플래시메모리 DBMS를 위한 블록의 비고정적 로그 영역 관리 기법

(A Non-fixed Log Area Management Technique in Block for Flash Memory DBMS)

조 혜 원 † 한 용 구 †† 이 영 구 †††

(Hye-Won Cho) (Yongkoo Han) (Young-Koo Lee)

요 약 빈번한 데이터 접근이 요구되는 DBMS 분야에서 플래시메모리의 빠른 연산 속도를 이용하여 시스템 성능을 향상시키기 위한 연구가 활발히 이루어지고 있다. 플래시메모리를 DBMS의 저장매체로 사용하는데 가장 큰 문제점 중에 하나는 비효율적인 덮어쓰기 연산으로 인한 성능 저하와 수명 단축이다. 페이지 보다 작은 크기의 쓰기 연산이 빈번히 발생하는 특성을 가진 DBMS 환경에서 비효율적인 덮어 쓰기 문제를 해결하기 위하여 업데이트 내용을 로그 형태로 저장하는 기법들이 연구되었다. 그러나 기존의 연구들은 로그 저장 영역을 고정적으로 관리하여 쓰기 성능 저하의 주요 원인인 합병이 빈번히 발생한다는 문제가 있다. 본 논문에서는 블록의 로그 영역을 비고정적으로 관리하여 로그 저장 공간의 부족으로 인한 합병의 발생을 최소화시키는 기법을 제안한다. 또한 블록 내에서 최소의 비용으로 로그 저장 공간을 사용할 수 있도록 하는 기준을 제시하여 블록의 비고정적인 로그 영역 관리 기법에 적용한다. 실험을 통하여 제안하는 비고정적인 로그 영역 관리 기법이 기존의 고정적으로 관리하는 기법들과의 비교하여 성능 향상을 가져올 수 있는 것을 입증하였다.

키워드 : 플래시메모리, 데이터베이스 시스템, 로그

Abstract Flash memory has been studied as a storage medium in order to improve the performance of the system using its high computing speed in the DBMS field where frequent data access is needed. The most difficulty using the flash memory is the performance degradation and the life span shortening of flash memory coming from inefficient in-place update. Log based approaches have been studied to solve inefficient in-place update problem in the DBMS where write operations occur in smaller size of data than page frequently. However the existing log based approaches suffer from the frequent merging operations, which are the principal cause of performance deterioration. Thus is because their fixed log area management can not guarantee a sufficient space for logs. In this paper, we propose non-fixed log area management technique that can minimize the occurrence of the merging operations by promising an enough space for logs. We also suggest the cost calculation model of the optimal log sector number minimizing the system operation cost in a block. In experiment, we show that our non-fixed log area management technique can have the improved performance compared to existing approaches.

Key words : Flash Memory, DBMS, Log

· 이 논문은 2010년도 정부(교육과학기술부)의 재원으로 한국연구재단의 지원을 받아 수행된 연구임(No.2010-0018941)

† 정 회 원 : 경희대학교 컴퓨터공학과
davinci44@khu.ac.kr
†† 학생회원 : 경희대학교 컴퓨터공학과
ykhan@khu.ac.kr
††† 종신회원 : 경희대학교 컴퓨터공학과 교수
yklee@khu.ac.kr
(Corresponding author임)

논문접수 : 2010년 8월 23일
심사완료 : 2010년 8월 26일

정보과학회논문지: 데이타베이스 제37권 제5호(2010.10)

1. 서 론

플래시메모리(flash memory)는 전원이 꺼져도 한 번 기록된 데이터가 지워지지 않는 비휘발성 메모리로서 빠른 연산 속도, 소형, 가벼운 무게, 강한 내구성, 저 전력 등의 다양한 장점을 가져 휴대용 전자기기나 내장형 시스템의 저장매체로 많이 사용되고 있다. 플래시메모리는 회로 구성에 따라 병렬적 회로 구성의 NOR 형과 직렬적 회로 구성의 NAND 형으로 구분되며 본 논문에서는 집적도가 높아 주로 대용량 데이터의 저장 용도로 쓰이는 NAND형 플래시메모리[1]를 대상으로 한다.

최근에는 플래시메모리의 대용량화와 가격 하락으로 데스크 탑과 노트북의 저장 매체로 사용하기 위한 시도가 이루어지고 있다. 이에 따라 빈번한 데이터 접근이 요구되는 DBMS 분야에서도 디스크보다 빠른 연산 속도를 가진 플래시메모리를 저장매체로 사용하여 성능을 향상시키기 위한 연구들[2,3]이 수행되고 있다. 플래시메모리를 DBMS의 저장매체로 사용하는데 가장 큰 문제점 중에 하나는 비효율적인 덮어쓰기 연산으로 인한 성능 저하와 수명 단축이다. 비효율적인 덮어쓰기 연산이란 업데이트 연산이 요청된 페이지가 속한 블록 전체를 주 메모리로 읽어와 업데이트를 하고 원 블록을 소거 및 초기화 한 후에 업데이트 된 내용을 다시 쓰는 복잡한 과정의 연산이다. 이와 같은 비효율성은 한 번 저장된 페이지는 삭제되어야만 재사용될 수 있으며 읽기/쓰기 연산과 삭제 연산의 단위가 각각 페이지와 블록으로 서로 다른 플래시 메모리의 특성 때문에 발생한다. 플래시메모리의 비효율적인 덮어쓰기 문제는 데이터베이스 서버에 대하여 하드 디스크와 플래시메모리 SSD의 성능을 비교 및 분석한 연구[4]에서도 제기되었다. 이 연구에 따르면 순차적 쓰기 연산과 임의적 읽기 연산이 빈번하게 발생하는 특징의 저장 공간에서는 SSD의 성능이 뛰어나지만, 작은 크기의 임의적 쓰기 연산이 빈번한 저장 공간 타입에서는 플래시메모리의 비효율적인 덮어쓰기로 인하여 하드 디스크보다 낮은 성능을 보였다.

비효율적인 덮어쓰기 문제를 해결하기 위하여 업데이트된 내용을 로그 형태로 저장하는 연구[5,6]들이 수행되고 있다. 로그 형태로 저장하는 기법은 업데이트가 발생한 페이지를 덮어쓰는 대신 새로운 빈 페이지에 업데이트 페이지를 저장하여 덮어쓰기 연산을 회피함으로써 쓰기 연산의 성능을 향상시킨다. 최근에는 페이지보다 작은 크기의 업데이트가 빈번히 발생하는 DBMS환경을 고려하여 데이터 페이지가 저장된 동일 블록 내에 업데이트 된 내용만을 로그로 저장하는 IPL[7]과 PM[8] 연구들이 수행되었다. 이 연구들은 데이터 페이지와 로그를 같은 블록 내에 저장하기 위하여 블록의 저장 공간을 고정적으로 분할하였다. 그런데 이와 같은 업데이트 사항을 로그 형태로 저장하는 기법들은 원본 페이지와 업데이트 내용이 저장된 로그들이 공존하여 논리적 합병과 물리적 합병이라는 또 다른 비효율성을 낳는다. 논리적 합병이란 페이지 읽기 연산을 수행하기 위하여 원본 페이지와 로그들을 주 메모리에 읽어와 원본 페이지에 로그들을 반영함으로써 임시적으로 최신 버전의 페이지를 생성하는 것이다. 그리고 물리적 합병이란 블록 내에 더 이상 로그를 기록할 공간이 없을 경우, 원본 페이지와 로그들을 합병하여 새로운 빈 블록에 최신 버전의 페이지를 생성하는 것이다. 이 두 가지 합병은 각각 읽기 성능과 쓰기 성능 저하의 주요 요인이다.

DBMS를 위하여 설계된 IPL은 물리적 합병을 블록 단위로 수행하기 때문에 합병 연산이 발생하면 블록 내의 모든 페이지가 자신의 로그들과 합병되어 새로운 블록으로 다시 옮겨가야만 한다. PM은 페이지의 업데이트 빈도를 고려하여 IPL을 페이지 단위의 물리적 합병 연산으로 재설계하였다. 그러나 페이지의 업데이트 빈도를 고려한 PM 또한 블록 내 로그 저장 영역을 고정적으로 관리하기 때문에 합병 연산의 성능저하를 모두 해결하지는 못하였다. 빈번한 업데이트가 발생하는 데이터 페이지가 저장된 블록에서는 물리적 합병을 방지하기 위하여 충분한 로그 저장 공간을 보장해야한다. IPL과 PM은 로그의 저장을 위한 공간이 고정적으로 한정되어 있기 때문에 이와 같은 페이지들의 쓰기 빈도 패턴을 반영할 수가 없어 쓰기 성능의 저하가 여전히 존재한다.

