JKSCI

한국컴퓨터정보학회논문지
Journal of The Korea Society of Computer and Information
Vol. 25 No. 10, pp. 1-14, October 2020
https://doi.org/10.9708/jksci.2020.25.10.001

Index management technique using Small block in storage device based on NAND flash memory

Seung-Woo Lee*, Se-Jin Oh*

*Professor, Dept. of Aeronautical Software Engineering, Kyungwoon University, Gumi, Korea
*Professor, Dept. of Aeronautical Software Engineering, Kyungwoon University, Gumi, Korea

[Abstract]

In this paper, we propose to solve the problem of increasing system memory usage due to an increase in the number of mapping information management when using a NAND flash memory-based storage device in an existing sector-based file system. The proposed technique is to store only mapping information in page units based on index blocks and manage them in block units. To this end, the proposed technique uses a sequential offset for storing and managing a plurality of mapping information in one page in a small block, and a reverse offset for a spare page corresponding to a change in mapping information in the block. Through this, the proposed technique has the advantage that the number of block-unit deletions is less than that of the existing technique, and the system memory usage required for mapping information management is low. Reduced by about 32%.

▸ **Key words**: Nand Flash Memory, FTL, Garbage Collection, Mapping table, Embedded system

[요 약]

이 논문에서는 낸드 플래시 메모리 기반 저장장치를 기존 섹터 기반 파일시스템에서 사용할 때 사상정보 관리개수의 증가로 인해 시스템 메모리 사용량이 증가하는 문제를 최대한 해결하기 위해 데이터 입출력 시 시간적집약성과 공간적집약성을 함께 고려한 사상기법을 제안한다. 제안기법은 색인 블록을 기반으로 페이지 단위로 사상정보만을 저장하여 이를 블록단위로 관리한다. 제안기법은 이를 위해 스몰 블록 내 하나에 페이지에 다수의 사상정보를 저장하여 관리하기 위한 순차 오프셋을 이용하며 블록 내에서 사상정보에 변경에 대응하는 여유페이지를 위한 역순 오프셋을 이용한다. 제안기법은 이를 통해 기존기법과 비교하여 블록단위 삭제 횟수가 적게 발생하며 사상정보 관리를 위해 필요한 시스템메모리 사용량이 낮은 장점이 있다 제안기법에 색인 블록 사상기법은 기존 로그블록 기법과 비교하여 블록병합 횟수를 약 32% 줄였다.

▸ **주제어**: 낸드 플래시 메모리, FTL, 가비지 컬렉션, 사상테이블, 내장형 시스템

• First Author: Seung-Woo Lee, Corresponding Author: Se-Jin Oh
*Seung-Woo Lee (swlee@ikw.ac.kr), Dept. of Aeronautical Software Engineering, Kyungwoon University
*Se-Jin Oh (sjoh@ikw.ac.kr), Dept. of Aeronautical Software Engineering, Kyungwoon University
• Received: 2020. 09. 21, Revised: 2020. 10. 12, Accepted: 2020. 10. 14.

http://www.ksci.re.kr pISSN:1598-849X | eISSN:2383-9945

I. Introduction

플래시메모리 기반 저장장치는 전기, 전자적으로 동작함으로 하드디스크와 비교해 더 빠른 데이터 입출력 속도와 더 낮은 전력소비 그리고 더 가벼운 무게로 인한 휴대성 등에 여러 장점을 가진다. 대표적인 플래시메모리 기반 저장장치로는 SSD(solid-state drive), USB메모리, 스마트폰, IoT(Internet of Things)장비, 임베디드 시스템 등이 있으며 현제 다양한 컴퓨팅 관련분야에서 하드디스크를 대체해 활발히 사용되고 있다[1, 2, 3].

플래시메모리는 플로팅 게이트 내에 전자의 충전과 방전 현상을 이용하여 bit를 표현하며 데이터를 기록하는 물리적 최소 저장단위를 셀이라고 한다. 플래시메모리는 다수의 셀로 페이지 단위를 구성하고 다수의 페이지로 블록 단위를 구성한다. 또한 플래시메모리는 데이터 쓰기와 읽기 명령에 경우 페이지 단위로 동작하며 데이터 삭제 명령은 블록 단위로 동작하는 특징이 있다.

플래시메모리는 셀의 배열 방식에 따라 NAND와 NOR로 구분되며 NAND 방식에 경우 NOR 방식에 비해 셀의 배열 방식에 이점이 있어 상대적으로 더 높은 집적도를 가진다. 또한 최근 초미세 공정에 지속적인 발전으로 셀당 멀티비트를 표현할 수 있어 가격대비 저장용량의 이점 또한 가진다. 이러한 다양한 이유로 플래시메모리 기반 저장장치는 NAND 방식이 대부분이며 대부분 대용량 데이터 저장목적으로 사용되는 것이 일반적이다.

하지만 여러 장점을 가진 낸드 플래시 메모리 기반 저장장치를 효율적으로 사용하기 위해서는 반드시 고려할 점이 있다. 특히 낸드 플래시 메모리는 물리적 구조로 인해 데이터 갱신 시 제자리 덮어쓰기(in-place updates)가 불가능하며 데이터 갱신 시에는 반드시 쓰기 전 지우기(erase-before-write) 동작이 선행되어야 한다. 하지만 앞서 언급한 것처럼 낸드 플래시 메모리는 데이터 삭제 시 블록단위로 동작하기 때문에 데이터 업데이트 요청 시 업데이트 된 데이터가 기록된 페이지를 포함한 블록 내의 모든 페이지를 새로운 블록에 복사해야하는 오버헤드가 발생한다. 이러한 데이터 갱신 시 발생하는 추가동작으로 인해 저장장치 내에 비어있는 블록의 수가 줄어들수록 쓰기 전 지우기 동작으로 인한 오버헤드는 점차 증가하게 된다. 즉 빈번한 데이터 갱신으로 발생하는 쓰기 전 지우기 동작은 낸드 플래시 메모리 기반 저장장치 사용에 있어 반드시 고려해야할 중요한 문제로 알려져 있다.

또한 기존 섹터 기반 파일시스템에 경우 데이터저장 및 삭제와 읽기 시 논리블록주소(Logical block addressing)를 이용하여 실제 데이터가 기록된 물리위치를 사상한다. 논리블록주소는 낸드 플래시 메모리 등장 이전 컴퓨팅시스템에서 사용된 대표적인 저장장치인 하드디스크를 기반으로 제안된 개념으로 하드디스크의 물리적 최소 저장단위인 섹터단위로 데이터 입출력을 명령한다. 하드디스크 섹터의 물리적 크기는 512byte이고 일반적으로 섹터 기반 파일시스템에 경우 512byte 크기의 물리위치를 하나의 논리블록주소로 사상한다[4]. 섹터 기반 파일시스템은 하드디스크의 저장용량이 커질수록 더 많은 물리위치를 사상해야하므로 대용량 하드디스크에 경우 운영체제가 관리해야하는 논리블록주소개수는 저장용량에 비례하여 증가한다. 일반적으로 사상정보는 시스템메모리에 구현 및 관리하기 때문에 논리블록주소개수에 증가는 시스템메모리 사용량 증가를 의미한다. 즉 이를 구현 및 관리하는데 많은 추가자원을 소모시킴으로 가능한 관리할 논리블록주소의 개수를 줄이는 것이 좋다.

제안기법은 낸드 플래시 메모리 기반 저장장치를 기존 섹터 기반 파일시스템에서 사용할 때 사상정보 관리개수의 증가로 인해 시스템 메모리 사용량이 증가하는 문제를 최대한 해결하기 위해 데이터 입출력 시 시간적집약성과 공간적집약성을 함께 고려한 사상기법을 제안한다. 제안기법은 기존 사상기법인 하이브리드기법 또는 로그블록 기법과 달리 블록 내 페이지 내에 다수의 사상정보를 일괄 저장하여 관리한다. 기존 기법에 경우 특정 블록 내 페이지에 하나에 데이터와 사상정보를 함께 관리함으로 일반적으로 라지 블록 내 페이지 개수가 64개일 때 64번에 무작위쓰기 또는 데이터갱신이 발생할 경우 해당 로그블록은 즉시 블록병합 대상이 된다. 즉 기존기법은 블록 내 사상정보 관리를 페이지단위로 관리함으로 빈번한 데이터갱신 발생 시 해당 블록이 처리할 수 있는 사상정보 관리에 임계치가 낮다. 반면 제안기법에 경우 데이터에 경우 라지 블록에 별도 저장하고 해당 데이터의 사상정보는 스몰 블록기반 색인 블록 내 페이지에서 별도로 저장하여 관리한다. 즉 제안기법은 색인 블록을 기반으로 페이지 단위로 사상정보만을 저장하여 이를 블록단위로 관리하는 것이다. 이를 위해 제안기법은 스몰 블록 내 하나에 페이지에 다수의 사상정보를 저장하여 관리하기 위한 순차 오프셋을 이용하며 블록 내에서 사상정보에 변경에 대응하는 여유페이지를 위한 역순 오프셋을 이용한다. 제안기법은 이를 통해 기존기법과 비교하여 블록단위 삭제 횟수가 적게 발생하며 사상정보 관리를 위해 필요한 시스템메모리 사용량이 낮은 장점이 있다[12].

II. Preliminaries

1. Related works

1.1 FTL(Flash Translation Layer)

FTL(Flash Translation Layer)은 주소 사상테이블과 가비지컬렉션 및 마모도 평준화 기능 이외에도 시스템 이상에 따른 복구기능 등 낸드 플래시 메모리 기반 저장장치 사용 시 발생하는 다양한 문제를 해결하고 효율적인 사용을 위한 필수적인 기능들을 포함한다. 일반적으로 FTL은 낸드 플래시 메모리 기반 저장장치 내부에 별도에 컨트롤러 칩에 존재하며 FTL은 주소 사상테이블 구현 및 관리를 위해 컨트롤러에 포함된 연산장치와 메모리를 사용한다.

