$의 값은 0이다. α 는 무효화된 로그 섹터가 병합 가치에 미치는 영향을 표현하기 위한

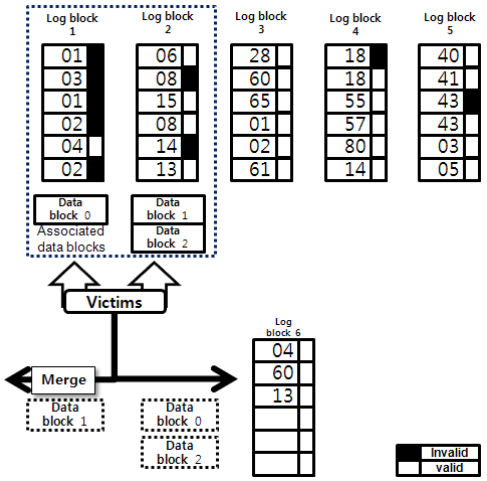
상수이다. 본 논문에서는 그 값을 1로 설정하여 성능 평가를 수행하였다.

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

3. 병합 가치를 고려한 로그 블록 간 병합 기법

3.2장에서의 상대적인 병합 가치 비교를 통해 N개의 병합할 로그 블록을 선정하게 되면 그 로그 블록과 연관된 데이터 블록과 병합 연산을 수행한다. 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 1은 로그 블록들 간의 병합을 통해 병합이 연장된 로그 섹터들과 병합이 수행된 로그 섹터들을 보여준다. 3.1장과 3.2장을 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 1에 비해 병합 가치가 높다고 가정한다. 그래서 데이터 블록 1과 로그 블록 1과 2의 연관된 로그 섹터들과 병합을 수행한다. 그리고 나서 로그 블록 1과 2를 병합하여 로그 블록 6을 새롭게 생성한다. 로그 블록 6에는 병합이 연장된 데이터 블록 0과 2에 연관된 로그 섹터가 기록된다.



▶▶ 그림 1. 로그 블록 병합 기법

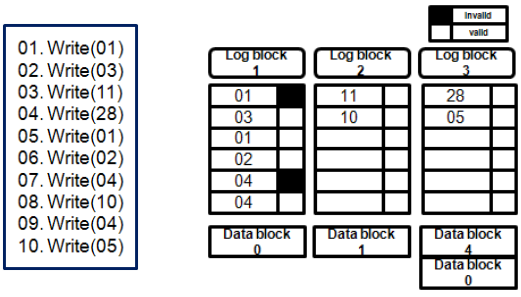
4. 로그 블록의 지역성을 위한 전략

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 2는 로그 블록의 공간 가치 평가를 통해 로그 블록에 데이터를 기록하는 예이다. 그림 2에서 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 기치 평가를 통해 선정 된다.



(a) 덮어쓰기 연산 (b) (a)연산 후 로그 블록 상태

▶▶ 그림 2. 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.

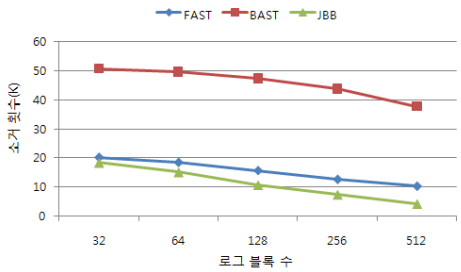
Ⅳ. 성능 평가

제안하는 기법의 우수성을 보이기 위해 Diskmon을 사용하여 일반 파일 시스템의 IO연산을 Trace한 데이터를 바탕으로 다른 기법과 비교 평가를 수행하였다. 실험은 Pentium-4 PC시스템과 2GB RAM 그리고 120G HDD에서 수행 하였다. 실험에 사용된 Trace 데이터는 Window XP환경에서 NTFS 파일 시스템에서 발생한 IO 연산을 Trace하여 구하였다.