본 논문에서는 플래시메모리 기반의 DBMS에서 물리적 합병으로 인한 시스템의 성능 저하를 최소화하기 위하여 블록의 로그 영역을 비고정적으로 관리하는 기법을 제안한다. 제안하는 비 고정적 관리 기법은 블록 내에서 페이지의 쓰기 빈도 패턴에 따라 로그 영역의 크기를 확대 및 축소하여 사용할 수 있기 때문에 로그 영역을 충분히 보장해줄 수 있다. 그리고 페이지들의 읽기 및 쓰기 연산 비율의 통계를 이용하여 블록에서 페이지에 대한 연산들의 비용을 최소화할 수 있는 최적의 로그 영역 크기에 대한 기준을 제시한다.

본 논문의 기여 사항은 다음과 같다.

- 제안하는 블록 내 로그 영역을 비고정적으로 관리하는 기법은 데이터 페이지의 쓰기 빈도 패턴에 따라 요구되는 로그 영역의 크기를 보장하여 물리적 합병 횟수를 줄일 수 있다. 물리적 합병 횟수를 최소화함으로써 플래시 메모리의 쓰기 성능을 향상시킬 수 있다.
- 물리적 합병과 논리적 합병 비용을 최소화 하는 최적 로그 섹터 개수를 계산하는 비용 모델을 제안한다. 데이터 페이지의 로그섹터 영역을 충분히 보장하면 물리적 합병을 줄일 수는 있지만, 읽기 연산 시 많은 수의

로그들을 주 메모리로 읽어와 데이터 페이지와 합병을 수행하기 때문에 논리적 합병 비용이 증가한다. 제안하는 비용 모델은 페이지의 읽기/쓰기 연산 빈도의 비율을 이용하여 최적의 로그 섹터 개수를 계산한다.

블록의 비고정적 로그 영역 관리에서 데이터 페이지의 최적 로그 섹터 개수를 보장하는 알고리즘을 개발하여 플래시 메모리의 읽기 및 쓰기 연산 속도를 향상시켰다.

본 논문의 구성은 다음과 같다. 2절에서는 플래시 메모리의 업데이트 연산을 로그 형태로 저장하는 관련 연구들을 소개하고, 3절에서는 제안하는 블록의 비고정적 로그 영역 관리 기법을 설명한다. 4절에서는 성능 평가를 통해 제안하는 방법의 우수성을 보이고, 마지막으로 5절에서 본 연구의 결론을 맺는다.

2. 관련연구

플래시메모리 DBMS를 위한 저장 기술은 플래시메모리의 비효율적인 덮어쓰기를 회피하기 위하여 업데이트된 페이지를 로그 형태로 저장하는 방향으로 연구되어 왔다. 본 장에서는 로그 형태로 저장하는 대표적인 두 가지 기법과 관련된 연구들을 기술한다. 2.1절은 가장 기본적인 방법으로서 업데이트된 페이지 전체를 로그 형태로 쓰는 기법을 소개하고, 2.2절은 업데이트된 내용만을 로그 형태로 저장하는 기법을 소개한다.

2.1 로그 형태 저장 기법

로그 형태로 저장한다는 것은 모든 쓰기 연산을 어펜드(append) 형태로 수행하는 것을 의미한다. 그림 1에서와 같이 두 번째 유효 페이지에 대한 업데이트 쓰기(write-update) 연산이 발생하면, 이 페이지에 덮어쓰기를 하지 않고 다른 빈 페이지에 변경된 페이지 내용 전체를 새롭게 저장(write-append)한다.

플래시메모리를 위한 저장기술 분야에서 로그 형태로 저장하는 기법의 시초는 디스크를 위한 로그 기반 파일 시스템(log-structured file system, LFS)[9]에서 유래한다. LFS는 작은 크기의 빈번한 쓰기 연산이 발생하는 작업부하 환경에서는 파일을 고정된 위치에 저장하는 기존의 디스크 기반 파일시스템 구조가 시스템 성능을 저하하는 주요한 원인으로 분석했다. 기계적 지연에 의하여 디스크 데이터 접근 속도가 느리며, 임의 데이터 쓰기 연산에 대하여 데이터 저장 위치를 탐색하는 비용이 커지기 때문이다. 따라서 LFS는 모든 쓰기 연산을 단지 순차적으로 어펜드 형태로 수행함으로써 탐색 비용을 감소시키고, 버퍼 캐시를 이용하여 여러 번의 쓰기 연산을 한 번의 쓰기 연산으로 대체시켜 시스템의 쓰기 성능을 향상시켰다.

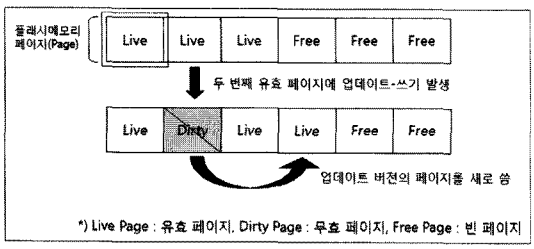

그림 1 로그 형태 저장 기법의 논리적 표현

플래시메모리는 디스크와 달리 데이터의 임의 접근 속도와 순차 접근 속도가 거의 동일하여 데이터 접근 시간으로 인한 성능 상의 문제는 중요하지 않다. 그러나 플래시메모리에서도 업데이트 연산에 대한 비효율적인 덮어쓰기를 로그 형태 저장 기법으로 대체시킴으로써 회피할 수 있다는 장점이 있다. 이와 같이 업데이트 연산을 덮어쓰기 또는 내부 공간 업데이트(in-place update)로 수행하지 않고, 다른 빈 저장 공간에 업데이트 본을 저장하는 것을 외부 공간 업데이트(out-place update)라 한다. 그런데 이와 같은 방법은 플래시메모리가 페이지(Small Block 기준: 512Bytes, Big Block 기준: 2,112Bytes) 단위로 읽기 및 쓰기 연산을 수행하기 때문에 그림 1에서와 같이 업데이트 버전의 페이지를 페이지 또는 페이지의 배수 단위로 저장해야 한다는 비효율성이 있다. 따라서 페이지보다 작은 크기의 쓰기 연산에 대하여 페이지와 업데이트하는 데이터의 차이만큼의 저장 공간을 낭비하는 문제가 발생한다.

2.2 업데이트 사항을 로그형태로 저장하는 기법

2.1절에서 소개한 로그 형태로 저장하는 기법은 페이지보다 작은 크기의 쓰기 연산 시에 저장 공간을 낭비하는 문제점이 있었다. 이와 같이 작은 크기의 업데이트가 빈번하게 발생하는 DBMS 작업부하 환경에서 쓰기 성능을 향상시키기 위한 연구로서 IPL(In-Page Logging Approach)[7] 이 있다. IPL은 업데이트된 사항만을 로그 형태로 저장하는 기법으로 그림 2와 같은 구조를 갖는다.

IPL은 기존 디스크 기반 DBMS에서 버퍼 관리자와 저장 관리자만을 재설계 하였다. 버퍼 관리자가 관리하는 버퍼는 데이터 페이지와 로그 섹터 버퍼로 구성된다. 데이터 페이지는 플래시메모리에 저장하는 데이터의 저장 단위이며, 로그 섹터는 플래시메모리에 저장하는 로그의 저장 단위이다. 저장 관리자는 플래시메모리의 블록을 데이터 페이지 영역과 로그 섹터 영역으로 구성하였으며 각각의 영역의 크기는 초기에 고정적으로 설정된다.

IPL의 기본 동작은 (1) 업데이트 쓰기 연산의 처리 루틴, (2) (1)번 과정 중 논리적 합병 루틴, (3) 플래시메모리에 로그 섹터를 저장하는 루틴, (4) (3)번 동작 중의 물리적 합병(블록 단위) 루틴이 있다.

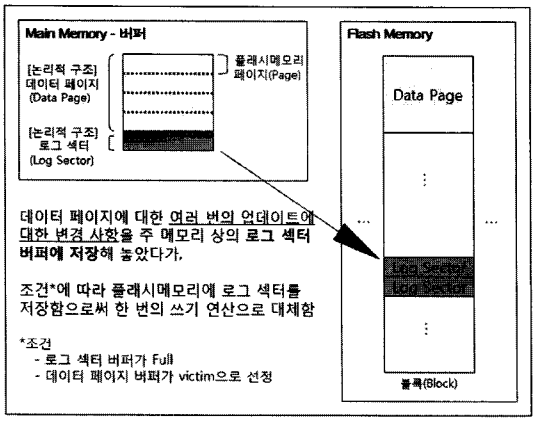

그림 2 업데이트 사항을 로그형태로 저장하는 기법

(1) 업데이트 쓰기 연산의 처리: 버퍼 관리자는 데이터 페이지들에 대한 버퍼 리스트를 관리한다. 버퍼 리스트에 쓰기 연산을 수행하려는 데이터 페이지가 존재하면 이 데이터 페이지 버퍼에 쓰기를 바로 수행하여 반영하고, 로그 섹터 버퍼에 업데이트 사항을 가변 길이의 로그 레코드 형태로 기록한다.

