Journal of The Korea Society of Computer and Information
Vol. 22 No. 5, pp. 1-9, May 2017
www.ksci.re.kr
https://doi.org/10.9708/jksci.2017.22.05.001

Sampling-based Block Erase Table in Wear Leveling Technique for Flash Memory

Seon Hwan Kim*, Jong Wook Kwak**

Abstract

Recently, flash memory has been in a great demand from embedded system sectors for storage devices. However, program/erase (P/E) cycles per block are limited on flash memory. For the limited number of P/E cycles, many wear leveling techniques are studied. They prolonged the life time of flash memory using information tables. As one of the techniques, block erase table (BET) method using a bit array table was studied for embedded devices. However, it has a disadvantage in that performance of wear leveling is sharply low, when the consumption of memory is reduced. To solve this problem, we propose a novel wear leveling technique using Sampling-based Block Erase Table (SBET). SBET relates one bit of the bit array table to each block by using exclusive OR operation with round robin function. Accordingly, SBET enhances accuracy of cold block information and can prevent to decrease the performance of wear leveling. In our experiment, SBET prolongs life time of flash memory by up to 88%, compared with previous techniques which use a bit array table.

▸ Keyword : Flash Memory, Wear Leveling, Storage Device, Block Erase Table, Round Robin

I. Introduction

수년간 하드 디스크가 저장장치로 많이 사용되어 왔다. 하지만 최근에는 플래시 메모리 기반 저장장치가 데이터베이스 시스템과 서버 분야에서 각광받고 있다. 플래시 메모리는 하드 디스크와 비교하여 빠른 접근속도를 보장하고 내진성과 전력 소비가 낮다는 장점이 있다. 또한 가격이 저렴해지고 있어 보조 저장장치로 많이 사용되고 있다.

플래시 메모리는 각 셀마다 비트를 저장할 수 있으며, 읽기와 쓰기의 단위에 따라 NAND와 NOR로 구분된다. NOR는 바이트 및 워드와 같이 읽기/쓰기 단위가 작으며, NAND는 상대적으로 읽기/쓰기 단위가 큰 페이지 단위를 사용한다. 최근 NAND 페이지의 용량은 2KB~8KB를 사용하고 있다. NAND는 NOR보다 직접도 면에서 유리하고 가격이 저렴하여 대용량 저장장치에 많이 사용되고 있다[1].

플래시 메모리는 여러 개의 블록으로 구성되어 있으며, 블록은 다시 페이지들로 구성되어 있다. 페이지는 각 비트를 저장할 수 있는 셀들의 배열로 되어 있다. NAND를 기준으로 쓰기/읽기의 단위는 페이지이고 지우기 단위는 블록이다. 이와 같이 플래시 메모리는 쓰기/읽기와 지우기의 단위가 다르다는 점 이외에도 쓰기 전에 지우기를 수행(erase before write)해야 하는 단점이 있어 데이터를 갱신할 때 문제가 된다. 한 바이트의 데이터를 갱신하기 위해서는 지우기 연산의 단위가 블록이기 때문에 같은 블록에 속해 있는 모든 데이터를 삭제하고 갱신된 데이터와 함께 이를 다시 써야 함으로 하드 디스크와 비교할 때 비효율적인 방식을 사용해야 한다. 이는 플래시 메모리에서 제자리 덮어쓰기 불가 문제(non-in-place problem)로 불린다.

제자리 덮어쓰기 문제를 해결하기 위해 갱신된 데이터를 다른 주소에 기록하고, 그 주소를 테이블에 저장하여 참조할 때 이를 이용한다. 이런 주소 매핑 테이블을 이용한 기법은 주소 단위에 따라 페이지 매핑, 블록 매핑, 하이브리드 매핑으로 나뉜다. 페이지 매핑은 각 페이지의 변경정보를 논리적인 주소와

• First Author: Seon Hwan Kim, Corresponding Author: Jong Wook Kwak
*Seon Hwan Kim (amexist@ynu.ac.kr), Dept. of Computer Engineering, Yeungnam University
**Jong Wook Kwak (kwak@yu.ac.kr), Dept. of Computer Engineering, Yeungnam University
• Received: 2016. 10. 25, Revised: 2016. 11. 15, Accepted: 2016. 12. 05.
• This research was supported by Basic Science Research Program through the National Research Foundation of Korea(NRF) funded by the Ministry of Education(No. NRF-2014R1A1A2057146).

물리적인 주소의 집합으로 구성하고 기록한다. 블록 매핑은 블록 주소만 사용하고 하이브리드 매핑은 페이지 주소와 블록 주소를 같이 사용한다.

주소 매핑 기법과 같이 플래시 메모리의 단점을 해결하고 하드 디스크 기반 응용 프로그램들을 구동할 수 있게 하는 소프트웨어 계층을 FTL(Flash Translation Layer)이라고 한다. FTL은 주소 매핑 테이블 이외에도 갑작스런 시스템 고장에 대비한 복구 기능, 무효 페이지(invalid page)를 수집하여 가용 페이지(free page)로 전환하는 가비지 컬렉션(garbage collection), 마모도 평준화(wear leveling) 등의 기능을 포함하고 있다[2-4].