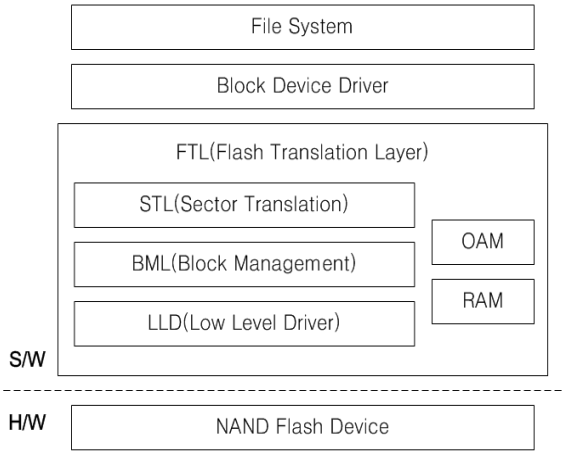

Fig. 1. Overview of FTL

FTL은 낸드 플래시 메모리에 구조적 특징으로 인해 발생하는 문제를 해결하기 위한 관리기법으로 구성되며 효율적인 구현 및 관리를 위해 계층적 구조로 이루어진다. FTL은 운영체제 및 파일시스템과 낸드 플래시 메모리 사이에 위치하는 시스템 소프트웨어이다. 응용프로그램은 파일시스템을 통해 데이터 입출력을 요청하고 파일시스템은 FTL을 통해서 실제 낸드 플래시 메모리에 데이터 입출력을 수행한다. FTL은 일반적으로 Fig. 1.과 같이 크게 세 부분으로 구성된다. 먼저 STL(Sector Translation Layer)는 주소 사상테이블과 가비지컬렉션, 마모도 평준화 기능을 담당하며 주소 사상테이블은 데이터 입출력 시 운영체제 및 파일시스템으로부터 논리적주소를 낸드 플래시 메모리의 물리적주소로 사상한다. 사상정보를 관리하기 위한 사상테이블은 시스템메모리 상에서 구현 및 관리됨으로 사상정보 구성과 개수에 따라 시스템메모리 사용량이 결정된다. 또한 주소 사상테이블은 낸드 플래시 메모리에 구조적 특징으로 인해 데이터갱신 시 추가동작이 발생함으로 이를 지연시키기 위해 갱신 데이터를 새로운 페이지에 기록한 뒤 사상정보를 갱신 데이터를 저장한 새로운 페이지주소로 갱신한다. BML(Bad-block Management Layer)은 낸드 플래시 메모리에 초기불량과 사용 중 불량을 관리하고 Error 처리 등을 담당한다. LLD(Low Level Driver)는 낸드 플래시 메모리를 사용하기 위해 Flash interface를 상위 계층에 제공한다.

1.2 Mapping Table

하드디스크는 플래시메모리 등장 이전 컴퓨팅시스템에 사용된 대표적인 저장장치이며 운영체제 및 섹터 기반 파일시스템은 하드디스크의 물리적 동작구조에 최적화되어 있다. 하드디스크 기반 운영체제 및 파일시스템은 데이터 입출력 시 논리블록주소를 이용하여 실제 저장장치에 물리위치를 지정한다. 논리블록주소에서 블록이란 하드디스크에 섹터를 의미하는 용어로써 논리블록주소는 하드디스크에 물리적 구조에 최적화된 논리적 섹터 주소 지정 체계이다. 페이지란 디스크기반 파일시스템에서 한 번에 데이터입출력에 단위를 의미한다. 반면 낸드 플래시 메모리는 하드디스크의 섹터구조와 달리 데이터 입출력 시 페이지 단위로 읽고 쓰기 때문에 기존 운영체제 및 파일시스템에 데이터 입출력 시 사용되는 논리블록주소 방식과 전혀 맞지 않다.

이를 해결하기 위해서 기존 운영체제와 파일시스템을 모두 낸드 플래시 메모리의 물리적 구조에 맞추어 변경하는 것은 사실상 불가능하며 비합리적이다. 대신 섹터 기반에 논리적인 입출력 명령을 실제 플래시메모리에 물리단위인 페이지단위 또는 블록단위로 변경하기 위한 주소변환 테이블 기법을 이용한다. 또한 주소 사상테이블은 낸드 플래시 메모리에 구조적 특징 중 데이터 제자리 갱신이 불가능함으로 발생하는 문제를 해결하기 위해 사용된다. 즉 주소 사상테이블은 제자리 갱신 시 발생하는 문제와 데이터 입출력 시 운영체제의 논리블록주소와 낸드 플래시 메모리의 물리주소를 서로 사상하여 관리하기 위해 필요하다. 일반적으로 주소 사상테이블은 컨트롤러 내 시스템메모리를 기반으로 구현 및 관리되기 때문에 가능한 시스템메모리 사용량을 최소화하는 것이 중요하다. 주소변환 테이블 또는 사상테이블을 구성하는 사상정보는 논리블록주소와 낸드 플래시 메모리의 물리주소의 쌍으로 이루어지며 사상정보들은 시스템메모리 테이블로 관리된다. 주소 사상테이블은 사상정보에 구성방식에 따라 크게 세 가지로 구분되며 사상정보에 구성은 논리블록주소와 사상되는 플래시메모리의 물리주소단위에 따라 나뉜다. 대표적인 주소 사상테이블 기법은 페이지단위 사상기법과 블록단위 사상기법 그리고 페이지단위와 블록단위에 장점을 모두 가지는 하이브리드 사상기법 또는 로그블록 기법이다[5, 6, 7].

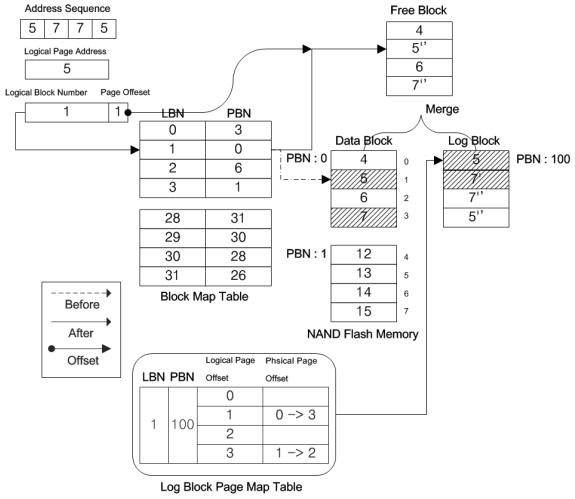

Fig. 2. Log Block Mapping Scheme

Fig. 2.는 로그블록 기법에 전체 동작과정을 보여준다. 로그블록 기법은 페이지단위 사상기법과 블록단위 사상기법의 장점을 모두 가진다. 즉 운영체제 및 파일시스템에 데이터 입출력 시 순차쓰기 요청에 대해 논리주소와 물리주소를 블록단위 사상 방식으로 관리하여 페이지단위 사상기법에 단점인 사상테이블에 필요한 메모리 사용량을 줄인다. 또한 무작위쓰기 요청에 대해 페이지단위 사상방식을 적용하여 블록단위 사상기법에 단점인 무작위쓰기 패턴에 많은 비용을 초래하는 블록단위 복사 및 삭제문제를 해결한다[10, 11]. 즉 로그블록 기법은 데이터블록과 로그블록으로 구성된다. 순차쓰기를 위한 데이터블록은 실제 데이터가 저장되는 블록이며 실제 사용자가 접근할 수 있는 블록을 말한다. 그리고 무작위쓰기를 위한 로그블록은 갱신 데이터가 저장되는 블록이며 로그블록은 사용자가 사용할 수 없는 공간이다. 특히 로그블록은 낸드 플래시 메모리에 구조적 특징 중 하나인 데이터 제자리갱신 불가에 대한 블록단위 복사 및 삭제를 해결하기 위해 사용된다[8].

Fig. 2.에서 쓰기요청 5, 7, 7, 5에 대해 논리블록번호와 페이지 오프셋을 계산한다. 이후 블록단위 사상테이블에 해당 논리블록번호와 사상된 물리블록번호를 찾는다. 논리블록번호 1번은 물리블록번호 0번과 사상됨으로 해당 데이터블록에 페이지 오프셋 1번에 데이터를 저장한다. 하지만 이미 해당 위치에 데이터가 저장되어 있기 때문에 데이터갱신이 발생하는 상황이며 이때 데이터블록과 연관된 로그블록을 할당받는다. 할당 받은 로그블록은 별도에 로그블록페이지 사상 테이블에 저장되며 해당 로그블록에 기존 페이지 오프셋 1번에 갱신 데이터를 저장한다. 이와 같이 나머지 쓰기요청을 처리한 후 데이터블록과 연관된 로그블록 간에 병합과정이 진행된다.

하지만 로그블록 기법은 데이터블록에 특정 데이터에 대한 갱신 요청이 지속되어 점차 로그블록 내 삭제상태 페이지에 개수가 줄어들 경우 특정 시점에 로그 블록 내 무효페이지를 일괄 삭제하여 삭제상태 페이지를 확보해야한다. 즉 로그블록 기법은 특정 시점에 Log block과 연관된 데이터 블록을 서로 병합하여 로그블록 내 삭제상태 페이지를 확보해야 한다. 로그블록 기법에 핵심은 병합과정에 효율성이며 병합과정에 효율에 따라 로그블록 기법에 전체성능이 결정된다[9].