(2) 논리적 합병: (1)번 과정에서 만약 버퍼 리스트에 쓰기 연산 수행하려는 데이터 페이지가 존재하지 않을 경우, 플래시메모리에서 해당 데이터 페이지를 읽어 와야 한다. 이때, 그림 2의 플래시메모리 블록과 같이 데이터 페이지와 함께 관련 로그 섹터들을 모두 주 메모리 버퍼로 읽어와 임시적으로 합병을 수행하여 최신 버전의 데이터 페이지를 생성하는 논리적 합병을 수행한다.

(3) 플래시메모리에 로그 섹터를 저장: (1)번 과정에서 로그 섹터 버퍼에서 오버플로우가 발생할 경우 또는 해당 데이터 페이지 버퍼가 버퍼교체정책의 희생자로 선택된 경우, 로그 버퍼내의 로그 섹터를 플래시메모리에 저장해야 한다. 이때, 로그 섹터는 관련된 데이터 페이지가 저장되어 있는 동일한 블록의 로그섹터 영역에 저장된다.

(4) 물리적 합병: (3)번 과정을 수행 시, 임의의 블록 내 로그 섹터 영역에서 오버플로우가 발생하면 블록 내 모든 데이터 페이지들의 합병을 차례대로 수행하여, 합병된 데이터 페이지들을 새 블록에 저장한다. 이때, 각 데이터 페이지는 먼저 논리적 합병으로 최신 버전의 데이터 페이지를 생성하고, 빈 블록의 데이터 페이지 영역에 이를 순차적으로 저장한다. 모든 데이터 페이지에 대한 합병 연산이 완료되면 이전 블록은 무효 상태가 되어 소거된다.

PM(Page Merging)[8]은 IPL과 같은 업데이트 사항을 로그 형태로 저장하는 기법으로, IPL의 블록 단위 합병을 페이지 단위 합병으로 재설계하였다. 그림 3은 IPL의 블록 단위 합병을 나타내며, 그림 4는 PM의 페이지 단위 합병을 나타낸다.

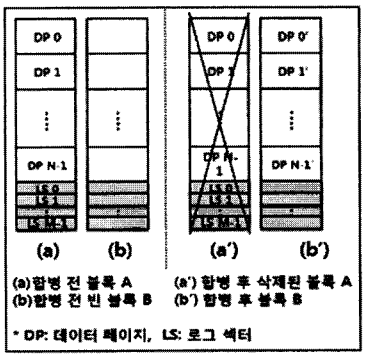

그림 3 블록 단위 합병

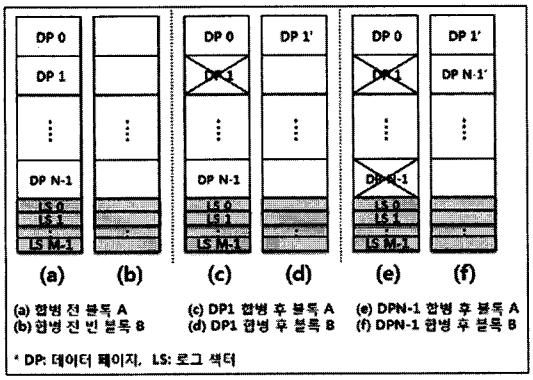

그림 4 페이지 단위 합병

두 합병 방식의 차이점은 블록 내 로그 섹터 영역에 오버플로우가 발생했을 때의 합병 대상이다. IPL의 블록 단위 합병은 그림 3과 같이 합병의 대상이 해당 블록 내 저장되어 있는 모든 유효 데이터 페이지가 되며, PM은 그림 4와 같이 오버플로우를 발생시킨 데이터 페이지만 합병을 수행한다. 그림 4와 같은 페이지 단위 합병은 블록 내에서 오버플로우와 관계없는 데이터 페이지들의 합병 시기를 블록 단위 합병에 비해 지연시킴으로서 플래시메모리의 블록 소거 횟수를 감소시키는 효과가 있다. 또한 블록 단위 합병은 실제 합병이 수행되는 데이터 페이지가 블록 내 저장될 수 있는 총 데이터 페이지의 개수보다 항상 작거나 같으므로 블록 내 사용되지 않고 소거되는 저장 공간 낭비가 발생한다. 페이지 단위 합병은 블록보다 작은 페이지 단위로 저장되는 방식이므로 저장 공간을 상대적으로 효율적으로 사용할 수 있는 장점을 가진다.

3. 블록의 비고정적 로그 영역 관리 기법

본 장에서는 블록의 비고정적 로그 영역 관리 기법을 제시한다. 3.1절에서는 블록의 비고정적 로그 영역 관리 기법을 제시하고, 3.2절에서는 임의 데이터 페이지에 대

해 블록 내에 관련 로그 섹터를 저장함으로써 드는 총 비용이 최소가 되는 최적 로그 섹터 개수를 계산하는 기법을 제시한다. 3.3절에서는 3.2절에서 구한 데이터 페이지의 최적 로그 섹터 개수를 블록의 비고정적 로그 영역 관리 기법에 적용하는 방법에 대해 기술한다.

3.1 블록의 비고정적 로그 영역 관리

플래시 메모리에 저장되는 데이터 페이지들은 각각 서로 다른 업데이트 발생 빈도를 가지고 있다. 따라서 업데이트 연산이 빈발하게 발생하는 데이터 페이지가 저장된 블록은 상대적으로 더 많은 로그 섹터 영역을 할당함으로써 물리적 합병을 최대한 지연시킬 수 있다. 본 논문에서 제안하는 블록의 비고정적 로그 영역 관리는 블록 내의 데이터 페이지 영역과 로그 섹터 영역의 크기를 고정적으로 설정하지 않고, 데이터 페이지의 쓰기 빈도 패턴에 따라 블록 내 저장 공간의 크기를 다르게 사용하는 기법이다. 기존의 고정적 로그 영역 관리 기법에서는 로그 저장 공간이 한정되어 있을 뿐만 아니라 업데이트 빈도가 높은 데이터 페이지의 로그 섹터들이 블록 내 로그 섹터 영역의 대부분을 차지하는 현상이 발생한다. 따라서 로그 섹터 영역의 오버플로우가 쉽게 발생하기 때문에 빈번한 물리적 합병을 야기하는 문제가 있다. 제안하는 기법은 블록의 로그 영역을 비 고정적으로 관리함으로써 데이터 페이지들이 필요한 크기만큼 로그 섹터 영역을 확장하여 사용할 수 있어 물리적 합병의 발생 빈도를 최소화 할 수 있다.

그림 5는 블록의 비고정적 로그 영역 관리 개념을 보여주는 예로서 데이터 페이지와 로그 섹터의 쓰기 빈도 패턴에 따른 저장 공간의 사용 변화를 나타낸다. 그림 5의 (A), (B), (C), (D), (E)는 모두 동일한 블록의 데이터 페이지 또는 로그 섹터의 쓰기로 인한 순차적인 상태 변화를 나타낸다. (A)는 데이터 페이지와 로그 섹터가 저장되지 않은 블록의 초기 상태를 나타내며, (B)~(E)는 블록에 데이터 페이지 또는 로그 섹터의 쓰기 연산이 수행된 블록의 상태를 나타낸다. 데이터 페이지는 블록의 시작부터 하향으로 써지며, 로그 섹터는 블록의 끝부분부터 상향으로 써진다. (E)는 데이터 페이지 영역과 로그 섹터 영역의 경계가 만나 더 이상 데이터를 저장할 수 없는 블록이 가득 찬 상태이다.