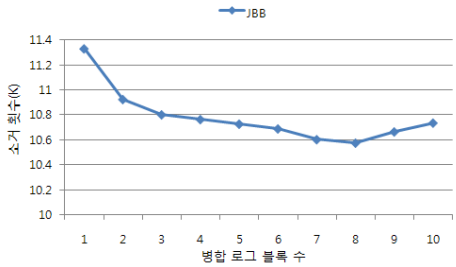
실험은 한 블록 당 64개의 섹터를 갖고 각 섹터가 2K의 크기를 갖는 플래시 메모리 파일 시스템을 가정하여 수행하였다. 실험에서 각 로그 블록은 고정된 크기를 할당 받아 사용한다고 가정하였다. 실험에서는 BAST 와 FAST를 비교 평가하였다. 본 논문에서 제안하는 기법은 JBB로 표기하였다.

BAST와 FAST와 성능을 비교하기 위해 로그 블록 수를 변경하면서 평가를 수행하였다. 또한 본 논문에서 제안하는 기법의 특징을 보여주기 위해 한 번에 병합하는 로그 블록의 개수와 병합 연기 데이터 블록 비율을 변경하면서 평가를 수행하였다. 식 3에서의 α 값은 1로 설정 하였다. 기본적으로 병합을 수행하는 로그 블록의 수는 5로 설정하여 로그 블록 병합 시, 병합 가치가 가장 높은 로그 블록 5개를 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.

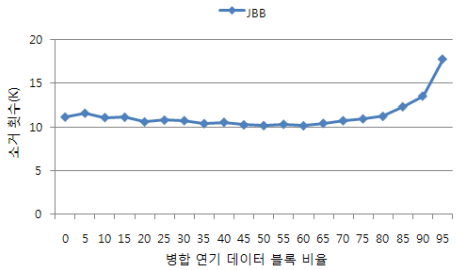
FAST와 BAST의 성능을 비교하기 위해 로그 블록의 수를 32에서 512로 변경하여 비교를 수행하였다. JBB는 BAST에 40%에서 10%, FAST에 비해 90%에서 40%의 소거 연산을 수행하였다. 그림 3은 로그 블록 수에 따른 비교를 나타낸다. 병합을 수행하는 로그 블록의 수가 너무 작거나 또는 너무 클 경우 상대적으로 성능이 좋아지지 않는 것을 그림 4에서 보여준다. 또한 로그 블록 병합 시 로그 데이터와 병합을 연기하는 데이터 블록의 비율이 너무 크면 성능이 소거 연산의 횟수가 급격히 증가하는 것을 그림 5에서 보여준다.



▶▶ 그림 3. 로그 블록 수에 따른 성능 평가 비교



▶▶ 그림 4. 병합 로그 블록 수에 따른 성능 비교



▶▶ 그림 5. 병합 연기 데이터 블록 비율에 따른 성능 비교

Ⅴ. 결론

본 논문에서는 로그 블록이 포함하는 데이터 블록들의 가치를 평가하여 병합할 로그 블록을 선택하여 병합 연산을 수행하는 효과 적인 로그 버퍼 관리 기법을 제안 한다. 제안 하는 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 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] "Understanding the Flash Translation Layer (FTL) Specification", Intel Corporation, 1998.

[2] M. Rosenblum and J. K. Ousterhout, "The Design and Implementation of a Log-Structured File System", ACM Transactions on Computer Systems, vol. 10, pp. 26-52, 1992

[3] A. Kawaguchi, S. Nishioka, and H. Motoda, "A Flash-Memory Based File System", in Proceedings of the Winter1995 USENIX Technical Conference, 1995, pp. 155-164

[4] Jesung Kim, Jong Min Kim, Noh S.H, Sang Lyul Min, Yookun Cho, "A space-efficient flash translation layer for CompactFlash systems", IEEE Transactions on Consumer Electronics, pp. 366~375, May 2002

[5] Sang-Won Lee, Dong-Joo Park, Sang-Won Lee, Tae-Sun Chung, Dong-Ho Lee, Sang-won Park ,Ha-Joo Song, "A Log Buffer-Based Flash Translation Layer Using Fully-Associative Sector Translation", ACM Transactions on Embedded Computing Systems, vol. 6 Issue 3, July 2007