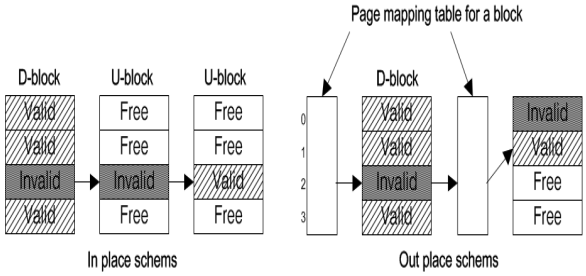

Fig. 3. Page Management schemes in a block

일반적인 데이터블록과 로그블록 간 병합과정은 아래와 같다. 병합과정은 일반적으로 다음과 같은 순서로 일어난다. 첫째. 병합하고자 하는 로그블록을 희생블록으로 선정한다. 둘째. 필요한 만큼의 Free Block을 할당받는다. 셋째. 데이터블록 및 로그블록의 유효한 데이터를 할당받은 Free block으로 복사한다. 병합과정은 병합 시 발생하는 Copy-back동작과 Erase동작에 발생횟수에 따라 전체 병합과 부분병합 그리고 스위치 병합으로 구분된다.

전체병합은 빈 블록을 선정하여 기존 데이터블록과 로그블록에 유효한 데이터를 복사하는 것이다. 전체병합은 기존 로그블록 및 데이터블록에 삭제상태 페이지가 충분하지 않을 경우 발생한다. 전체병합은 데이터블록 및 로그블록 내의 모든 유효페이지 개수만큼 Copy-back이 발생하며 기존 데이터블록과 로그블록을 삭제하기 때문에 최종 두 번에 블록단위 삭제가 발생한다. 부분병합은 데이터블록 내 유효페이지가 로그블록에 동일 오프셋 위치에 복사되는 것이다. 부분 병합은 희생블록으로 선정된 로그블록의 Free영역이 충분하여 데이터블록의 유효페이지가 모두 로그블록으로 복사된 후 기존 로그블록은 데이터블록으로 대체되며 새로운 로그블록이 할당된다. 부분 병합에 경우 데이터블록 내의 유효페이지 개수만큼의 복사가 발생하며 기존 데이터블록을 삭제함으로 최종 한 번에 블록단위 삭제가 발생한다. 스위치병합은 기존 로그블록 내 모든 페이지가 유효페이지이고 데이터블록 내 모든 페이지

가 무효페이지일 때 발생한다. 이러한 경우 단순히 로그블록을 데이터블록으로 대체함으로써 병합과정이 종료된다. 스위치병합은 최종 1회의 블록단위 삭제만이 요구된다.

III. The Proposed Scheme

1. Overall operation structure of the proposed technique

임베디드 시스템에 낸드 플래시 메모리 기반 저장장치 사용 시 기존 기법을 적용할 경우 빈번한 무작위쓰기 및 데이터갱신으로 인한 블록병합으로 블록단위 데이터복사 및 삭제동작과 로그블록에 저장된 사상정보 관리에 필요한 시스템메모리 사용량문제를 해결하기 위해 제안기법은 스몰 블록 기반 색인 블록 관리기법을 제안한다. 제안기법에 전체 동작구조는 다음과 같다.

제안기법은 먼저 데이터 입출력 시 실제데이터를 라지 블록에 순차적으로 저장한다. 이는 실제 데이터에 저장을 위해 별도로 저장위치를 검색하는 과정을 줄일 수 있어 보다 빠른 쓰기속도를 기대할 수 있다. 또한 실제데이터가 저장된 물리위치와 해당 쓰기요청 논리블록주소를 사상한 사상정보를 순차적으로 Hot data 판단기법을 이용하여 해당 쓰기요청 논리블록주소에 대한 발생패턴을 분석하여 Hot data 여부를 판단한다. Hot Data 판단기법은 상대적으로 데이터갱신을 빈번히 발생시키는 쓰기요청 논리블록주소를 Hot data로 판단하여 해당 논리블록주소를 별도에 시스템메모리에 구현 및 관리함으로 소수에 Hot data로 인한 빈번한 블록단위 복사 및 삭제동작을 줄일 수 있다.

또한 Hot data 판단에 제외된 Cold data에 해당 사상정보는 스몰 블록 기반 색인 블록에 블록단위로 관리된다. 제안기법에서 실제데이터와 사상정보를 별도로 관리하는 이유는 블록 내 사상정보에 관리개수를 높이기 위함이며 기존 로그블록 기법에 단점을 극복하기 위하여 사상정보를 색인 블록 기반 블록단위 매핑기법으로 관리한다. 최초 디스크기반 파일시스템으로부터 쓰기요청 시 해당 데이터는 라지 블록에 발생순으로 저장되며 해당 데이터가 저장된 물리위치와 쓰기요청 논리블록주소를 사상한 사상정보는 색인 블록 내 특정 페이지에 오프셋을 이용하여 관리된다. 제안기법은 다수에 사상정보를 페이지 내에서 함께 관리함으로 빈번한 무작위쓰기 및 데이터갱신에 대한 대비가 필요하다.

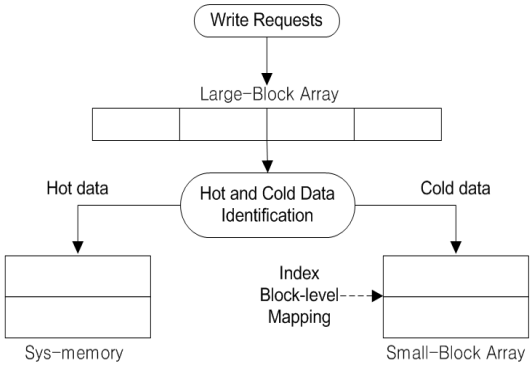

Fig. 4. The overall architecture of scheme

물론 Hot data 판단기법으로 Hot data를 판단하여 별도 관리하지만 시스템에 쓰기패턴은 수시로 변하기 때문에 이러한 변경가능성이 높은 사상정보를 즉시 색인 블록 내 특정 페이지에 다수 저장하는 것은 비효율적이다. 때문에 제안기법은 사상정보 발생 시 즉시 색인 블록 내 페이지에 쓰지 않고 임시적인 색인 블록 버퍼에 저장한다. 제한된 크기를 가지는 색인 블록 버퍼는 정해진 양에 사상정보를 임시 저장하여 관리하며 일정시간 이후 실제 페이지에 일괄 저장된다. 색인 블록 버퍼는 페이지에 사상정보를 기록하기 전 최대한 사상정보에 변경가능성을 줄이기 위한 제안기법이다. 또한 제안한 색인 블록 기반 블록단위 사상기법은 SLC 스몰 블록 기반으로 저장된 사상정보에 갱신 시 동일 블록 내 설정된 여유페이지로 이용하기 때문에 시스템에서 발생하는 쓰기패턴에 따라 갱신되는 사상정보를 별도 블록을 생성하지 않고 동일블록 내에서 페이지를 능동적으로 관리할 수 있다. 여유페이지는 데이터 입출력 패턴에 따라 가변적으로 개수가 증가 또는 감소함으로 데이터 입출력에 최대한 능동적으로 색인 블록을 관리하여 블록 내 페이지 공간 활용도를 높인다.

2. index block mapping technique based on small block

기존 블록단위 사상기법에 경우 블록 내 페이지 접근을 위해 고정 순차오프셋을 사용하기 때문에 사상정보 관리에 추가자원이 필요하지 않은 장점이 있지만 무작위쓰기 및 데이터갱신 발생 시 기존 고정 순차오프셋 유지를 위해 해당 블록이 아닌 별도에 로그블록을 생성하여 로그블록에 오프셋과 상관없이 데이터를 저장한다. 즉 무작위쓰기 및 데이터갱신 처리를 위한 로그블록 생성은 기존 하이브리드기법에 최대 장점이자 최대 단점으로 로그블록에 저장된 데이터에 사상정보를 관리하기 위한 별도에 시스템 메모리가 소모되며 다수에 무효페이지 발생으로 인한 빈번한 블록단위 복사 및 삭제발생에 원인이 된다.

제안기법은 이를 해결하기 위해 스몰 블록 기반 색인 블록을 이용한 블록단위 사상기법을 제안한다. 제안기법은 기존 블록단위 사상기법에 고정오프셋 유지로 인해 발생하는 문제를 해결하기 위해 앞서 언급한 것처럼 데이터 입출력 시 논리블록주소를 색인 블록과 연관된 색인 블록 버퍼를 이용하여 사상정보를 임시저장 한 후 일괄적으로 색인 블록 내 페이지에 저장함으로 빈번한 무작위쓰기 및 데이터갱신에 의한 무효페이지 발생을 최대한 줄여 블록단위 삭제동작을 최대한 지연시킨다.

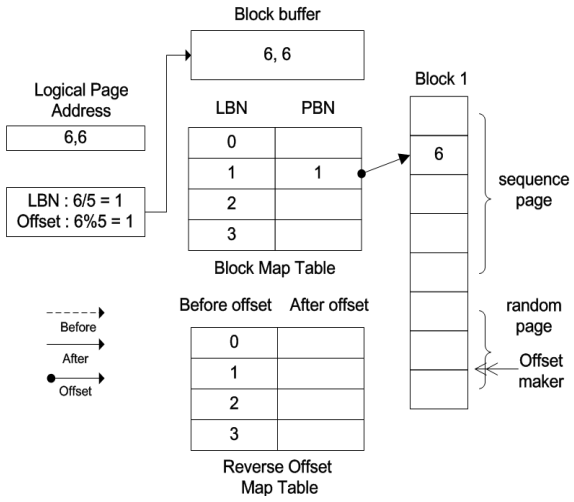

Fig. 5. Operational structure of proposed scheme

Fig. 5.는 스몰 블록 기반 색인 블록을 이용한 블록단위 사상기법에 동작과정을 설명한다. 제안기법은 데이터 입출력 시 해당 쓰기요청에 논리블록주소를 즉시 색인 블록 버퍼에 저장하여 시스템메모리상에서 관리하며 더 이상 여유 공간이 없는 경우 최초 7대 3비율로 나누어진 색인 블록 내 페이지에 일괄 저장한다.