블록에 수행되는 기본 연산은 데이터 페이지 쓰기, 로그 섹터 쓰기, 블록이 가득 찼는지 확인, 블록이 초기화되어 있는지 확인 그리고 블록 내 사용할 수 있는 빈 저장 공간 크기의 계산으로 다섯 가지가 있으며, 각 연산은 다음과 같이 수행된다. 첫째, 데이터 페이지 쓰기 연산은 블록 내 사용할 수 있는 빈 저장 공간의 크기가 데이터 페이지의 크기만큼 같거나 큰 경우 수행할 수 있다. 이 조건을 만족하면 쓰기가 발생한 해당 데이터

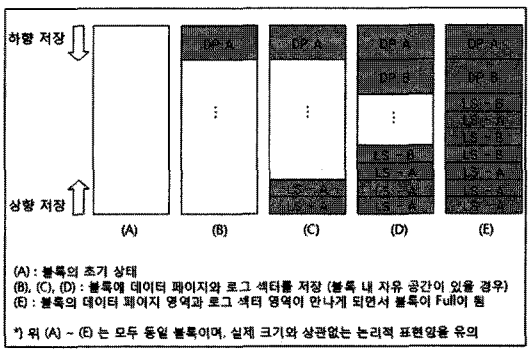

그림 5 블록의 비고정적인 로그 영역 관리 개념

페이지는 블록에 마지막으로 저장된 데이터 페이지의 다음 페이지부터 하향으로 저장된다. 둘째, 로그 섹터 쓰기 연산은 데이터 페이지의 쓰기 연산과 동일한 방식으로 블록 내 로그 섹터를 저장할 수 있는 크기의 빈 저장 공간이 있는지 확인하여, 이 조건을 만족시킨 경우 블록 내 마지막으로 저장 된 로그 섹터 다음 페이지부터 상향으로 저장한다. 이때 마지막으로 저장 된 로그 섹터의 다음 페이지의 기준은 로그 섹터를 블록의 마지막 번째 페이지에서부터 상향으로 저장하는 것이다. 셋째, 블록이 가득 찼는지에 대한 확인은 블록 내 저장된 데이터 페이지들과 로그 섹터들의 경계가 만났는지에 따라 결정된다. 즉, 블록 내에 데이터 페이지들과 로그 섹터들의 경계가 만났을 경우 블록은 가득 찼음을 의미한다. 이와 반대로 블록 내 저장된 데이터 페이지들과 로그 섹터들의 경계가 없는 경우, 즉 블록 내 저장 된 데이터 페이지들과 로그 섹터들 사이에 빈 저장 공간이 존재할 경우는 블록이 가득 차지 않았음을 의미한다. 블록이 가득 찼는지의 여부는 이후 블록 내 임의 데이터 페이지의 로그 섹터 쓰기가 발생 할 경우 해당 데이터 페이지의 물리적 합병(페이지 단위) 수행 여부를 결정하는 기준이 된다. 즉, 임의의 블록에서 임의의 데이터 페이지에 대한 로그 섹터 쓰기가 발생했을 때, 해당 블록의 빈 저장 공간이 없는 경우 해당 데이터 페이지에 대한 물리적 합병이 수행된다. 넷째, 블록이 비어있는 지의 여부는 블록 내 저장되어 있는 데이터 페이지 또는 로그 섹터가 있는지에 따라 결정되며, 아무것도 저장되어 있지 않은 경우 빈 저장 공간은 블록 전체 크기가 되며 이는 블록이 비어있음을 의미한다. 마지막으로 블록 내 사용 가능한 빈 저장 공간의 크기는 블록 내 저장되어 있는 데이터 페이지 영역과 로그 섹터 영역 사이의 빈 저장 공간의 크기로 계산할 수 있다. 이 연산은 블록이 새로운 데이터 페이지를 저장할 수 있을 만큼의 충분한 저장 공간을 가졌는지 판단하는데 사용된다.

블록의 비고정적 로그 영역 관리를 수행하기 위하여

블록 내 데이터 페이지 영역의 하향선과 로그 섹터 영역의 상향선이라는 두 가지 메타정보를 유지하여 관리한다. 이 두 가지 메타정보를 이용하여 블록 내 저장 공간에 대한 모든 상태정보들을 계산할 수 있다. 하향 선은 블록의 데이터 페이지가 저장되는 기준이 되는 선으로, 데이터 페이지의 쓰기가 발생했을 때 블록에서 몇 번째 페이지부터 하향 저장해야 하는지를 나타낸다. 상향선은 블록의 로그 섹터가 저장되는 기준이 되는 선으로, 로그 섹터의 쓰기가 발생했을 때 블록에서 몇 번째 페이지부터 상향 저장해야 하는지를 나타낸다.

표 1은 상향선과 하향선의 초기 값과 블록 내 저장 공간의 상태정보들을 상향선과 하향선으로 계산하는 식을 나타낸다. 우선 하향선과 상향선은 각각 블록의 첫 번째 페이지와 마지막 페이지로 초기화된다. 하향선과 상향선으로 각각 데이터 페이지를 블록 내 몇 번째 페이지부터 저장해야 하는 지와 로그 섹터를 블록 내 몇 번째 페이지부터 저장해야 하는 지를 계산할 수 있다. 또한 블록이 가득 차 더 이상 데이터를 저장할 수 없는지의 여부는 상향 선에서 하향 선의 차에 1을 더한 값이 0 인지로 계산할 수 있으며, 0인 경우 블록이 가득 찬 경우이며 0이 아닌 경우는 블록이 가득 차지 않아 데이터를 저장할 수 있는 공간이 있음을 나타낸다. 블록이 비어있는 지의 여부 또한 동일하게 계산되며, 이 값이 블록 내 총 페이지 개수와 같을 경우가 블록이 비어있는 상태이다. 마지막으로 상향 선과 하향 선의 차에서 1을 더한 값은 블록 내 사용 가능한 빈 페이지의 개수를 의미한다.

블록의 비고정적 로그 영역 관리는 블록의 로그 영역을 고정적으로 관리하는 기존의 방법과 비교하여 다음과 같은 특징을 가진다. 기존의 관리 기법에서는 블록 내 저장 된 데이터 페이지들의 물리적 합병(페이지 단위)의 조건은 로그 섹터 영역이 가득 찼는지의 여부였다. 따라서 로그 섹터 영역의 크기의 초기 설정에 따라 물리적 합병의 발생 빈도가 달라진다. 이 방법은 각 블

표 1 하향선과 상향선을 이용한 블록 저장 공간의 상태 정보 계산

초기화	하향선 (D_Line)	상향선 (U_Line)
	0	*NumOfPageInBlock-1*
블록이 Full 인지 확인	*U_Line-D_Line+1 == 0*	
블록이 Empty인지 확인	*U_Line-D_Line+1 == NumOfPagesInBlock*	
빈 페이지 개수	*U_Line-D_Line+1*	
다음 데이터 페이지가 저장될 페이지 번호	*D_Line*	
다음 로그 섹터가 저장될 페이지 번호	*U_Line*	

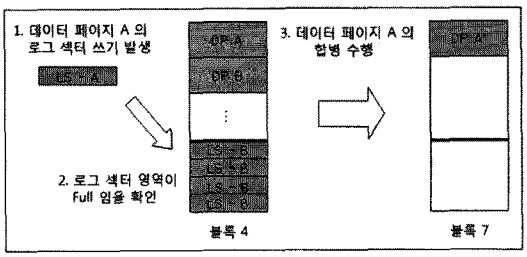

그림 6 블록의 고정적인 로그 영역 관리의 예

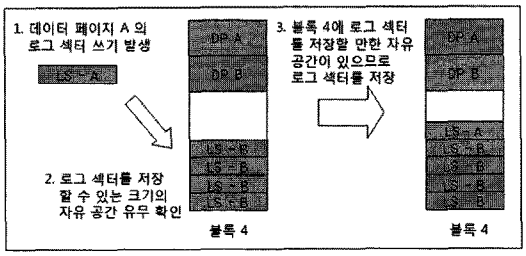

그림 7 블록의 비고정적인 로그 영역 관리의 예

록 내 페이지들의 업데이트 빈도가 모두 다르기 때문에 너무 많은 물리적 합병을 발생시킬 수 있는 문제가 있다. 왜냐하면 빈발한 업데이트 연산이 발생하는 페이지의 로그들이 블록 내 로그 섹터 영역의 대부분을 차지하기 때문이다. 반면에 비고정적 로그 영역 관리 기법은 로그 섹터 영역의 확장이 가능하기 때문에 물리적 합병의 발생 빈도를 줄일 수 있다. 즉, 이 기법은 플래시 메모리의 쓰기 성능에 나쁜 영향을 미치는 물리적 합병 연산을 제한함으로서 쓰기 성능을 향상 시킬 수 있다.