마모도 평준화 기법을 설명하기 위해 플래시 메모리의 마모도에 대해서 먼저 기술한다. 플래시 메모리는 셀에 비트를 쓰고 지우기 위해서 강한 전압을 가하게 되는데 이 때 셀 안에 있는 절연체에 손상을 주게 된다. 그래서 절연체가 일정이상 손상되어 비트 에러가 나는 것을 방지하고자 각 블록 당 쓰기/지우기 횟수(P/E cycle : Program/Erase cycle)가 제한되어 있다. 그리고 NAND는 셀에 저장할 수 있는 비트의 개수에 따라 SLC(Single-Level Cell), MLC(Multi-Level Cell), TLC (Triple-Level Cell)로 구분하고 각각 셀에 1비트, 2비트, 3비트를 저장할 수 있다. 하지만 많은 비트를 저장할수록 하드웨어적으로 높은 정확도가 요구되기 때문에 가능한 P/E 횟수가 적어진다. 일반적으로 SLC, MLC, TLC는 각각 105, 104, 103번의 P/E 제한을 가지고 있다. 따라서 NAND 플래시 메모리가 대용량화 될수록 가능한 P/E의 횟수가 낮음으로 전체적으로 균등하게 P/E를 실현하는 것이 중요하다[5-7].

플래시 메모리의 각 블록을 균등하게 지우게 하여 전체 수명을 연장시키는 기법을 마모도 평준화라고 한다. 최근 플래시 메모리의 수명을 연장시키기 위해 여러 마모도 평준화 기법들이 연구되었다. 대부분의 연구들에서는 각 블록의 지우기 횟수, 참조 및 지우기 경과 시간, 각 페이지의 참조 횟수 등을 메모리에 테이블로 저장하여 마모도 평준화를 수행한다. 하지만 각종 정보가 저장되는 테이블의 메모리 소모가 많아 메모리가 부족한 임베디드 시스템이나 센서 노드에는 적용하기 힘들다.

이를 위해 각 블록의 지우기 수행 여부를 비트의 배열로 되어 있는 테이블에 저장하고 메모리의 사용을 최소화하여 마모도 평준화에 이를 이용하는 BET(Block Erase Table) 기법이 제안되었다[8]. BET는 'T' 값을 두어 마모도 평준화의 빈도수를 조절하며 'k'값을 두어 여러 개의 블록을 하나에 비트에 대응시켜 메모리의 사용량을 더욱 줄일 수 있다. 하지만 BET의 'k'가 증가됨에 따라 메모리의 사용량이 줄어들수록 마모도 평준화의 성능이 급격히 저하되는 단점이 있다. BET의 단점을 극복하고자 성능 저하의 원인을 숨겨진 콜드 블록 문제(hidden cold block problem)로 지적하고 무효 페이지(invalid page)의 개수를 기반으로 계산된 비트 설정 임계값(BST: Bit Set Threshold)을 이용한 기법이 제안되었다[9]. 하지만 BST는 무효 페이지 개수로 콜드(cold) 블록을 구별하기에는 한계가 있고 계산비용이 많아진다는 단점이 있다.

본 논문에서는 이를 해결하기 위해 XOR(eXclusive OR) 연산과 라운드 로빈 방식을 이용하는 SBET(Sampling-based BET)를 제안하다. SBET는 '*k*'값이 증가될 때 여러 개의 블록이 하나의 비트에 대응되어 콜드 블록의 구별이 어려워진다는 것에 착안하여 XOR 연산과 라운드 로빈을 이용해서 BET의 각 비트를 하나의 블록에 대응되도록 한다. 이하 본 논문의 구성은 다음과 같다. 2장에서 마모도 평준화 관련 연구과 숨겨진 콜드 블록 문제를 설명하고, 3장에서 제안하는 SBET 기법을 서술한다. 4장에서 실험환경을 구축하고 SBET를 평가한다. 그리고 마지막으로 5장에서 결론을 서술한다.

II. Related Works

1. Wear leveling techniques

플래시 메모리의 수명을 연장하기 위해 여러 마모도 평준화 기법들이 연구되었다. 마모도 평준화를 위해 여러 정보를 테이블에 저장하고 이를 이용한다. 마모도 평준화에 사용되는 정보를 큰 범주로 구분하면 지우기 횟수, 참조 횟수, 경과 시간 등으로 나눌 수 있다[7].

지우기 횟수를 이용한 대표적인 마모도 평준화 기법으로 DP(Dual-Pool) 기법과 지연 마모도 평준화 기법(lazy wear leveling)이 있다[10-11]. DP는 각 블록의 지우기 횟수에 따라 갱신 확률이 낮은 콜드 블록 그룹과 갱신 확률이 높은 핫(hot) 블록 그룹으로 나누고, 콜드 블록 그룹의 첫 번째 블록과 핫 블록 그룹의 마지막에 해당하는 블록의 지우기 횟수를 서로 비교하여 임계값 이상이면 교체한다. 지연 마모도 평준화 기법은 블록의 지우기 횟수가 임계값 이상이면 바로 삭제 하지 않고 추후 콜드 데이터를 해당 블록으로 옮긴다.

참조 횟수를 기반으로 하는 대표적인 마모도 평준화 기법으로 Rejuvenator와 OWL(Observational Wear Leveling)이 있다[12-13]. Rejuvenator는 참조 횟수를 각 페이지별로 저장하고 참조 횟수가 낮은 페이지는 콜드 페이지로 구분하고 참조 횟수가 높은 페이지는 핫 페이지로 구분한다. 콜드 페이지들은 지우기 횟수가 많은 블록에 저장하고 핫 페이지들은 지우기 횟수가 적은 블록에 저장한다. OWL은 참조된 블록의 주소를 리스트로 보관하고 주기적으로 관찰하여 자주 참조되는 블록들을 리스트에 보관하도록 유도한다. 그리고 자주 참조된 블록을 지우기 횟수가 적은 블록으로 이주한다.

경과 시간을 이용한 최근의 기법들로 FeGC(Fast and endurant Garbage Collection)와 NSAGC(Swap-Aware Garbage Collection for NAND)가 있다[14-15]. 이들은 무효 페이지가 발생한 후 경과된 시간을 이용하여 각 블록의 비용을 계산하고 이를 이용하여 마모도 평준화를 수행한다.