Fig. 5.에서 최초 쓰기요청 발생 논리블록주소 6은 블록 내 여유페이지를 제외한 페이지개수 5로 나누어 논리블록주소 1과 오프셋 1을 계산한다. 이후 블록사상테이블을 이용하여 해당 논리블록주소에 사상된 물리블록주소 1을 계산한다. 이후 색인 블록 버퍼에 해당 쓰기요청에 논리블록주소 6을 저장한다. 두 번째 쓰기요청에 논리블록주소 역시 동일한 과정으로 색인 블록 버퍼에 저장된다.

제안기법은 앞서 설명한 것처럼 사상정보를 즉시 낸드 플래시 메모리에 저장하지 않고 대신 시스템메모리인 색인 블록 버퍼에 우선 저장한다. 이는 데이터 제자리갱신이 불가능한 특성을 고려한 것으로 무작위 쓰기 및 데이터갱신 요청 논리블록주소를 색인 블록 버퍼에 저장하여 일정시간 동안 관리하기 때문에 연속 데이터갱신이 발생하는 상황에서도 실제 색인 블록에 저장되는 논리블록주소는 마지막 데이터갱신 요청에 해당 논리블록주소 6뿐이다. 제안기법에 색인 블록 버퍼는 512byte에 고정크기에 시스템메모리이며 이는 한번 색인 블록 내 페이지에 관리할 수 있는 사상정보에 개수와 같다.

제안기법은 색인 블록 내 페이지에 Cold data로 판단된 쓰기요청에 논리블록주소를 저장한다. 즉 제안기법은 실제 Cold data가 저장된 라지 블록에 물리위치와 쓰기요청 논리블록주소를 사상하여 해당 논리블록주소를 bit shift 하여 해당 색인 블록 내에 페이지에 일괄 저장함으로 연속적으로 라지 블록에 저장된 데이터의 물리주소를 연속하는 논리 페이지주소에 대응시켜 라지 블록에 비순차로 저장된 데이터를 순차 쓰기요청으로 관리할 수 있다. 또한 색인 블록 내 순차 쓰기용도에 페이지 접근 시 오프셋 개념을 이용하여 실제 데이터의 물리주소와 사상된 논리블록주소를 별도로 저장하여 관리할 사상테이블이 필요없어 사상정보 관리에 필요한 추가적인 시스템메모리 사용량을 줄일 수 있다.

또한 색인 블록 버퍼는 변경가능성을 가진 쓰기요청에 논리블록주소를 즉시 데이터갱신이 불가능한 낸드 플래시 메모리에 저장하지 않고 일정시간 동안 색인 블록 버퍼에서 관리하며 변경가능성이 제거된 논리블록주소를 색인 블록 내 페이지 단위로 일괄 저장하는 Index 기법에 일종이다. 제안기법은 무작위쓰기 및 데이터갱신 시 로그블록을 생성하여 처리하는 기존기법은 빈번한 로그블록 생성에 하이브리드기법에 최대 단점을 해결하기 위해 Index 기법을 이용하며 실제 데이터를 저장하는 라지 블록에 물리주소와 별도로 사상정보를 저장하여 관리하는 색인 블록으로 나누어 동작한다. 색인 블록 버퍼는 연속적으로 발생하는 데이터 입출력에 의한 빈번한 사상정보 변경이 발생할 가능성을 시스템메모리에서 변경하여 최종적으로 변경된 사상정보만을 색인 블록 내 페이지에 저장하기 위함이다.

색인 블록 버퍼는 고정크기를 가지는 시스템메모리이며 색인 블록 버퍼는 동적으로 관리된다. 제안기법은 빈번한 무작위쓰기 및 데이터갱신 발생으로 색인 블록 버퍼에 더 이상 사상정보를 저장할 공간이 없을 경우 색인 블록 버퍼에 저장된 사상정보를 해당 색인 블록 에 저장한다. 제안기법은 이를 통해 색인 블록 버퍼에 필요한 최소한에 시스템메모리를 사용하여 변경되는 사상정보를 유동적으로 관리할 수 있다. 또한 제안기법에 색인 블록 내 최초 30% 비율로 여유페이지를 할당한다. 이러한 데이터 갱신용도에 페이지는 블록 내 가장 마지막 페이지부터 역순으로 대응시키며 색인 블록 버퍼에 의해 사상정보가 저장될 경우 이를 별도에 오프셋인 역순오프셋 기법이 요구된다.

3. Reverse offsets map table

블록단위 매핑기법은 블록 내 페이지 접근을 위한 고정 오프셋 유지를 위해 기존 블록에 유효페이지 또한 갱신 데이터가 기록된 블록에 동일 오프셋에 모두 복사해야한다. 또한 하이브리드기법에 경우에도 순차쓰기에 대해서 데이터블록에 오프셋을 이용함으로 별도에 사상정보를 관리할 필요가 없지만 무작위쓰기 및 데이터갱신 발생 시 별도에 로그블록을 이용하기 때문에 로그블록을 위한 별도에 오프셋관리를 위한 시스템메모리가 소모된다.

제안기법은 기존 기법에 단점을 보완하기 위해 색인 블록 버퍼를 이용하여 무작위쓰기 및 데이터갱신 발생으로 변경된 사상정보를 임시 저장하여 관리하며 최종적으로 동일 블록 내 페이지에 색인 블록 버퍼에 저장된 사상정보를 저장한다. 즉 기존기법처럼 무작위쓰기 및 데이터갱신 발생으로 인한 사상정보 변경 시 즉시 이를 페이지에 기록하는 것이 아닌 색인 블록 버퍼를 이용하여 일정 시간동안 지속적인 발생하는 무작위쓰기 및 데이터갱신에 사상정보를 시스템메모리상에서 저장 및 관리하는 것이며 최종적으로 변경된 사상정보를 동일 블록 내 페이지에 기록하여 이후 오프셋을 이용하여 접근함으로 기존 기법에 문제점을 해결한다.

즉 색인 블록 버퍼에 저장된 다수에 사상정보는 기존 오프셋과 상관없이 일괄 여유페이지에 저장되며 이렇게 저장된 사상정보에 접근하기 위해 역순 오프셋 테이블을 이용한다. 제안기법은 색인 블록을 최초 순차 쓰기용도에 페이지와 데이터갱신 용도에 페이지로 나누어 7대 3의 비율로 운용한다. 또한 블록 내 페이지에 접근하기 위해 블록 내 첫 페이지를 기준으로 하는 순차 오프셋과 블록 내 마지막 페이지를 기준으로 하는 역순 오프셋을 사용한다. 블록 내 첫 페이지를 기준으로 하는 순차 오프셋은 순차 쓰기용도에 페이지에 접근하기 위해 사용되며 이는 기존 블록 매핑 기법에서 사용되는 오프셋 개념과 동일하다.

역순 오프셋은 데이터갱신 용도의 페이지에 접근하기 위해 사용되는 오프셋을 말하며 제안기법은 블록 내 페이지를 7대 3비율로 나누어 순차 쓰기용도에 페이지와 데이터갱신 용도의 페이지로 사용한다. 데이터갱신 용도의 페이지는 호스트로부터 발생한 데이터갱신 요청에 대응하는 페이지로 제안기법은 블록 내 가장 마지막 페이지를 기준으로 대응한다. 이러한 데이터갱신 용도의 페이지에 접근하기 위해서는 별도에 오프셋이 필요하며 이를 역순 오프셋이라고 한다. 제안기법은 순차 오프셋을 이용하여 순차 쓰기용도에 페이지에 접근하는 경우 별도에 사상정보 테이블이 필요하지 않지만 역순 오프셋의 경우 별도에 오프셋 매핑 테이블이 필요하다. 오프셋 매핑 테이블은 데이터 갱신 요청 시 변경된 사상정보를 색인 블록 버퍼에 임시 저장하고 이후 여유페이지에 일괄 저장한다. 동일 블록 내에 페이지를 이용하여 데이터 갱신작업을 처리함으로써 해당 블록에 공간 활용도를 최대한 높일 수 있으며 이를 통해 전체 블록에 삭제 횟수를 감소시키는 장점이 있다.

4. Offset Marker

오프셋을 이용하는 매핑기법의 최대 장점은 논리페이지 주소와 물리페이지주소 간에 사상정보를 생략할 수 있다는 점과 항상 연속하는 물리 페이지에 연속하는 데이터가 기록되어있으므로 순차 쓰기요청에 좋은 성능을 보인다는 점이다. 하지만 이러한 장점은 곧 단점으로 작용한다. 왜냐하면 블록 내 페이지에 접근할 수 있는 유일한 방법은 고정된 오프셋이며 이러한 고정된 오프셋을 이용한 블록 내 페이지 접근이 가능하기 위해서는 항상 순차적으로 데이터가 기록되어 있어야하기 때문이다. 이를 해결하기 위해 제안기법은 앞서 언급한 것처럼 다중 오프셋을 사용한다. 더불어 제안기법은 운용 중 용도별 페이지의 비율을 가변적으로 변화시킴으로 최대한 데이터갱신 요청을 동일 블록 내에서 처리하도록 한다. 이러한 방법을 통해 블록 공간 활용도를 최대한 높일 수 있으며 이는 곧 전체 블록 삭제 횟수에 감소를 의미한다. 제안기법은 블록 내 페이지의 가변적인 운용을 위해 오프셋 마커를 이용한다. 오프셋 마커는 현재 블록 내 페이지에 사용 중인 여유페이지의 비율을 확인하기 위한 값이며 최초 블록 내 데이터갱신 쓰기용도에 페이지의 오프셋 값을 뜻한다.