그림 6과 그림 7은 각각 블록의 고정적 로그 영역 관리와 비고정적 로그 영역 관리를 비교한 예이다. 여기서 DP는 데이터 페이지(Data Page), LS는 로그 섹터(Log Sector)를 지칭한다. 그림 6에서 블록 4에 저장된 데이터 페이지 A에 대한 로그 섹터 쓰기가 발생하였으나 이미 블록 4의 로그 섹터 영역이 가득 차 해당 로그 섹터를 저장할 수 없다. 따라서 데이터 페이지 A는 블록 4에서 한 개의 로그 섹터도 저장하지 못하고 물리적 합병으로 인하여 다른 블록에 저장된다. 이 경우는 데이터 페이지 B의 빈발한 업데이트로 인하여 이 로그들이 로그 섹터 영역을 모두 차지하였기 때문에 발생한 상황이다. 반면, 그림 7은 블록의 로그 섹터 영역의 크기를 고정적으로 설정하지 않음으로써 동일한 상황에서 데이터 페이지 A의 로그 섹터를 블록 4에 저장할 수 있도록 한다. 이 경우는 블록 내 빈 공간을 효율적으로 사용하여 물리적 합병의 발생을 방지하였음을 볼 수 있다.

3.2 데이터페이지의 최적 로그섹터 개수 계산

3.1절에서 제시한 블록의 비고정적 로그 영역 관리 기

법을 통해 데이터페이지와 로그 섹터의 쓰기 빈도 패턴에 따라 블록 내 저장 공간을 비고정적으로 사용하여 물리적 합병의 횟수를 제한할 수 있었다. 그러나 이와 같은 블록의 비고정적 로그 영역 관리는 데이터 페이지 영역과 로그 섹터 영역의 정적인 경계가 없기 때문에 새로운 문제들을 발생시킬 수 있다. 첫 째는 상대적으로 너무 많은 데이터 페이지들이 블록의 저장 공간을 차지하여 각 데이터 페이지의 쓰기 빈도 패턴을 보장해주지 못하는 경우이다. 두 번째는 하나의 데이터 페이지에 대한 로그들이 블록을 대부분 차지하여 논리적 합병 비용의 증가를 유도하는 것이다. 자세한 설명은 다음 예제들로 설명한다.

그림 8은 블록에 저장된 데이터 페이지들이 필요로 하는 크기만큼의 로그 섹터 저장 공간을 보장받지 못하는 최악의 경우에 대한 예시이다. 그림 8의 (A), (B), (C), (D), (E), (F)는 모두 동일한 임의 블록의 데이터 페이지 또는 로그 섹터의 쓰기로 인한 순차적인 상태 변화를 나타낸다. 각 (A)~(E)에서는 5개의 데이터 페이지가 차례로 저장됨을 나타내고, (F)에서는 '데이터 페이지 C'의 로그 섹터가 저장됨을 나타낸다. 비고정적 로그 영역 관리 기법은 그림 8에서 볼 수 있듯이 블록 내 대부분의 공간을 데이터 페이지들이 차지하여 로그 섹터를 저장할 수 있는 기회를 단 한번 밖에 가질 수 없는 현상이 발생할 수 있다. 데이터 페이지들에 대하여 업데이트 연산이 한 번만 더 발생하면 데이터 페이지들은 자신의 쓰기 빈도를 보장 받지 못하고 물리적으로 합병되어야 한다. 이와 같은 현상은 불필요한 물리적 합병을 최소화하려는 블록의 비고정적 로그 영역 관리의 목적을 충족시키지 못하므로 비고정적 관리가 가지는 장점을 효과적으로 이용하지 못하는 것이다.

그림 9는 블록에 저장된 데이터 페이지 A에 대하여 계속적으로 업데이트 연산이 발생한 예시이다. 비 고정적인 로그영역 관리에 따라 물리적 합병 발생 없이 자

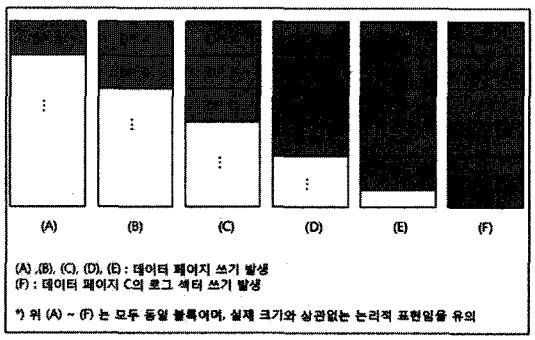

그림 8 비 고정적인 로그 영역 관리에서 페이지의 쓰기 빈도를 보장하지 못하는 경우의 예시

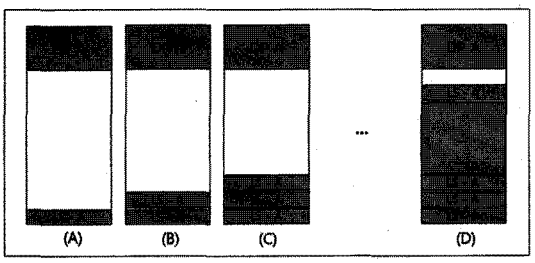

그림 9 비 고정적인 로그 영역 관리에서 높은 논리적 합병 비용을 발생 시키는 경우의 예시

신의 쓰기 빈도 패턴을 보장 받았다. 그러나 이 경우는 데이터 페이지 A에 대한 읽기 연산이 발생하면 데이터 페이지 A와 이 페이지에 관련된 모든 로그 섹터들이 주 메모리에 로드되기 때문에 논리적 합병 연산의 비용이 증가한다. 따라서 그림 9의 (D) 이후에 업데이트 연산은 발생하지 않고 지속적인 읽기 연산이 발생하면 플래시 메모리의 읽기 성능이 저하되는 문제가 있다.

본 논문에서는 이와 같은 문제들을 해결하기 위하여 플래시 메모리의 읽기/쓰기 성능을 모두 고려한 최적의 페이지 당 로그섹터 개수를 계산하여 이를 보장해주기 위한 기법을 제안한다. 본 3.2절의 구성은 다음과 같다. 3.2.1절에서는 데이터 페이지의 최적 로그 섹터 개수에 대한 정의를 제시하고, 3.2.2절에서는 최적 로그 섹터 개수를 구하기 위한 방법을 제시한다.

3.2.1 데이터페이지의 최적 로그섹터 개수

본 절에서는 먼저 데이터 페이지가 필요로 하는 최적 로그 섹터 저장 공간의 크기를 계산하기 위하여 필요한 몇 가지 중요한 개념들에 대하여 정의한다.

정의 1. 데이터 페이지의 주기: 데이터 페이지가 처음 블록에 저장되었거나 물리적으로 합병 되어 블록에 저장된 이후부터 다음 물리적 합병으로 다른 블록에 저장될 때까지를 데이터 페이지의 주기라고 정의한다.

정의 2. 데이터 페이지의 쓰기 빈도: 데이터 페이지가 한 주기에서 업데이트된 횟수를 데이터 페이지의 쓰기 빈도라고 정의한다. 블록 내에는 데이터 페이지의 쓰기 빈도와 동일한 개수의 로그 섹터가 존재한다. 초기의 데이터 페이지 쓰기 빈도는 '0'이며, 물리적으로 합병될 때마다 마지막 주기에서의 쓰기 빈도 값으로 갱신 된다.

정의 3. 데이터 페이지의 읽기 빈도: 한 주기에서 데이터 페이지에 대한 읽기 연산이 발생한 횟수를 데이터 페이지 읽기 빈도라고 정의한다. 만약 읽기 연산이 발생할 때마다 블록이 캐싱되지 않았다면 데이터 페이지의 읽기 연산이 발생한 횟수만큼 논리적 합병 연산이 발생한다. 이 빈도 값은 쓰기 빈도 값과 같은 방식으로 갱신 된다.

정의 4. 데이터 페이지의 최적 로그 섹터 개수: 한 주기에서 데이터 페이지의 총 쓰기 연산 비용과 총 읽기 연산 비용의 합을 최소로 만드는 로그 섹터의 개수이다. 블록 내 데이터 페이지들에 대하여 많은 수의 로그 섹터를 보장하면 물리적 합병의 빈도를 줄일 수 있지만 논리적 합병의 비용이 증가한다. 그리고 적은 수의 로그섹터를 보장하면 반대의 경우가 발생한다. 따라서 물리적 합병이 발생하여 새로운 블록에 데이터 페이지가 저장 될 때 최적 로그 섹터 개수를 계산하여 이를 보장해주어야 한다. 그림 10은 데이터 페이지가 합병된 후 최적 로그섹터 개수를 보장해주는 예시이다. 만약 최적 로그섹터가 3이라면 합병된 데이터 페이지 A가 새로운 블록에 저장될 때 3개의 로그섹터를 데이터 페이지 A를 위하여 보장해주어야 한다.