이상의 기법들은 마모도 평준화를 수행하기 위해 블록 지우

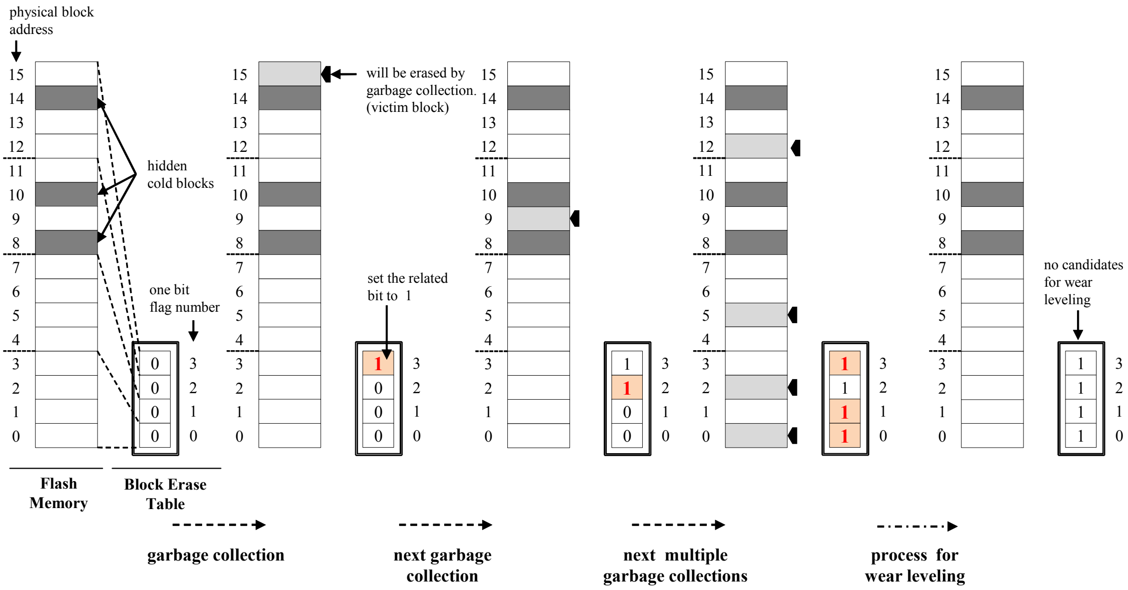


Fig. 1. Hidden cold block problem in BET technique. (k=2)

기 횟수 테이블, 참조 횟수 테이블 등과 같은 각종 정보들을 여러 테이블에 저장하고 사용하기 때문에 일정 이상의 메모리 사용량이 요구된다. 따라서 메모리 사용량에 제한이 있는 저사양의 임베디드 장치나 사물 인터넷(IoT)에서 사용되는 단말기에서는 적용하기 힘들다는 단점이 있다.

2. BET(Block Erase Table) and BST(Bit Set Threshold)

BET 기법은 각 블록의 지우기 수행 유무에 대한 이력을 비트로 저장하고 이를 이용한다[8]. 따라서 비트 배열 테이블로 되어 있는 BET를 사용하기 때문에 다른 기법들보다 메모리 사용량을 줄일 수 있다는 장점을 가지고 있다. BET의 각 비트는 하나의 블록 또는 여러 블록과 연관되어 있으며 관련 블록이 지워지면 해당 비트를 1로 설정한다. 일정 기간 후 BET에서 0인 비트에 관련된 블록들은 콜드 블록으로 인식하고 강제로 이주한다. 즉, 일정 기간 동안 한 번도 지우기 연산을 수행하지 않은 블록은 콜드 블록으로 간주하고 이주시켜 마모도 평준화를 수행한다.

BET 기법은 마모도 평준화 빈도를 조절하기 위해 ‘T’ 값과 메모리의 사용량 조절하기 위해 ‘k’ 값을 가지고 있다. 전체 블록의 지우기 횟수를 BET에 저장되어 있는 1비트의 개수로 나누어 구한 값과 ‘T’ 값을 비교하여 ‘T’ 값이 작게 되면 콜드 블록의 이주를 시작한다. 초기에 설정된 ‘T’ 값이 작을수록 마모도 평준화의 수행 빈도가 많아지게 된다. 또한 ‘k’ 값에 따라 2k 개의 블록들을 하나의 비트에 대응시키기 때문에 ‘k’ 값이 커짐에 따라 BET에 소요되는 메모리를 줄일 수 있다.

하지만 ‘k’ 값이 커짐에 따라 마모도 평준화의 성능은 급격히 떨어지는 단점이 있다. 이는 ‘숨겨진 콜드 블록 문제’가 원인으로 여러 개의 블록이 하나의 비트에 대응되면 콜드 블록의 정보가 일부 누락된다. 이를 해결하기 위해 무효 페이지의 개수를 통해 콜드 블록을 감지하고 그룹 내의 블록이 지워지더라도 1 비트로 설정되는 것을 방지하는 BST 기법이 제안되었다[9].

BST 기법은 희생 블록이 지워질 때 해당 그룹의 평균 무효 페이지 개수와 희생 블록의 무효 페이지 개수의 차이를 BST 값으로 설정하고, BST 값보다 무효 페이지 개수가 적은 블록이 존재하면 해당 블록을 콜드 블록으로 인식하고 1 비트로 설정하는 것을 방지한다. 하지만 무효 페이지의 개수로 콜드 블록을 구별하기에는 한계가 있다. 그 예로 무효 페이지의 개수가 적더라도 핫 블록일 수 있으며 무효 페이지의 개수가 많더라도 콜드 블록일 가능성이 있다. 따라서 BST 기법은 잘못된 콜드 블록의 정보로 인하여 콜드 블록 정보의 정확도를 향상시키고 숨겨진 콜드 블록 문제를 해결하는데 한계가 있다.