제안기법은 섹터 기반 파일시스템으로부터 발생하는 다양한 쓰기요청 패턴에 능동적으로 대체하기 위해 데이터갱신 요청에 대응하는 페이지의 개수를 고정하지 않으며 특정 블록 내 페이지에 대한 데이터갱신 요청이 빈번히 발생할 경우 이에 대응하는 데이터갱신 용도에 페이지의 비율을 증가시킨다. 또한 색인 블록 버퍼에 저장된 변경된 사상정보를 최초 역순으로 여유페이지에 기록하며 이후 무작위쓰기 및 데이터갱신 발생이 지속되어 색인 블록 버퍼를 통한 여유페이지에 복사가 진행될 경우 이전 여유페이지를 무효화하며 이때 오프셋 마커는 현재 활성화 여유페이지를 가리키는 오프셋으로 동작한다. 또한 제안기법은 최초 정한 데이터갱신 용도에 페이지 비율을 넘어서는 경우 블록 내 가장 마지막 순차 쓰기용도에 페이지부터 순차적으로 데이터갱신 용도에 대응하는 페이지로 전환되며 해당 블록에 블록 경계포인터 값은 갱신된다.

즉 제안기법의 오프셋 마커는 현재 활성화 데이터갱신 용도에 페이지의 오프셋을 나타내며 이를 통해 호스트로

부터 발생하는 다양한 쓰기 패턴에 대해 효과적인 대응할 수 있다. 호스트로부터 발생하는 순차 쓰기요청 시 순차 쓰기용도에 페이지와 대응시킴으로 순차 오프셋을 이용한 사상정보의 메모리 사용량을 줄일 수 있으며 또한 무작위 쓰기 요청 시 블록 내 정해진 페이지 비율을 가변적으로 조정함으로 최대한 해당 블록 내 페이지에 갱신 데이터를 기록할 수 있으므로 블록 공간 활용도를 높일 수 있다. 또한 이를 통해 블록병합 시 최소한에 페이지 복사 및 블록 삭제가 가능하다. 제안기법에서 오프셋 마커는 쓰기, 읽기 요청에 모두 사용되며 해당 블록에 현재 오프셋 마커 값은 블록 내 별도에 페이지에 기록되어 유지된다.

5. Writing of the proposed technique

Fig. 6.은 제안한 스몰 블록 기반 색인 블록을 이용하여 섹터 기반 파일시스템으로부터 쓰기요청 발생 시 제안기법에 전체동작과정이다. 현재 블록 내 페이지는 8개이며 그중 5개는 순차 쓰기용도로 사용되고 나머지 3개는 데이터 갱신 용도에 여유페이지로 사용된다. 또한 현재 오프셋 마커의 값은 1이다. 제안기법은 쓰기요청 발생 시 해당 논리블록주소를 블록 내 페이지의 개수로 나누어 몫을 논리블록주소로 사용하며 나머지 값은 해당 논리블록의 페이지 순차오프셋 값으로 사용한다.

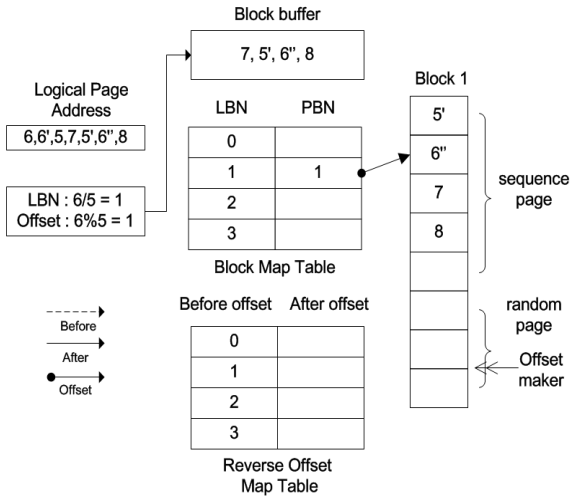

Fig. 6. Writing process in the proposed technique

Fig. 6.처럼 최초 빈 블록을 대상으로 첫 번째 쓰기요청 6을 bit shift 연산하여 논리블록주소 1과 페이지 순차오프셋 1값을 계산한다. 이후 해당 쓰기요청 6을 색인 블록 버퍼에 저장한다. 이어 두 번째 쓰기요청 6에 대해서 역시 동일한 방식으로 논리블록주소와 페이지 순차오프셋을 계산한다. 하지만 색인 블록 버퍼에 해당 쓰기요청에 논리블록주소가 저장되어 있으므로 시스템메모리인 색인 블록 버퍼 상에서 기존 쓰기요청에 논리블록주소는 갱신된다. 즉 제안기법에 핵심인 변경가능성을 가진 쓰기요청을 즉시 제자리 갱신이 불가한 블록 내 페이지에 저장하는 것이 아닌 색인 블록 버퍼를 이용하여 최대한 갱신 가능성을 제거하는 것이다. 이를 통해 Fig. 6.처럼 색인 블록 버퍼 상에서 갱신 처리된 쓰기요청은 3개로 이러한 쓰기요청이 발생할 무효페이지 생성을 사전에 최소화하는 것이다.

또한 Fig. 7.은 제안기법 적용 시 무효페이지가 발생하는 과정을 설명한다. 앞서 설명한 것처럼 제안기법은 쓰기요청 발생 시 해당 쓰기요청을 우선 색인 블록 버퍼에 저장한다. 해당 버퍼는 고정크기를 가지는 시스템메모리이며 512byte에 크기를 가진다. 즉 제안기법에 스몰 블록은 하나에 페이지의 크기가 512byte로 버퍼에 크기와 동일하여 한 번에 한 페이지에 저장할 수 있는 최대 쓰기요청 논리블록주소 개수는 64개이다. 버퍼 역시 발생한 쓰기요청을 64개 단위로 색인 블록 버퍼에 저장하며 이때 갱신이 발생하는 경우 추가적인 쓰기요청에 논리블록주소를 저장할 수 있다.

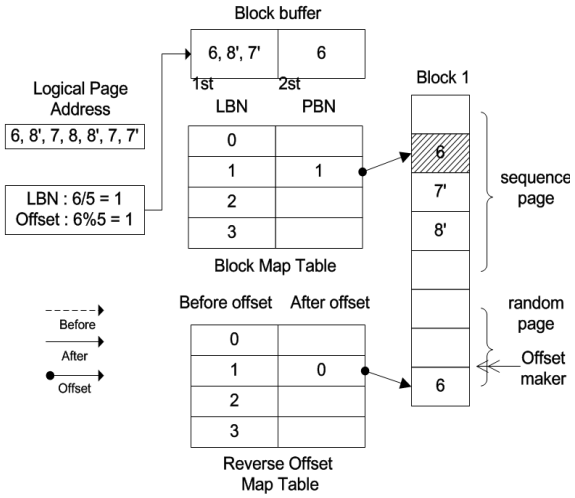

Fig. 7. Reverse offsets table

제안기법은 쓰기요청 발생 시 이를 색인 블록 버퍼에 저장한 이후 두 번째 쓰기요청에서 이전 쓰기요청 논리블록주소가 발생할 경우 이전 쓰기요청 논리블록주소를 저장한 페이지를 무효페이지 처리하며 해당 페이지에 저장된 다수에 쓰기요청 논리블록주소를 역순오프셋을 이용하여 동일 블록 내 여유페이지에 저장하게 된다. 제안기법은 무작위 쓰기 및 데이터갱신이 빈번히 발생하는 상황을 가정한다. 즉 앞서 설명한 Hot data 판단기법을 이용하여 빈번한 쓰기요청을 최대한 소거하고 이후 쓰기요청에 사상정보를 색인 블록 내 페이지에 저장하기 전 다시 버퍼를 이용하여 변

경가능성을 소거하는 것이다. 또한 색인 블록 내 페이지에 해당 구간에 발생한 쓰기요청에 논리블록주소를 일괄 저장함으로 블록 내 페이지의 공간 활용도를 높인다.

6. Index block Merge of the proposed technique

제안기법의 색인 블록 병합과정에 대해 설명한다. 제안기법에 색인 블록 내 삭제상태 페이지에 여유가 없는 경우 해당 블록 내 무효페이지를 제외한 유효페이지를 새로운 색인 블록에 페이지단위로 복사한다. 색인 블록 복사는 기존 블록의 순차 오프셋을 참고하여 순차 쓰기 용도의 페이지를 빈 블록에 복사하며 이후 역순 오프셋을 이용하여 데이터 갱신 용도에 페이지를 빈 블록에 복사한다. 또한 기존 색인 블록에 오프셋마커 역시 초기화되며 해당 블록과 연관된 블록상태 비트배열 역시 초기화 된다. 제안기법에 색인 블록 병합과정은 100% 부분병합으로 진행되며 이는 기존기법에 전체병합에 비해 블록삭제 횟수 감소에 이점이 있다. 즉 제안 기법은 색인 블록 내 페이지를 가변적으로 운용함으로 블록 내 페이지 활용도가 상대적으로 높으며 항상 삭제 블록의 수는 1개이다. 또한 페이지 접근 시 오프셋을 이용함으로 기존 블록 내 순차 쓰기 용도에 페이지가 다수인 경우 블록 복사 시 추가 동작을 줄일 수 있다.