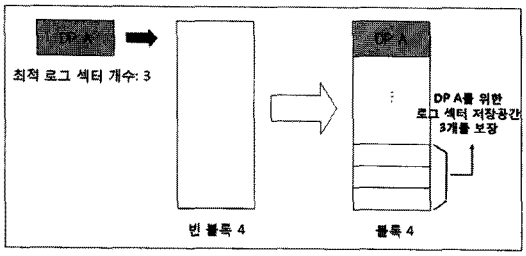

그림 10 데이터페이지의 최적 로그 섹터 개수 보장

3.2.2 데이터페이지의 최적 로그 섹터 개수 계산

데이터 페이지의 최적 로그 섹터 개수의 계산은 두 단계의 과정을 갖는다. 우선 데이터 페이지의 한 주기에서의 읽기 및 쓰기 비용 모델을 계산한다. 그리고 비용 계산 모델을 이용하여 최적 로그 섹터 개수를 결정한다.

데이터 페이지의 총 비용은 읽기 연산과 쓰기 연산의 빈도 패턴에 따라 달라진다. 따라서 합병되기 이전의 블록들에서의 읽기 빈도와 쓰기 빈도 비율의 통계치를 통해 총 합병 비용을 계산할 수 있다. 본 논문에서는 읽기/쓰기 빈도의 비율을 이용하기 위하여 다음과 같은 가정을 한다.

가정 1. 주기 내에서 임의 데이터 페이지의 읽기 연산과 쓰기 연산의 발생이 균등 분포(Uniform Distribution)한다. 그림 11은 임의 데이터 페이지의 한 주기에서의 읽기 연산과 쓰기 연산의 발생 분포가 2 : 1로 균등 분포하는 예시이다.

가정 2. 물리적 합병이 발생되기 이전의 데이터 페이지의 읽기/쓰기 연산 빈도 패턴은 다음 블록의 주기에서도 유사하게 발생한다. 만일 빈도 패턴이 크게 달라지면 통계치를 새로 계산하여 사용한다.

가정 1과 가정 2를 기반으로 읽기/쓰기 연산 발생 빈도 RW_{ratio}를 계산한다. RW_{ratio}는 현재 합병이 수행

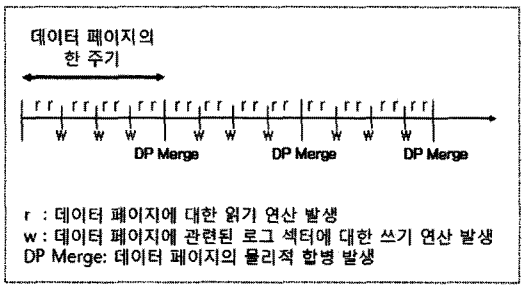

그림 11 읽기 연산과 쓰기 연산 발생의 균등 분포 예시

되기 이전 주기들에서 데이터 페이지의 읽기 연산 발생 횟수의 평균과 쓰기 연산 발생 횟수의 평균값을 이용하여 식 (1)과 같이 계산한다. 여기서 r과 w는 각각 과거 주기들에서 읽기 빈도와 쓰기 빈도의 평균값이다.

$$RW_{ratio} = \frac{r}{w} \tag{1}$$

다음으로 한 주기에서 하나의 데이터 페이지에 대한 총 비용을 식 (1) RW_{ratio}를 이용하여 계산할 수 있다. 우선 총 비용 T는 총 읽기 비용 T_R_Cost와 T_W_Cost의 합으로 식 (2)와 같이 계산된다.

$$T_{total} = T\_R\_Cost + T\_W\_Cost \tag{2}$$

T_R_Cost와 T_W_Cost는 각각 한 주기에서의 임의의 데이터 페이지에 대해 발생하는 모든 읽기 연산들과 쓰기 연산들의 비용으로 다음과 같이 보조정리로부터 계산한다.

보조정리 1. T_R_Cost는 데이터 페이지의 한 주기에서의 총 읽기 비용으로 한 주기에서 읽은 총 로그 섹터 개수와 데이터 페이지 읽기 횟수를 합한 값이다. 한 주기에 저장한 총 로그 섹터 개수가 N, 하나의 로그섹터를 읽는 비용이 LS_R_Cost, 하나의 데이터 페이지를 읽는 비용이 DP_R_Cost이면 RW_{ratio}를 이용하여 T_R_Cost를 식 (3)과 같이 계산할 수 있다.

$$T\_R\_Cost = (RW_{ratio} \times \sum_{1}^{N+1} i \times LS\_R\_Cost) + (RW_{ratio} \times N \times DP\_R\_Cost) \tag{3}$$

증명. 한 주기 내에 읽은 총 로그 섹터 개수는 RW_{ratio}를 이용하여 계산할 수 있다. 한 개의 로그 섹터가 저장될 때마다 RW_{ratio}번의 로그 섹터 읽기 연산이 발생하므로 N개의 로그섹터에 대한 읽기 연산은 $RW_{ratio} + (RW_{ratio} \times 2) + \ldots + (RW_{ratio} \times N)$ 번 발생한다. 그리고 물리적 합병이 발생하기 위해서는 한 개의 로그 섹터가 더 저장되어야 하므로 한 주기 내에 읽은 총 로그 섹터의 수는 $RW_{ratio} \times \sum_{1}^{N+1} i$ 과 같다. 따라서 총 로그 섹터 읽기 비용은 하나의 로그 섹터를 읽는 비용을 곱

한 값인 $RW_{ratio} \times \sum_{1}^{N+1} i \times LS\_R\_Cost$ 가 된다. 본 논문에서는 LS_R_Cost 의 값을 플래시 메모리에서 페이지를 읽는 속도인 $25\mu s$로 계산한다. 그리고 한 주기 내에서 읽은 총 데이터 페이지 읽기 연산 횟수는 한 개의 로그 섹터를 쓸 때마다 RW_{ratio} 만큼의 데이터 페이지 읽기 연산이 발생하므로 RW_{ratio} 에 한 주기에 저장한 총 로그 섹터 개수를 곱하여 계산한다. 따라서 총 데이터 페이지 읽기 비용은 하나의 데이터 페이지를 읽는 비용을 곱한 값인 $RW_{ratio} \times N \times DP\_R\_Cost$ 가 된다.

보조정리 2. T_W_Cost는 한 주기에서 데이터 페이지의 총 쓰기 비용으로 로그 섹터를 저장한 총 비용과 데이터 페이지의 물리적 합병 비용, 그리고 블록 소거 비용을 합한 값으로 식 (4)와 같이 계산할 수 있다.

$$T\_W\_Cost = (N \times LS\_W\_Cost) + Merge\_Cost + \beta \times BlockErase\_Cost \quad (4)$$

여기서 LS_W_Cost 는 하나의 로그 섹터를 쓰는 비용이고, $Merge\_Cost$ 는 데이터 페이지의 물리적 합병 비용이다. 그리고 $\beta \times BlockErase_Cost$는 블록 소거 비용으로 β는 $\frac{1}{\text{블록내총데이터페이지개수}}$로 계산한다.

증명. 한 주기 내에 데이터 페이지의 총 쓰기 비용은 계산은 간단하기 때문에 증명을 생략한다. 다만 물리적 합병 비용 $Merge\_Cost$ 는 식 (5)와 같이 계산할 수 있다.

$$Merge\_Cost = DP\_R\_Cost + (N \times LS\_R\_Cost) + (N \times MCInMM) + DP\_W\_Cost \quad (5)$$

여기서 $MCInMM$ 은 주 메모리상에서 한 개의 로그 섹터를 데이터 페이지와 합병시키는 비용이다. 이 식에서 $DP_R_Cost + (N \times LS_R_Cost) + (N \times MCInMM)$는 데이터 페이지가 이미 버퍼에 있다는 가정 하에 제외시킬 수 있다. 따라서 데이터페이지의 물리적 합병비용을 다시 계산하면 식 (6)과 같다.

$$Merge\_Cost = DP\_W\_Cost \quad (6)$$

총 비용 계산식 (2)에 보조정리 1과 2를 대입하여 정리하면 식 (7)과 같은 최종적인 계산식을 구할 수 있다.

$$T_{total} = (RW_{ratio} \times ((N+1)(N+2)/2) \times LS\_R\_Cost + RW_{ratio} \times N \times DP\_R\_Cost) + (N \times LS\_W\_Cost + DP\_W\_Cost + \beta \times BlockErase\_Cost) \quad (7)$$

식 (7)은 데이터 페이지의 한 주기에서의 총 비용을 의미하는 읽기 및 쓰기 비용 모델이다. 다음으로 읽기 및 쓰기 비용 모델 식 (7)을 이용하여 데이터 페이지의 최적 로그 섹터 개수 계산을 정리 1에 정의한다.