3. Hidden cold block problem

숨겨진 콜드 블록 문제는 그룹 내에 콜드 블록이 존재함에도 불구하고 다른 블록이 지워지면 콜드 블록의 정보를 알 수 없는 상황을 말한다. 최악의 경우 모든 그룹에 핫 블록과 콜드 블록이 존재한다면 자주 지워지는 핫 블록의 특성으로 인하여 모든 그룹의 콜드 블록을 인식하지 못하는 문제가 발생할 수 있다. 그러면 마모도 평준화는 수행이 되지 않아 플래시 메모리의 전체 수명은 짧아지게 된다.

그림 1은 숨겨진 콜드 블록 문제가 발생하는 예를 나타낸 것이다. 플래시 메모리의 블록이 16개이고 ‘k’ 값이 2일 때 4개의 블록이 하나의 비트에 대응되기 때문에 BET의 비트는 4개가 된다. 8번, 10번, 14번의 콜드 블록이 존재한다고 할 때 8번과 10번은 BET의 2번 비트에 대응되고 14번은 BET의 3번 비트에 대응된다. 가비지 컬렉션에 의해서 15번 블록이 희생 블록으로 선정되어 지워지면, 콜드 블록인 14번이 존재함에도 불구

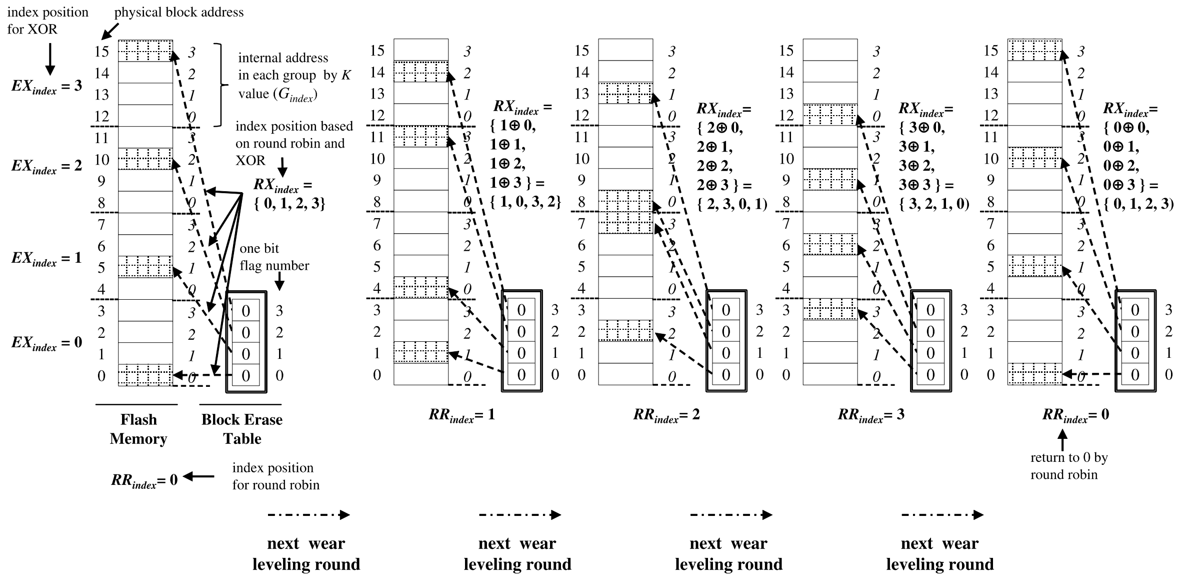


Fig. 2. Mapping method for one block by one bit using SBET.

하고 BET의 3번 비트는 1로 설정된다. 다음 가비지 컬렉션에서 9번이 선택되면 BET의 2번 비트가 1로 설정된다. 마찬가지로 다음의 여러 가비지 컬렉션에서 0번, 2번, 5번, 12번 블록이 희생블록으로 선택되면 BET의 0번, 1번, 3번 비트가 1로 설정된다. 결국 3개의 콜드 블록이 존재함에도 불구하고 BET의 모든 비트는 1로 설정되어 마모도 평준화가 수행되지 않는다. 이렇게 콜드 블록을 구별하지 못하여 각 블록의 지우기 횟수의 차이는 커지게 된다.

특히 갱신 빈도가 매우 낮은 데이터가 저장되어 있는 블록들은 물리적으로 연속되게 저장되는 경우가 많지 않다. FTL의 주소 매핑 기법으로 인하여 논리적인 주소가 연속적으로 저장되어 있다고 하여도 물리적인 주소는 항상 그렇지 않다. 따라서 같은 그룹 내에 콜드 블록과 핫 블록이 함께 존재할 가능성이 높다. 또한 콜드 블록이 연속적으로 저장되어 있다고 하여도 하나의 블록이 핫 블록으로 변경되면 나머지 콜드 블록들의 정보를 인식할 수 없게 된다. 결국 이런 상황과 운영체제 데이터와 같은 콜드 블록들은 대부분 존재하기 때문에 마모도 평준화의 성능이 저하될 가능성이 높아지게 된다.

III. SBET Wear Leveling Technique

1. Sampling-based BET

숨겨진 콜드 블록 문제를 해결하기 위해 SBET 기법은 'k'값이 증가하더라도 BET의 각 비트를 하나의 블록에만 대응시킨다. 그리고 기존의 BET 기법처럼 블록의 지우기 이력을 이용하여 마모도 평준화를 수행한다.