또한 색인 블록 병합 시 초기화되는 블록상태 비트배열이란 색인 블록 내 유효페이지와 무효페이지를 구분하기 위한 비트배열을 말한다. 제안기법은 최초 블록 내 페이지 접근 시 순차 오프셋을 이용하여 접근하며 이때 해당 오프셋에 특정 페이지가 유효페이지와 무효페이지 여부를 확인하기 위해 직접 페이지단위 읽기동작이 요구된다.

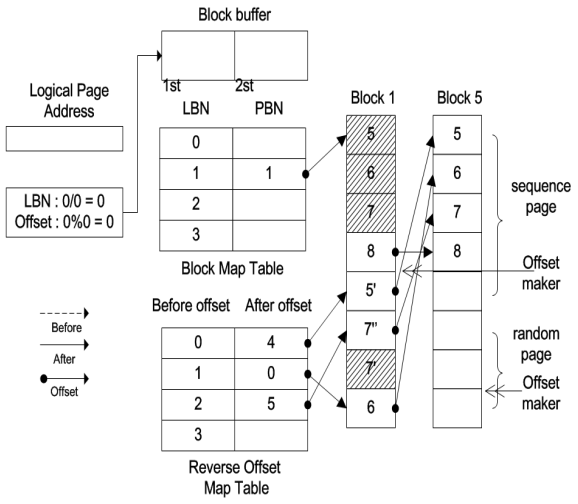

Fig. 8. Block copy of the proposed technique

낸드 플래시 메모리는 시스템메모리와 비교해 상대적으로 쓰기 및 읽기 속도가 느리기 때문에 페이지에 유효 또는 무효 상태를 페이지 읽기동작 없이 비트배열을 통해 확인할 필요가 있으며 이를 위해 블록상태 비트배열을 사용함으로 페이지단위 읽기 동작에 필요한 부가동작을 줄일 수 있다. 블록상태 비트배열을 이용하여 해당 블록 내에 유효페이지는 0 무효페이지를 1로 마킹하고 해당 블록이 삭제될 경우 상태 비트 배열 또한 0으로 초기화 된다. 즉 블록상태 비트배열은 빈번한 읽기동작 발생을 방지하기 위해 이용하며 해당 페이지의 유효페이지와 무효페이지 여부를 확인한다.

IV. Experimental

본 논문에서 제안한 스몰 블록 기반 색인 블록의 성능분석을 위해 낸드 플래시 메모리 기반 저장장치 시뮬레이터인 FlashSim을 이용하였다[13]. FlashSim은 낸드 플래시 메모리 하드웨어와 컨트롤러 그리고 FTL 부분으로 모듈화 되어있고 실제 낸드 플래시 메모리 기반 저장장치에 동작 구조와 동일하여 실제 장비로 진행하기 어려운 다양한 실험을 빠르게 진행할 수 있어 많은 연구에서 이용된다. 또한 실험에서 사용한 낸드 플래시 메모리의 성능은 표1.와 표2.과 같다.

1. Experimental environment and Trace file spec

Table 1. 1Gb NAND Flash Large-Block Array Organization

Total capacity	4GB
Block count	4096
Page count per block	64
Page size	2KB
Erase time	2ms
Write time	5.20MB/s
Read time	16.13MB/s

Table 2. 1Gb NAND Flash Small-Block Array Organization

Total capacity	1GB
Block count	8192
Page count per block	32
Page size	512byte
Erase time	2ms
Write time	2.33MB/s
Read time	12.65MB/s

라지 블록 낸드 플래시 메모리에 경우 총 저장용량 4GB이며 블록에 개수는 4096개이고 블록 당 페이지 개수는 64개이며 블록 내 페이지에 크기는 2KB이다.

스몰 블록에 경우 총 저장용량 1GB이며 블록에 개수는 8192개이고 블록 당 페이지 개수는 32개이며 블록 내 페이지에 크기는 512byte이다.

또한 실험에 사용한 I/O Trace file은 TPC-C 벤치마크 테스트모델을 이용하였다. TPC는 트랜잭션 처리 및 데이터베이스의 벤치마크를 정의하고 Disk I/O 및 S/W를 포함하는 전체 System 성능측정에 사용된다. 본 실험에서는 TPC-C 벤치마크 모델을 사용하였다. TPC-C 벤치마크는 보통 데이터베이스의 성능을 측정하는데 사용되지만 본 실험에서는 I/O trace를 낸드 플래시 메모리에 데이터 입출력 성능평가 실험을 위한 입력데이터로 사용한다.

실험은 기존 로그블록기법과 제안기법인 색인 블록을 이용한 블록단위 사상기법을 각각 적용하여 로그블록과 색인 블록의 생성개수에 결과를 분석하고 또한 기존 로그블록 기법에 데이터갱신 시 발생되는 로그블록에 생성에 대해 제안기법과 기존기법을 각각 적용하여 블록생성 실험을 진행하여 결과를 분석한다. 특히 해당 로그블록 생성으로 발생하는 데이터블록과에 빈번한 병합발생 시 해당 페이지 내 유효 및 무효페이지에 개수를 분석하며 제안한 기법에 색인 블록 적용 시 효율성에 대해 블록삭제 실험을 진행한다.

2. Experimental environment and Trace file

스몰 블록 기반 색인 블록을 이용한 블록단위 사상기법과 기존 로그블록기법에 실험을 위해 사용한 I/O Trace는 첫 번째 실험과 동일하다. 다만 해당 I/O Trace는 무작위 쓰기 비율이 순차쓰기 비율보다 압도적으로 많아 실험결과 분석에 편의성을 위해 해당 I/O Trace에 일부분인 1만회 쓰기요청을 입력 값으로 사용하였다.

3. Block generation experiment result

블록생성 실험결과는 제안기법과 기존기법 각각 총 쓰기횟수에 대비하여 이를 처리하기 위해 생성된 블록에 개수를 의미한다. 표 3.에서 제안기법은 총 쓰기횟수 4475회이다. 반면 입력 쓰기요청에 횟수는 9,876회로 이러한 차이가 발생한 것은 앞서 설명한 Hot data 판단기법을 통해 판단된 Hot data에 해당하는 쓰기요청이 제외되었기 때문이며 제안기법에 색인 블록 버퍼 상에서 실제 색인 블록에 저장되기 전 소거된 쓰기요청을 모두를 반영하였기 때문이다. 기존기법에 경우 입력 쓰기횟수와 동일한 쓰기횟수인 9,876회를 기록하였다.

Table 3. Block creation experiment result

	Total number of writes	Total number of block creation	Number of log blocks	Total accumulated invalid pages
Proposed technique Experiment result	4,475	1,182	none	532
Log block experiment result	9,876	3,081	533	5,877

표 3.에서 제안기법은 총 쓰기횟수 4475회에 대해 총 색인 블록 1182개를 생성하여 처리하였다. 제안기법에서 색인 블록 생성개수에 의미는 총 누적 invalid 페이지 생성개수에 따라 달라진다. 즉 제안기법은 색인 블록 내 페이지에 저장된 사상정보 중 데이터갱신이 발생할 때 이를 동일 색인 블록 내 여유페이지에 저장한다. 즉 이때 기존 사상정보가 저장된 페이지는 invalid 페이지로 기록되는 것이다. 이러한 이유로 제안기법에 성능은 총 블록생성 개수 대비 총 누적 invalid 페이지 개수가 적을 때 좋은 성능을 의미하고 반대에 경우 좋지 않은 성능임을 의미한다. 이는 앞서 동작하는 Hot data 판단기법에서 빈번한 데이터갱신을 발생시키는 논리블록주소를 정확히 판단하지 못했거나 또는 색인 블록 버퍼에서 이를 소거하지 못하여 실제 Hot data가 색인 블록 내 페이지에 저장된 이후 데이터갱신이 발생하였음을 의미한다. 하지만 실험결과는 제안기법은 총 블록생성 개수 대비 총 누적 invalid 페이지 개수가 약 50% 정도 적게 발생하였음으로 실험에 사용된 입력 값에 무작위쓰기 비율에 대비하여 좋은 성능을 보인다.

제안기법에 성능은 기존기법에 결과와 비교할 때 더욱 확실하게 알 수 있다. 기존기법에 경우 총 쓰기횟수에 소거 없이 9,876회 쓰기요청에 대해 총 블록생성 개수는 3081개이고 총 누적 invalid 페이지 개수는 5,877이다. 앞서 설명한 것처럼 로그블록 기법은 데이터블록에 실제데이터를 저장하며 데이터블록에 오프셋으로 해당 데이터의 위치를 알 수 있다. 즉 데이터블록 내 페이지에 개수는 한 블록에서 발생하는 데이터갱신에 허용치를 의미한다. 블록 당 페이지개수를 64개로 할 때 블록 당 데이터갱신 역시 64회로 판단할 수 있다. 물론 해당 데이터갱신은 연관된 로그블록을 이용하여 처리하지만 결국 동일한 라지 블록임으로 하드웨어 스펙은 동일하다. 기존 기법에 로그블록 개수는 533개로 총 블록생성 개수 3081개 중 로그블록 개수를 의미함으로 실제 데이터블록에 생성개수는 2,548개이다. 즉 2,548개에 데이터블록과 연관된 로그블록에 개수가 533개임을 의미한다.