정리 1. 데이터 페이지의 최적 로그 섹터 개수 $N_{optimal}$ 은 로그 섹터 한 개에 대한 연산 비용을 최소로 만드는 개수이다.

$$N_{optimal} = \underset{N}{\arg\min} \frac{T_{total}}{N} \quad (8)$$

증명. 데이터 페이지의 최적 로그 섹터 개수를 계산하기 위하여 총 비용을 다음 주기에서 데이터 페이지에 보장해주어야 할 로그섹터 개수 N과 로그 섹터 한 개당 연산 비용 C_{LS} 의 곱으로 식 (9)와 같이 다시 정의한다.

$$C_{total} = N \times C_{LS} \quad (9)$$

한 주기에서의 총 비용이 최소가 되기 위해서는 로그 섹터 한 개당 연산 비용이 최소가 되어야 하므로 식 (9)를 식 (10)과 같이 다시 쓴다. 따라서 각 데이터 페이지에 보장해주어야 할 최적 로그 섹터의 개수 N은 식 (10)의 값을 최소로 만드는 값이다. 여기서 총 비용 C_{total} 은 식 (7)의 T_{total}과 같다.

$$C_{LS} = \frac{C_{total}}{N} \quad (10)$$

데이터 페이지의 최적 로그 섹터를 계산하는 전체 알고리즘은 그림 12와 같다.

```
Input:  RWratio (=r/w), DP_R_Cost, DP_W_Cost, LS_R_Cost, LS_W_Cost
        β (=1/블록내 총 데이터페이지 개수),
        m (=블록 내에서 저장 가능한 총 로그 섹터 개수)
Output: Noptimal
-------------------------------------------------------------------
데이터 페이지의 RWratio를 이용하여 총 비용식 수립;
  Ttotal = (RWratio × ((N+1)(N+2)/2) × LS_R_Cost + RWratio × N × DP_R_Cost)
           + (N × LS_W_Cost + DP_W_Cost + β × BlockErase_Cost)

for N ← 0 to m do
    Ttotal 계산;
      Ttotal = (RWratio × ((N+1)(N+2)/2) × LS_R_Cost + RWratio × N × DP_R_Cost)
               + (N × LS_W_Cost + DP_W_Cost + β × BlockErase_Cost)
    로그 섹터 1개에 대한 연산 비용 계산;
      CLS[m] = Ttotal / N
End

Noptimal = CLS가 최소 값일 때의 N;
```

그림 12 데이터 페이지의 최적 로그 섹터 개수 계산 알고리즘

3.3 최적 로그섹터 개수를 보장하는 블록의 비고정적 로그 영역 관리 기법

본 절에서는 3.1절과 3.2절로부터 계산한 최적 로그섹터 개수를 블록의 비고정적 로그 영역 관리 기법에 적용시켜 이를 보장하는 방법을 설명한다.

블록의 비고정적 로그 영역 관리 기법에서 최적 로그 섹터 개수를 보장하는 방법은 블록의 상태변수 $GuaranteeLS$ 를 추가함으로써 간단히 구현 될 수 있다. $GuaranteeLS$ 는 블록에 저장된 모든 데이터 페이지들의 각 최적 로그 섹터 개수들의 합으로 해당 블록에 저장되어 있지 않은 다른 데이터 페이지들에 의해 $GuaranteeLS$ 에서 블록의 마지막 페이지까지의 로그섹터 영역이 사용되어 지는 것을 방지하는 역할을 한다.

```
1) 로그 섹터 저장 시, 다음을 검사 함
    if numOptLS < numCurLS
        then GuaranteeLS++
2) 데이터 페이지가 합병될 때, 다음을 검사함
    if numOptLS > numCurLS
        then GuaranteeLS -= (numOptLS – numCurLS)
```

그림 13 블록의 *GuaranteeLS* 정보 갱신 루틴

그림 13은 블록의 *GuaranteeLS* 정보를 갱신하는 루틴이다. 여기서 *numOptLS*는 최적 로그 섹터 개수이다.

4. 성능평가

본 절에서는 제안하는 블록의 비고정적 로그 영역 관리 기법의 성능을 검사하는 실험에 대하여 논한다. 실험의 목적은 물리적 합병과 논리적 합병으로 인한 시스템 성능의 향상을 보이는 것으로, 성능 평가의 척도로 블록 소거 횟수, 데이터 페이지의 물리적 합병 횟수, 로그 섹터 읽기 횟수를 사용한다. 블록 소거 횟수와 데이터 페이지의 물리적 합병 횟수는 쓰기 성능의 척도이며 로그 섹터 읽기 횟수는 읽기 성능의 척도이다. 그리고 본 논문의 목적인 쓰기 비용과 읽기 비용을 합한 총 비용이 최소가 되는 것을 보이기 위하여 세 가지 성능 평가 척도 비용을 모두 합한 비용으로 성능을 비교한다.

4.1 실험 환경 및 실험 데이터

실험을 위하여 블록에서 로그 영역을 관리하는 방식과 물리적 합병을 수행하는 단위에 따라 세 가지 시뮬레이터를 구현하였다. 첫 번째로, 블록의 로그 영역을 고정적으로 관리하고 블록 단위 합병을 수행하는 IPL 연구의 시뮬레이터인 Fixed_BM(fixed log area management_block merge), 두 번째로, 블록의 로그 영역을 고정적으로 관리하고 페이지 단위 합병을 수행하는 PM 연구의 시뮬레이터인 Fixed_PM(fixed log area management_page merge), 그리고 본 논문에서 제안하는 블록의 로그 영역을 비고정적으로 관리하고 페이지 단위 합병을 수행하는 시뮬레이터인 NonFixed_PM(non fixed log area management_page merge)을 구현하였다. Fixed_BM과 Fixed_PM은 각 논문들에서 기술한 알고리즘을 토대로 구현하였다.

이전 연구들의 실험 데이터는 TPC-C 벤치마크에 대한 오라클의 트랜잭션 수행[10]으로 생성된 리두(Redo) 로그 레코드를 추출할 수 있었으나, 리두 로그는 쓰기 연산에 관한 로그 레코드만 기록되므로 본 연구에의 목적에 적합하지 않다. 따라서 읽기 연산과 쓰기 연산에 대한 입력 데이터를 직접 생성하여 실험하였다.

플래시메모리의 크기는 64MB로 빅 블록 낸드(Big Block NAND) 모델을 기준으로 설정하였다. 따라서 페이지 크기는 2,112B, 블록의 크기는 2,112*64MB 이며, 총 블록의 개수는 64MB / 2,112*64MB 인 248개이다. 또한 총 데이터 페이지의 개수는 1,000개로 실험하였으며 데이터 페이지의 주소 값은 균등분포의 난수로 생성하였다.

4.2 실험 결과

그림 14는 트린잭션 횟수의 증가에 따른 데이터 페이지 합병 횟수를 비교한 것이다. 제안하는 블록의 비고정적 로그 영역 관리 NonFixed_PM은 Fixed_BM, Fixed_PM과 비교하여 데이터 페이지의 물리적 합병 횟수가 감소하였다. 로그 영역을 비고정적으로 관리함으로써 고정적으로 관리하는 다른 기법들보다 데이터 페이지의 쓰기 빈도 패턴만큼의 로그 영역을 보장해주었기 때문이다. 제안하는 비고정적인 기법은 블록 내의 저장 영역을 데이터 페이지의 쓰기 빈도 패턴과 읽기 빈도 패턴에 따라 효율적으로 관리함으로써 쓰기 성능 저하의 커다란 원인인 물리적 합병 횟수를 감소시킬 수 있었다.

그림 15는 트랜잭션 횟수의 증가에 따른 블록 소거 횟수를 비교한 것이다. 제안하는 기법에서는 다른 기법들과 비교하여 블록 소거 횟수가 감소하였다. 블록 소거 연산은 새로운 블록에 데이터 페이지를 쓰기 전에 발생할 수 있다. 따라서 물리적 합병의 횟수가 감소함에 따라 블록 소거 횟수도 감소했음을 알 수 있다. 그림 14와 그림 15에서 트랜잭션 개수의 증가에 따른 각 기법들의 물리적 합병 횟수와 블록 소거 횟수가 유사한 비율로 증가하는 것을 알 수 있다.