SBET는 두 개의 주요 변수를 사용한다. 첫 번째 변수는 XOR 기반 라운드 로빈 인덱스(RX_{index})이고 두 번째 변수는 각 블록의 그룹 내부 주소(G_{index})이다. RX_{index}는 라운드 로빈 인덱스인 RR_{index}와 XOR 인덱스인 EX_{index}를 기반으로 계산된다. RR_{index}는 각 마모도 평준화 수행이 끝날 때마다 식 (1)을 통해 계산된다.

$$RR_{index} = (RR_{index-1} + 1) \bmod 2^k, \quad 0 \le RR_{index} \le (2^k - 1) \tag{1}$$

기존의 BET 기법에서는 'k' 값이 1 이상이 되면 각 비트에 2개 이상의 블록들이 대응된다. 따라서 하나의 비트에 대응되는 블록들을 하나의 그룹이라고 할 때, RR_{index}는 BET의 각 비트에 하나의 블록이 대응되도록 각 그룹 내의 블록들 중에서 하나를 선택하게 된다. RR_{index}는 순차적인 방식을 사용하기 때문에 규칙적인 패턴을 가지고 있지만, 시스템에서 사용되는 각 응용들의 쓰기 패턴은 서로 다르다. 따라서 여러 응용들을 위해 불규칙적인 패턴을 가지도록 XOR 연산을 이용한다. 이를 위해 사용되는 EX_{index}는 식 (2)를 통해 계산된다. EX_{index}는 각 블록(b_{index})의 주소에 따라 값이 다르며 지우기 연산이 수행될 때만 계산되고 참조 된다.

$$EX_{index} = \left\lfloor \frac{b_{index}}{2^k} \right\rfloor \bmod 2^k, \quad 0 \le EX_{index} \le (2^k - 1) \tag{2}$$

RX_{index}는 RR_{index}와 EX_{index}를 기반으로 계산되어 각 그룹 내에서 하나의 비트에 대응되는 블록을 선택하게 한다. RX_{index}는 식 (3)을 통해 계산된다. RX_{index}는 EX_{index}와 마찬가지로 지우기 연산이 수행될 때만 계산되고 참조된다.

$$RX_{index} = RR_{index} \oplus EX_{index}, \quad 0 \le RX_{index} \le (2^k - 1) \tag{3}$$

각 블록의 G_{index}는 식 (4)를 통해 계산된다. G_{index}는

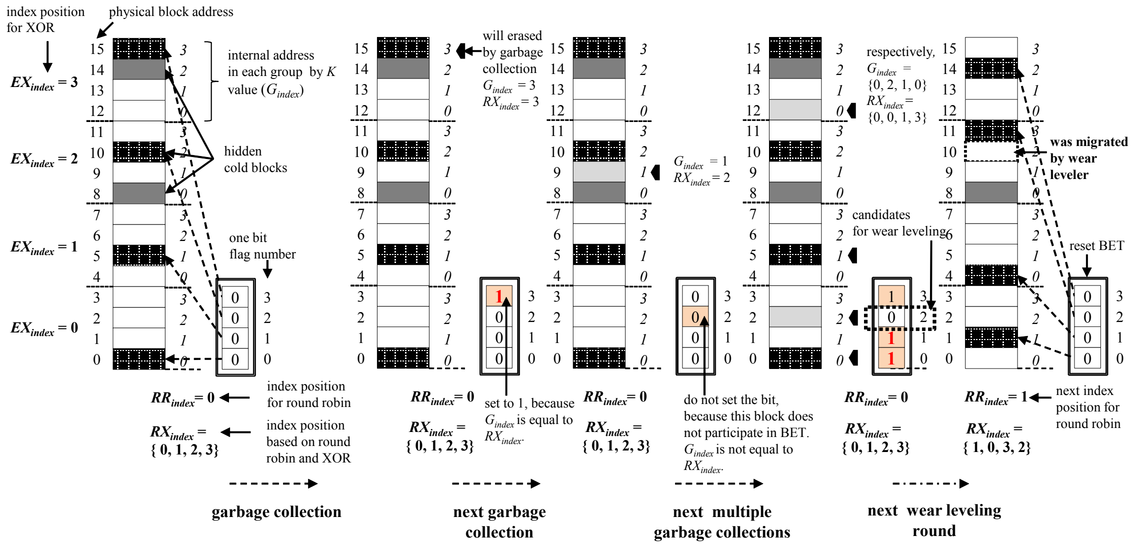


Fig. 3. An example of SBET to solve hidden cold problem.

EX_{index}와 동일하게 'K' 값에 따라 그 값의 범위가 한정된다. G_{index}는 가비지 컬렉션에 의해 희생 블록이 선택되어 지우기 연산이 수행될 때 계산된다.

$$G_{index} = b_{index} \bmod 2^k, \quad 0 \leq G_{index} \leq (2^k - 1) \tag{4}$$

그림 2는 XOR 기반 라운드 로빈 인덱스를 이용하여 BET의 각 비트에 하나의 블록을 대응시키는 예를 보여주고 있다. 그림 1과 동일하게 각 그룹은 4개의 블록으로 구성되어 있으며, BET의 비트는 4개로 구성되어 있다. 첫 번째 마모도 평준화 수행 기간에는 라운드 로빈 인덱스가 0으로 설정되어 있어, 각 그룹의 XOR 인덱스에 따라 얻어진 XOR 기반 라운드 로빈 인덱스에 관련된 블록들이 BET의 비트에 대응한다. 관련 없는 나머지 3개의 내부 주소들은 BET의 비트설정에 관여하지 않는다. 두 번째 마모도 평준화가 발생하면 식 (1)에 의해 라운드 로빈 인덱스는 1이 되고, XOR 기반 라운드 로빈 인덱스는 다시 식 (3)에 따라 계산되어 해당하는 각 그룹의 블록들이 BET의 비트에 대응한다. 이와 같은 방식으로 진행이 되면 라운드 로빈 인덱스는 0으로 되돌아오고, 반복적으로 그룹 내에서 각 블록들을 순서대로 BET의 비트에 대응시킨다.