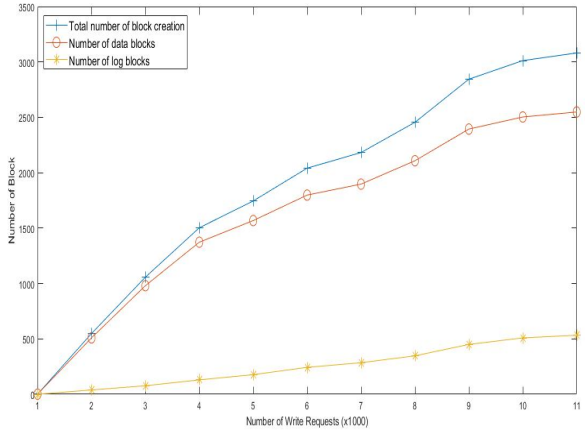

Fig. 9. Block generation when applying the log block technique

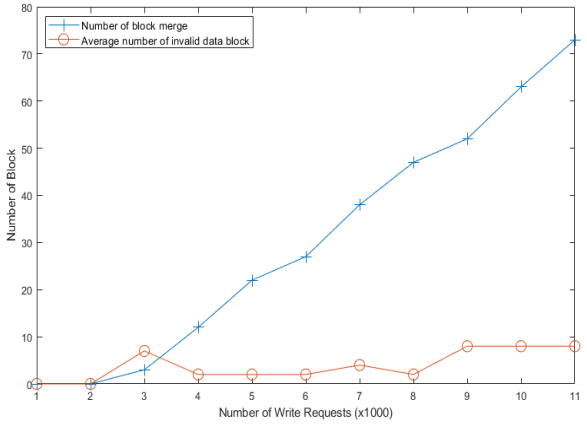

Fig. 10. Average invalid page when merging blocks

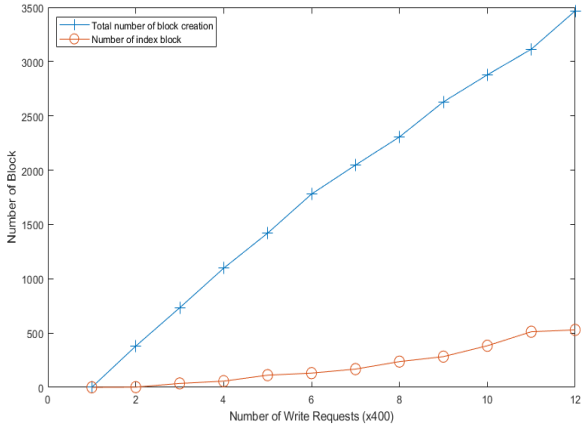

Fig. 11. Block creation when applying the proposed technique.

Fig. 9.은 기존기법에 총 블록생성 개수 중 데이터블록과 로그블록에 생성비율을 쓰기요청 당 누적그래프로 나타낸다. 그래프를 통해 알 수 있듯이 생성된 블록에 대부분은 데이터블록으로 이러한 결과는 무작위쓰기 비율이 압도적으로 많은 실험 입력 값에 의해 대부분에 데이터블록 내 소수에 페이지만을 가지는 데이터블록이 대부분임을 의미한다. 이는 Fig. 10.를 통해 알 수 있다. Fig. 10.은 기존기법 적용 시 생성된 데이터블록 내 valid 페이지의 비율을 나타낸 것으로 데이터블록 내 생성 페이지는 평균 8개를 내외를 가지는 것을 알 수 있다. 즉 대부분에 쓰기요청이 무작위쓰기 및 데이터갱신 요청인 입력 값에 쓰기요청으로 기존기법은 별도에 방법 없이 해당 무작위쓰기 및 데이터갱신을 로그블록에서 대부분 처리하고 데이터블록은 대부분 비어있는 것을 의미한다. 즉 기존기법에 경우 무작위쓰기 및 빈번한 데이터갱신 요청으로 인해 블록 내 페이지 공간 활용도가 낮음을 알 수 있다.

Fig. 11.은 제안기법에 총 블록생성 개수 대비 블록 내 유효페이지와 무효페이지의 비율을 나타낸다. 특히 쓰기요청 1000회 이하에 경우 블록 내 무효페이지에 개수는 평균 15개로 색인 블록 내 유효페이지에 개수가 무효페이지에 개수에 비해 월등히 높음을 알 수 있다.

제안기법에 경우 다수에 사상정보를 페이지에 저장함으로 페이지 내 사상정보 중 갱신이 발생으로 인해 페이지 쓰기횟수가 증가할 것으로 예상할 수 있지만 Hot data 판단기법과 색인 블록 버퍼로 무작위쓰기 및 데이터갱신을 발생시키는 빈번한 데이터갱신 쓰기요청 대부분을 충분히 소거하여 블록 내 무효페이지 개수는 크게 증가하지 않음을 알 수 있다. 즉 제안기법은 기존 로그블록 기법에 최대 단점인 로그블록 생성 및 병합을 최소화하기 위해 색인 블록 사상기법을 제안하였다. 제안기법은 무작위쓰기 및 데이터갱신에 대해 별도에 로그블록을 생성하여 처리하지 않고 순차쓰기 및 무작위쓰기 그리고 데이터갱신에 대한 데이터를 일괄적으로 Large블록에 발생순서 대로 저장한다. 또한 제안기법은 Large블록에 저장된 실제 데이터의 Index를 Small 블록을 이용하여 관리한다. 제안기법은 실제 데이터의 저장위치인 Index 정보만을 Small 블록으로 관리하기 때문에 Small블록 내 한 페이지에 다수에 Index 정보를 저장할 수 있어 블록생성 개수를 줄일 수 있는 장점이 있고 또한 무작위쓰기 또는 데이터갱신 발생 시 색인 블록 버퍼에서 임시저장 하여 관리함으로 invalid 페이지에 발생개수를 최소화할 수 있다.

제안기법에 색인 블록 버퍼는 Small 블록에 한 페이지 크기이며 무작위쓰기 및 데이터갱신에 대한 Index가 저장되어 색인 블록 버퍼에 여유가 없을 경우 이를 블록 내 페이지에 기록하게 된다. 즉 색인 블록 버퍼를 통해 무작위쓰기 및 데이터갱신에 대한 변경을 시스템메모리에서 최대한 허용한 후 최종적으로 페이지에 저장함으로 무작위쓰기 및 데이터갱신 시 최대한 블록단위 삭제발생을 억제

한다. 또한 제안기법은 Index 정보를 블록단위로 관리하기 때문에 순차쓰기에 대한 데이터 Index에 대해 별도에 관리가 필요하지 않다. 또한 제안기법은 블록 내 사용가능 페이지가 없는 경우 항상 부분병합과정을 진행하기 때문에 기존기법에 전체병합과 비교해 병합과정에 효율성을 높인다. 결론적으로 제안기법은 로그블록 기법과 비교해 생성된 블록에 개수가 적고 무작위쓰기 및 데이터갱신을 위한 로그블록이 필요하지 않으며 Hot data 판단기법을 통해 빈번한 데이터갱신을 발생시키는 논리블록주소를 사전에 소거하여 데이터갱신을 줄여 invalid 페이지 발생 개수 역시 기존기법 보다 낮음을 확인하였다. 또한 제안기법은 별도에 병합과정이 필요하지 않지만 블록 내 페이지가 더 이상 없는 경우 결국 블록 내 invalid 페이지를 정리해야한다. 하지만 제안기법은 최악에 경우에도 블록 내 부분병합으로 해결이 가능하다.

4. Block deletion experiment result

블록삭제 실험결과는 제안기법과 기존기법 각각 총 쓰기횟수에 대비하여 이를 처리하기 위해 제안기법에 색인 블록에 삭제횟수와 기존기법에 로그블록 삭제횟수를 의미한다. 실험결과 제안기법에 색인 블록 삭제횟수는 49회이고 기존기법에 로그블록 삭제횟수는 73회이다. 또한 제안기법에 색인 블록 삭제 시 평균 invalid 페이지개수는 4.16개이고 총 블록 내 생성된 누적 valid 페이지 개수는 3,519개 이며 블록 내 생성된 누적 invalid 페이지 개수는 532개이다. 기존기법에 경우 병합 시 데이터블록 내 평균 invalid 페이지 개수는 8개 이고 총 블록 내 생성된 누적 valid 페이지 개수는 3927개이며 블록 내 생성된 누적 invalid 페이지 개수는 5377개이다.

제안기법에 색인 블록 삭제 횟수에 의미는 다음과 같다. 제안기법은 기존기법에 문제점인 빈번한 데이터갱신으로 인해 발생하는 병합과정을 줄이기 위해 실제 데이터를 라지 블록에 저장하고 해당 물리위치와 사상된 사상정보를 색인 블록에서 별도 관리한다. 즉 제안기법은 색인 블록 내 무효페이지 생성으로 더 이상 여유페이지가 없을 경우 색인 블록 내 부분병합과정이 발생한다. 색인 블록 삭제 횟수는 결국 제안기법에 부분병합 발생 횟수를 의미한다.

제안기법 색인 블록은 스몰 블록에 구조적 특징을 이용하여 무작위쓰기 및 데이터갱신 시 블록 내 여유페이지를 이용하여 처리한다. 즉 블록 내 페이지에 용도를 쓰기요청 패턴에 맞춰 가변적으로 운용할 수 있는 장점이 있다. Fig. 12.에서 쓰기요청 1400번에서 2200번 대까지는 해당 색인 블록은 삭제 시 블록 내 평균 invalid 페이지의 개수가 5개 미만인 것을 알 수 있으며 해당 쓰기구간에 쓰기패턴은 무작위쓰기 및 데이터갱신보다 순차쓰기 비율이 더 높아 해당 색인 블록은 블록 내 무효페이지보다 유효페이지의 비율이 더 높은 것이다.