데이터 페이지 합병 횟수			
트랜잭션	Fixed_BM	Fixed_PM	NonFixed_PM
100,000	30,769	15,763	5,673
500,000	155,855	80,265	26,626
1,000,000	312,336	160,914	52,978
2,000,000	624,992	322,193	105,602

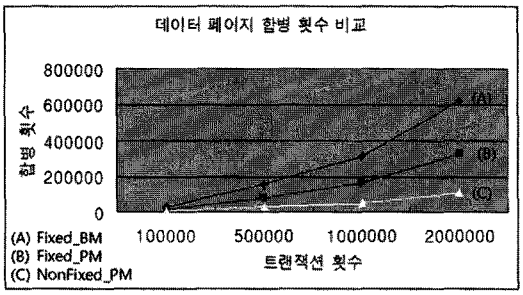

그림 14 데이터 페이지 합병 횟수 비교

블록 소거 횟수			
트랜잭션	Fixed_BM	Fixed_PM	NonFixed_PM
100,000	1,929	972	423
500,000	9,770	5,279	2,171
1,000,000	19,581	10,653	4,369
2,000,000	39,182	21,407	8,755

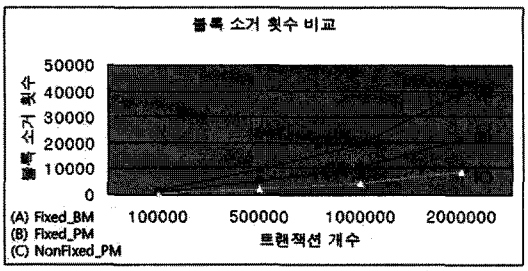

그림 15 블록 소거 횟수 비교

읽은 로그 섹터 개수			
트랜잭션	Fixed_BM	Fixed_PM	NonFixed_PM
100,000	35,334	51,576	184,078
500,000	177,785	257,317	999,057
1,000,000	358,205	516,967	2,021,198
2,000,000	715,187	1,029,176	4,044,130

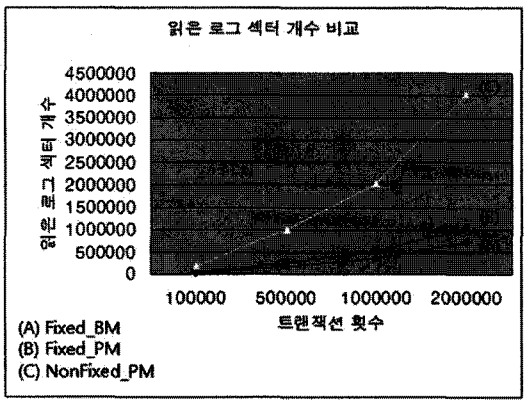

그림 16 읽은 로그 섹터 개수 비교

그림 16은 트랜잭션 횟수의 증가에 따른 읽은 로그 섹터의 개수를 비교한 것이다. 제안하는 비고정적인 기법은 다른 기법들과 비교하여 로그 섹터를 읽은 개수는 증가하였음을 알 수 있다. 다른 기법들에 비하여 블록 내에 로그 섹터의 개수를 더 많이 보장해주었기 때문이다. 이와 같은 결과는 물리적 합병 횟수는 감소시키지만 읽기 성능에는 상대적으로 나쁜 성능을 보일 수 있다.

[총 데이터 페이지 합병 비용 + 총 블록 소거 비용 + 총 로그 섹터 읽기 비용]			
트랜잭션	Fixed_BM	Fixed_PM	NonFixed_PM
100,000	28,392,050	15,357,800	9,774,850
500,000	143,783,625	78,563,425	49,533,725
1,000,000	288,195,425	157,634,875	99,465,850
2,000,000	576,646,275	315,594,300	198,717,350

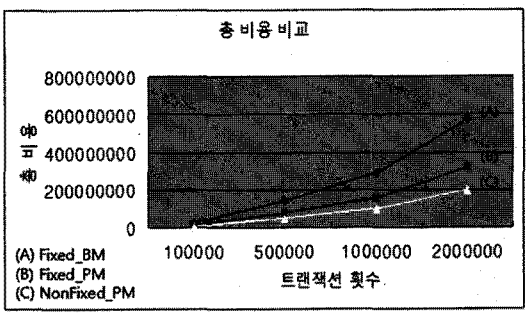

그림 17 총 비용 비교

그러나 플래시메모리의 읽기 속도는 쓰기 속도와 비교하여 2~3배 빠르며, 또한 그림 17에서와 같이 읽기 비용과 쓰기 비용을 더한 총 비용에서는 성능이 향상됨을 보여 읽기 성능의 저하가 전체 시스템 성능에 크게 영향을 끼치지 않았음을 보였다.

그림 17은 데이터 페이지 합병 비용, 블록 소거 비용, 그리고 로그 섹터 읽은 비용을 모두 더한 총 비용을 비교한 것이다. 그림 17에서 볼 수 있듯이 제안하는 비고정적인 기법의 성능이 가장 우수한 것을 알 수 있다. 제안하는 NonFixed_PM 기법은 Fixed_BM과 Fixed_PM에 비교하여 블록 당 관리해야 하는 메타 데이터가 존재하므로 저장 공간을 상대적으로 더 많이 필요로 하는 추가 비용이 발생하나 그 크기가 무시할 만한 수준이라 판단되므로 총 비용에서 제외시켜 계산하였다.

5. 결 론

본 논문의 주요 공헌은 업데이트 사항을 로그 형태로 저장하는 기법에서 데이터 페이지의 읽기 및 쓰기 빈도 패턴을 반영할 수 있는 비 고정적인 로그 영역 관리 기법을 고안한 것이다. 그리고 블록의 비고정적 로그 영역 관리에서 데이터 페이지 영역과 로그 섹터 영역의 정적인 경계가 없기 때문에 발생하는 문제점들을 최적 로그 영역 개수를 보장함으로서 해결하였다. 최적 로그 영역 개수는 데이터 페이지의 읽기 및 쓰기 빈도 패턴을 고려하여 한 주기에서 블록 내 총 비용을 최소로 만드는 개수로 결정하였다. 이로써 기존의 고정적 로그 영역 관리 기법들보다 물리적 합병 횟수를 감소시켜 쓰기 성능

을 향상시켰고 성능평가를 통하여 제안하는 기법이 기존 연구들과 비교하여 쓰기 성능이 향상되었음을 보였다. 제안하는 비 고정적 로그영역 관리 기법은 플래시 메모리 기반 DBMS 환경에서 기존의 기법들보다 더 우수한 시스템 성능을 발휘할 수 있다.

참 고 문 헌

[1] Samsung Electronics, Flash Memory K9XXG08UXA Datasheet, http://www.samsung.com/global/business/semiconductor

[2] G. Kim, S. Baek, H. Lee, H. Lee, M. Joe, "LGe-DBMS: A Small DBMS for Embedded System with Flash Memory," *Proc. of the 32nd Int. Conf. on Very Large Data Bases*, pp.1255-1258, 2006.

[3] G. Na, S. Kim, J. Kim, S. Lee, "Applying In-Page Logging to SQLite DBMS," *Journal of KIISE : Database*, vol.35, no.5, pp.400-410, Oct. 2008.

[4] S. Lee, B. Moon, C. Park, J. Kim, S. Kim, "A Case for Flash Memory SSD in Enterprise Database Applications," *Proc. of the ACM SIGMOD*, pp.1075-1086, 2008.

[5] D. Woodhouse, "JFFS: The Journalling Flash File System," *Proceedings of Ottawa Linux Symposium*, 2001.

[6] C. Manning and Wookey, "YAFFS Specification," Aleph One Limited, 2001.

[7] S. Lee, B. Moon, "Design of Flash-Based DBMS: An In-Page Logging Approach," *Proc. of the ACM SIGMOD*, pp.55-66, June 2007.

[8] H. Cho, Y. Lee, "Page Merging Technique for Flash Memory DBMS," *Proc. of the 35th KIISE Conference*, vol.36, no.1(C), pp.197-202, 2009. (in Korean)

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

[10] J. Dyke, S. Shaw, "Pro Oracle Database 10g RAC on Linux: Installation, Administration, And Performance," Apress, 2006.

조 혜 원

2008년 경희대학교 컴퓨터공학과 학사
2010년 경희대학교 컴퓨터공학과 석사
관심분야는 플래시메모리 파일시스템, 유비쿼터스 데이터 관리

한 용 구

2005년 경희대학교 컴퓨터공학과 학사
2007년 경희대학교 컴퓨터공학과 석사
2007년~현재 경희대학교 대학원 컴퓨터공학과 박사과정. 관심분야는 행위인지, 클라우드 컴퓨팅, 데이터 마이닝

이 영 구

1992년 한국과학기술원 과학기술대 전산학과 학사. 1994년 한국과학기술원 전산학과 석사. 2002년 한국과학기술원 전산학과 박사. 2002년 9월~2004년 2월 미국 UIUC 전산학과 Post Doctoral Research Fellow, 2004년 3월~현재 경희대학교 컴퓨터공학과 부교수. 관심분야는 대용량 데이터 관리, 클라우드 컴퓨팅, 유비쿼터스 데이터관리, 데이터 마이닝