따라서 XOR 기반 라운드 로빈 인덱스를 사용하면, 그룹 내에 관련 없는 블록들이 지워지더라도 BET의 비트를 설정하지 않기 때문에 일정 기간 후 BET의 비트를 이용해 콜드 블록을 구별할 수 있다.

2. Wear leveling operations

BET에서 'K' 값이 1 이상일 때에는 2개 이상의 블록이 하나의 비트에 대응하기 때문에, SBET는 그룹 내부 주소와 XOR 기반 라운드 로빈 인덱스를 이용해 하나의 비트에 하나의 블록을 대응시켜 숨겨진 콜드 블록 문제의 원인을 해결한다.

제안하는 알고리즘의 주요 동작은 다음과 같다. 전체 블록의 지우기 횟수(e_{cnt})를 BET의 1비트 개수(f_{cnt})로 나누어 'T'보다 크면 마모도 평준화의 수행을 시작한다. BET의 1비트를 검색하고 해당 비트의 번호(f_{index})를 *EraseBlockSet* 함수에 전달하여 관련 블록을 강제로 이주한다. 가비지 컬렉션 또는 마모도 평준화의 작업에 의해서 BET의 모든 비트가 1로 설정되면 BET를 초기화하고 라운드 로빈 인덱스의 값을 다시 계산한다. *EraseBlockSet* 함수는 이주할 블록을 계산하고 이주하는 단순 작업을 수행한다. 이주할 희생 블록($victim_{index}$)은 식 (5)과 같이 계산하여 얻을 수 있다.

$$victim_{index} = (f_{index} \times 2^k) + RR_{index} + f_{index} \bmod 2^k \tag{5}$$

가비지 컬렉션이나 마모도 평준화에 의해서 해당 블록(b_{index})의 유효 페이지가 이주되고 해당 블록이 삭제되었을 때, BET의 관련 비트를 1로 설정하는 알고리즘은 다음과 같다. 우선 XOR 기반 라운드 로빈 인덱스와 내부 주소로 변환한다. 그리고 그 값들을 서로 비교하여 상호 같다면 관련 비트를 1로 설정한다.

그림 3은 SBET를 이용하여 콜드 블록을 마모도 평준화에 참여시키는 사례이다. 전체 블록 개수, 'K' 값, 콜드 블록의 개수와 위치는 그림 1과 동일하다. 라운드 로빈 인덱스의 값이 0인 상태에서 가비지 컬렉션이 발생하여 희생 블록인 15번 블록이 선택되어 지워지면, 15번 블록은 내부 주소가 3이고 XOR 기반 라운드 로빈 인덱스의 값은 3이 된다. 따라서 두 변수의 값이 같기 때문에 BET의 3번 비트를 1로 설정한다. 두 번째 가비지 컬렉션에서 희생 블록으로 9번이 선택되면 내부 주소는 1이 되고 XOR 기반 라운드 로빈 인덱스 값은 2가 된다. 이 경우 두 변수의 값이 다르기 때문에 BET의 비트를 1로 설정하지 않고 기존의 값을 유지한다. 마찬가지로 0번, 2번, 5번, 12번 블록이 가비지 컬렉션의 희생 블록들로 선택되면 BET의 2번

비트를 제외한 나머지 비트는 1로 설정이 된다. 그 후 마모도 평준화 수행 조건이 되면, BET의 비트가 0인 번호는 2번이 된다. 따라서 라운드 로빈 인덱스가 0이고 BET의 2번 비트가 0일 때, 식 (5)에 의해 관련 블록은 10번이 되기 때문에 10번 블록이 이주된다. 이와 같은 방식으로 나머지 콜드 블록인 8번과 14번 블록은 다음 마모도 평준화 수행에서 이주되게 된다.

IV. Experiment and Evaluation

SBET 기법을 평가하기 위해 시뮬레이터를 구축하고, 이를 기존의 BET 기법과 BST 기법과 서로 비교하였다. 시뮬레이션 환경은 표 1과 같다.

Table 1. Details of the simulation environment

items	values and description
block count / page count	2048 / 128
page size / total capacity	4KB / 1024MB
each file size	222 page (888KB)
file count	1000
garbage collection trigger	the number of free blocks is under 102. (2%)
initial free space	15%

시뮬레이션에 적용한 각 파일들은 동일한 크기를 가진다고 가정하였고, 초기 가용 공간은 15%으로 설정하였다. 다양한 실험 환경을 위해서 3개의 상황으로 나누고 정규 분포를 적용하였다. 그림 4는 각 파일에 적용된 갱신 확률을 표시하고 있다. 파일들 중에서 700개만 갱신 확률을 적용하였고 나머지 300개는 갱신 확률을 0으로 고정하여 운영제제 데이터와 같은 콜드 데이터를 적용하였다. 그리고 각 파일을 무작위로 배치하여 콜드 블록과 핫 블록이 물리적인 위치에 연속적으로 저장되지 않고 불규칙적으로 저장될 수 있도록 하였다.

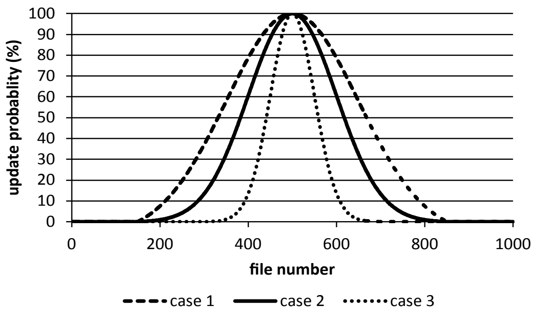


Fig. 4. Update probabilities of trace files for simulation

시뮬레이션에 주소 매핑 기법은 페이지 기반 매핑 기법(FTL)과 블록 기반 매핑 기법(NFTL: NAND FTL)을 적용하였다. NFTL의 MLC NAND 플래시 메모리를 고려하여 각 논리적인 블록 주소에 하나의 교체 블록만 사용하도록 하였다. FTL은 NFTL보다 매핑 테이블에 소요되는 메모리가 많아 임베디드 장치에는 NFTL이 많이 사용된다. 하지만 NFTL은 FTL보다 읽기 속도가 느리고 가비지 컬렉션의 수행에 필요한 시간비용이 많다는 단점이 있다[4].