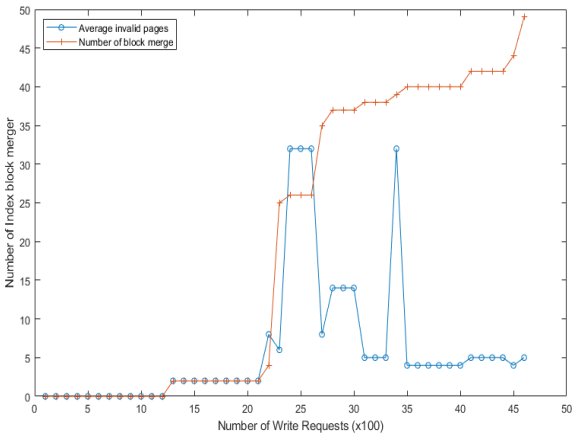

Fig. 12. Average invalid pages when merging index blocks

반면 쓰기요청 2300번에서 2500번 대까지는 해당 색인 블록은 삭제 시 블록 내 평균 invalid 페이지의 개수가 33개인 것을 알 수 있다. 즉 해당 쓰기구간에 쓰기패턴은 무작위쓰기 및 데이터갱신 비율이 더 높아 해당 색인 블록은 블록 내 유효페이지의 비율보다 무효페이지의 비율이 더 높은 것이다. 결론적으로 제안기법에 색인 블록은 쓰기패턴에 따라 블록 내 페이지의 공간을 활용하는 가변적 페이지 활용에 장점이 있다. 반면 Fig. 10.처럼 기존기법인 로그블록 기법에 경우 블록 내 공간 활용도가 제안기법에 비해 떨어지는 것을 알 수 있다. 로그블록에 경우 블록병합은 데이터블록과 연관된 로그블록 간에 발생하며 쓰기패턴이 무작위쓰기 및 데이터갱신이 많을 경우 데이터블록 내 페이지에 공간은 거의 비어있는 상태로 로그블록과 전체병합이 발생하게 된다. 기존기법에 경우 Fig. 10.에서 블록병합 시 데이터블록에 평균 페이지 개수는 8개로 이는 쓰기요청에 패턴이 무작위쓰기 및 데이터갱신이 많을 경우 데이터블록 내 페이지는 거의 활용되지 못하는 것을 의미한다.

또한 제안기법에 색인 블록 삭제 횟수는 총 49회이다. 제안기법은 앞서 Hot data 판단기법과 색인 블록에 사상정보를 저장하기 전 색인 블록 버퍼를 이용하여 최대한 무작위쓰기 및 데이터갱신을 소거하는 기법을 이용하였다. 결국 스몰 블록 기반 색인 블록 내 페이지에 다수에 사상정보를 다수 저장하는 것은 무작위쓰기 및 데이터갱신에 취약한 방법일 수 있지만 제안기법은 이를 극복하기 위해 Hot data 판단기법과 색인 블록 버퍼 기법을 제안하였으

며 이를 통해 장점을 극대화 할 수 있다. 기존기법에 경우 무작위쓰기 및 데이터갱신을 로그블록만을 이용하여 처리함으로 동일한 쓰기요청에 대해 블록삭제 횟수는 73회로 제안기법 보다 더 많은 블록삭제 횟수를 기록하였다.

V. Conclusions

본 논문은 낸드 플래시 메모리 기반 저장장치 사용을 위해 기존 로그블록 기법 적용 시 무작위쓰기 및 빈번한 데이터갱신 발생으로 인한 잦은 블록병합 발생과 로그블록에 필요한 시스템메모리 사용량 문제를 해결하기 위해 SLC type 스몰 블록 기반 색인 블록을 이용한 사상기법을 제안하였다.

제안기법은 실제 Data와 해당 쓰기요청에 사상정보를 각각 라지 블록과 스몰 블록 기반 색인 블록에 별도 저장하여 관리함으로 색인 블록 내 페이지에 쓰기요청에 시간적집약성과 공간적집약성을 반영한 다수에 사상정보를 일괄 저장함으로 색인 블록 내 페이지 공간 활용도를 높였으며 색인 블록 버퍼를 이용하여 하나에 페이지에 저장할 수 있는 크기에 사상정보를 우선 시스템메모리에서 관리함으로 사상정보에 변경패턴에 최대한 능동적으로 대응할 수 있다. 이를 통해 동일 블록 내 여유페이지를 이용하여 무작위쓰기 및 데이터갱신을 처리함으로 블록 내 페이지 공간 활용도를 높여 블록병합과정 발생을 최대한 지연시킴으로 블록단위 복사 및 삭제횟수를 감소시킨다. 제안기법에 색인 블록 사상기법은 기존 로그블록 기법과 비교하여 블록병합 횟수를 약 32% 줄였으며 무효페이지 발생에 따른 시스템메모리 사용량에 경우 기존기법 대비 90% 감소하였다. 제안기법에 총 시스템메모리 사용량에 경우 Hot data 판단기법에 사용된 시스템메모리를 고려할 경우 전체 시스템메모리 사용량은 약 10%에 시스템메모리 사용량 감소에 효과를 보인다.

REFERENCES

[1] Y. Takai, M. Fukuchi, R. Kinoshita, C. Matsui, K. Takeuchi, “Analysis on Heterogeneous SSD Configuration with Quadruple-Level 셀 (QLC) NAND Flash Memory,” 2019 IEEE 11th International Memory Workshop (IMW), June 2019. DOI: 10.1109/IMW.2019.8739689

[2] S.W Choi, K.T Park, M. Passerini, H.J Park, D.Y Kim, C.H Kim, K.W Park, J.W, Kim “A 셀 current compensation scheme for 3D NAND FLASH memory,” 2015 IEEE Asian Solid-State Circuits Conference (A-SSCC), January 2016. DOI: 10.1109/ASSCC.2015.7387432

[3] Hang-Ting Lue, “3D NAND flash memory,” https://patents.google.com/patent/US9018047B2/en

[4] A. Soga, C. Sun, K. Takeuchi, “NAND flash aware data management system for high-speed SSDs by garbage collection overhead suppression,” 2014 IEEE 6th International Memory Workshop (IMW), May 2014. DOI: 10.1109/IMW.2014.6849374

[5] T.S Chung, D.J Park, S.W Park, D.H Lee, S.W Lee, and H.J Song, “A survey of Flash Translation Layer,” Journal of Systems Architecture, Vol. 55, No. 5, pp. 332-343, April 2009. DOI: https://doi.org/10.1016/j.sysarc.2009.03.005

[6] S.J Kwon, A. Ranjitkar, Y.B Ko, and T.S Chung, “FTL algorithms for NAND-type flash memories,” Design Automation for Embedded Systems, Vol. 15, No. 3-4, pp. 191-224, March 2011. DOI: https://doi.org/10.1007/s10617-011-9071-9

[7] D. Ma, J. Feng, and G. Li, “A Survey of Address Translation Technologies for Flash Memories,” ACM Computing Surveys (CSUR), Vol. 46, No. 3, pp. 1-39, January 2014. DOI: https://doi.org/10.1145/2512961

[8] Y. Wang, Z. Qin, R. Chen, Z. Shao, Q. Wang, S. Li, L.T. Yang, “A Real-Time Flash Translation Layer for NAND Flash Memory Storage Systems,” IEEE Transactions on Multi-Scale Computing Systems Vol. 2, No. 1, pp. 17-29, March 2016. DOI: 10.1109/TMSCS.2016.2516015

[9] Y. Hu, H. Jiang, D. Feng, L. Tian, S. Zhang, J. Liu, W. Tong, Y. Qin, L. Wang, “Achieving page-mapping FTL performance at block-mapping FTL cost by hiding address translation,” 2010 IEEE 26th Symposium on Mass Storage Systems and Technologies (MSST), pp. 1-12, June 2010. DOI: 10.1109/MSST.2010.5496970

[10] S.W Lee, D.J Park, T.S Chung, D.H Lee, S.W Park, 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, July 2007. DOI: https://doi.org/10.1145/1275986.1275990

[11] D.W Jung, J.U Kang, H.S Jo, J.S Kim, and J.W Lee, “Superblock FTL: A Superblock-Based Flash Translation Layer with a Hybrid Address Translation Scheme,” ACM Transactions on Embedded Computing Systems (TECS), Vol. 9, No. 4, April 2010. DOI: https://doi.org/10.1145/1721695.1721706

[12] S.W Lee, K.W Ryu, “Block Unit Mapping Technique of NAND Flash Memory Using Variable Offset,” Journal of The Korea Society of Computer and Information Vol. 24 No. 8, pp. 9-17 August 2019. DOI : http://dx.doi.org/10.9708/jksci.2019.24.08.009

[13] Y.J Kim, B. Tauras, A. Gupta, B. Urgaonkar, “FlashSim: A Simulator for NAND Flash-Based Solid-State Drives,” 2009 First International Conference on Advances in System Simulation, October 2009. DOI: 10.1109/SIMUL.2009.17

Authors

Seung-Woo Lee received the M.S and Ph.D. degree in Computer Science and Engineering form Kyungpook National University, South Korea, in 2013 and 2020, respectively. He is currently a Professor in the Department of Aeronautical Software Engineering, Kyungwoon University. He is current interests include embedded system and flash memory based storage system.

Se-jin Oh received the M.S. and Ph.D. degrees in Computer Science and Engineering from Kyungpook National University, Korea, in 2011 and 2014, respectively. Dr. Oh joined Advanced Technology Center at Samsung Techwin in 2014. He is currently a Professor in the Department of Aeronautical Software Engineering, Kyungwoon University. He is interested in Embedded System, Information Security, IoT System.