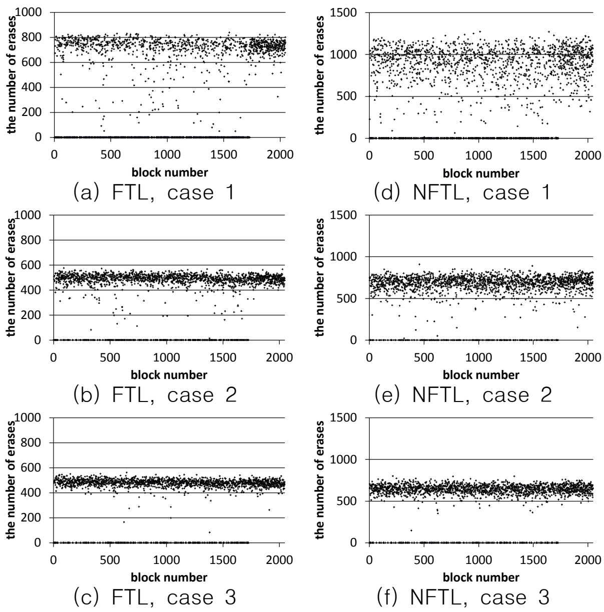
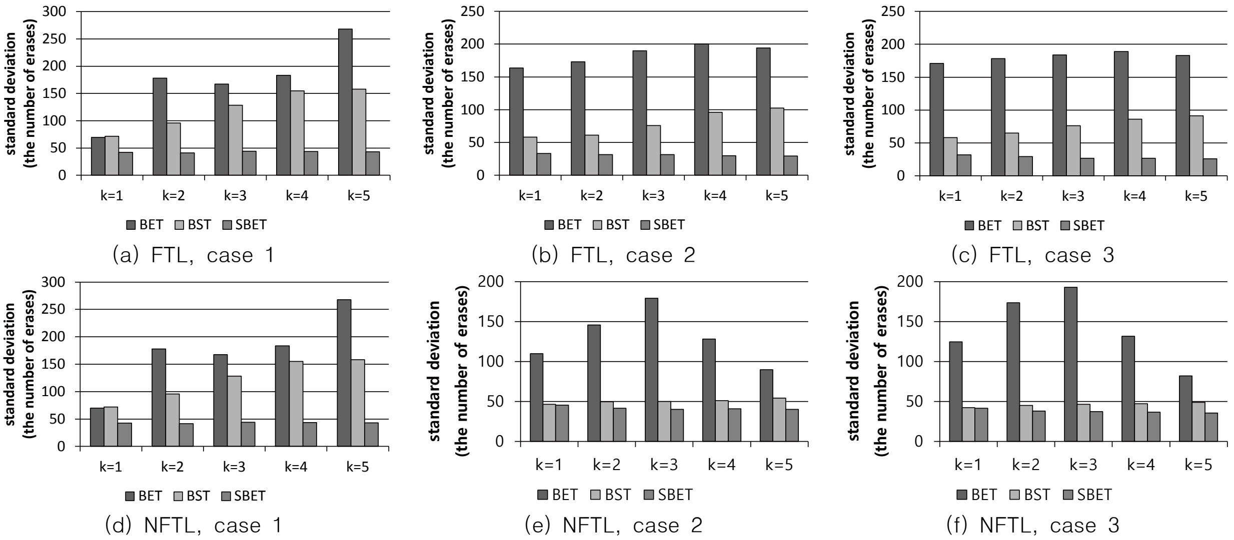


Fig. 5. Distribution of the number of block erases for each case, when GA is used in FTL and NFTL.

그림 5는 정적 마모도 평준화를 수행하지 않고 탐욕 알고리즘(GA)을 사용하는 가비지 컬렉션을 수행하였을 때, 각 상황별로 블록의 지우기 횟수 분포를 나타낸 것으로 쓰기를 1억 번 수행하였을 때의 결과이다. NFTL은 FTL보다 가비지 컬렉션에 소요되는 시간비용이 많기 때문에, NFTL의 평균 블록 지우기 횟수가 FTL보다 많다. 그리고 NFTL이 FTL보다 블록 지우기 횟수의 분산이 더 크다는 것을 알 수 있다. 각 하위 그림에서 블록의 지우기 횟수가 0인 블록들이 존재하며, 이들은 갱신 확률이 0이고 무효 페이지가 발생하지 않아 가비지 컬렉션의 대상이 되지 못한 블록들이다.

그림 6은 '*K*' 값이 1이상이고 '*T*' 값이 10일 때, 각 기법의 표준 편차를 보여주고 있다. 표준 편차는 1억 번 쓰기가 발생하였을 때 측정하였다. FTL에서 BST는 BET보다 표준 편차가 낮았지만 '*K*' 값이 증가함에 따라서 표준 편차가 증가하였다. 하지만 SBET는 FTL에서 '*K*' 값이 증가하여도 표준 편차의 큰 차이가 없다는 것을 알 수 있다. 따라서 SBET는 FTL에서 다른 기법과는 다르게 '*K*' 값이 증가하여도 성능이 균일하다는 것을 알 수 있다.

그림 6의 NFTL에서 BET는 '*K*' 값에 상관없이 불규칙적인 표준 편차를 보이고 있다. 이는 숨겨진 콜드 블록 문제 등의 원인으로 콜드 블록을 정확하게 인식하지 않아 불규칙적인 표준 편차의 값이 나온 것이다. (b)와 (c)를 보면, FTL에서도 '*K*' 값이 5일 때 BET의 표준 편차가 약간 감소하여 불규칙적인 형태를 보이고 있다. NFTL이 FTL보다 더욱 심하게 불규칙적인 것은 가비지 컬렉션을 수행할 때 NFTL의 특성상 논리적인 블록 주소에 추가적인 교체 블록이 연결되어 있기 때문에 FTL에서

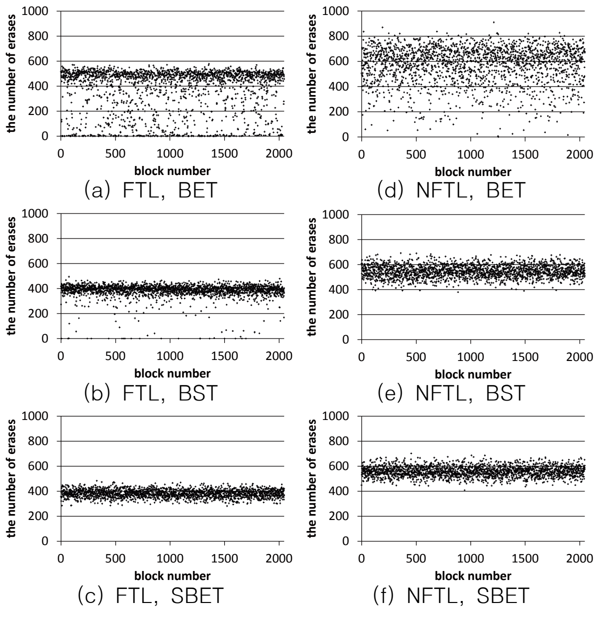


(a) FTL, case 1 (b) FTL, case 2 (c) FTL, case 3

(d) NFTL, case 1 (e) NFTL, case 2 (f) NFTL, case 3

Fig. 6. Standard deviation of the number of block erases for each wear leveling technique in FTL and NFTL. (T=10)

삭제되는 블록보다 NFTL에서 삭제되는 블록이 2배까지 많을 수 있다. 따라서 이런 특징은 'k' 값에 따라 불규칙적인 표준 편차를 더욱 가중시킨다. 하지만 주소 매핑 기법이 달라도 BST와 SBET는 상대적으로 표준 편차가 균일하거나 일정하게 증가한다는 것을 알 수 있다. BST는 NFTL에서 FTL과는 다르게 상대적으로 균일한 성능을 보였지만 SBET가 여전히 표준 편차가 낮다는 것을 알 수 있다. 결국 SBET는 FTL에서 BET보다 표준 편차가 최대 84% 감소하였고, BST보다 최대 73% 감소하였다. 그리고 NFTL에서의 SBET는 표준 편차가 BET보다 최대 81%, BST보다 최대 27% 감소하였다.



(a) FTL, BET (d) NFTL, BET

(b) FTL, BST (e) NFTL, BST

(c) FTL, SBET (f) NFTL, SBET

Fig. 7. Distribution of the number of block erases for each wear leveling technique in FTL and NFTL. (case 2, k=2)

그림 7은 상황 2에서 'k' 값이 2일 때 각 기법별 블록의 지우기 횟수 분포를 나타내고 있다. (a)에서 FTL에 적용된 BET는 숨겨진 콜드 블록 문제로 마모도 평준화에 참여하지 못한 블록들이 있어 지우기 횟수가 0인 블록들이 존재하였다. (b)에서 BST는 BET보다는 블록의 지우기 횟수가 고르게 분포하였지만 일부 블록들은 차이가 난다. (c)에서 SBET는 대부분의 콜드 블록들이 마모도 평준화에 참여하여 다른 기법보다 상대적으로 블록의 지우기 횟수가 고른 분포를 보였다. (d)에서 BET는 NFTL에서 FTL보다 지우기 횟수가 0인 블록들이 눈에 뛰게 적었지만 여전히 지우기 횟수의 분산이 컸다. (e)와 (f)에서 BST와 SBET는 블록의 지우기 횟수가 고르게 분포하였지만 SBET에서 블록의 지우기 횟수가 더 밀집된 형태라는 것을 알 수 있다. 따라서 BST와 SBET는 숨겨진 콜드 블록 문제를 해결하지만 SBET가 성능이 더 좋다는 것을 알 수 있다.

그림 8은 각 기법을 적용하였을 때 플래시 메모리의 수명을 나타낸 것이다. 플래시 메모리의 각 블록은 1000번의 지우기 횟수 제한이 있다고 가정하였고 실험 중에 임의의 블록이 1000번의 지우기 횟수가 발생하는 시점을 플래시 메모리의 수명으로 가정하였다. 그리고 각 수명은 진행된 쓰기 횟수를 환산하여 적용하였다. 각 그림에서는 실험 결과의 비교를 위해 추가적으로 정적 마모도 평준화가 수행되지 않은 GA의 결과를 검정색으로 표시하였고 BST와 SBET를 적용할 수 없는 'k' 값이 0인 결과를 역시 검정색으로 표시하였다. (a)~(c)에서 FTL이 적용되었을 때는 각 기법들이 'k' 값이 증가함에 따라 수명이 감소하였으나 SBET는 상대적으로 다른 기법보다 감소량이 적었다. 특히 'k' 값이 0일 때와 'k' 값이 1 이상일 때를 비교하였을 때, BET는 큰 차이를 보였지만 SBET는 비슷한 수명을 보였다. (d)~(f)에서 NFTL을 각 기법을 적용하였을 때, 'k' 값이 0일 경우와 'k' 값이 1 이상일 경우를 비교하면 BET는 FTL에서 적용된 상황과 비슷하게 많은 차이가 발생하였다. 하지만 BST와 SBET는 모두 근소한 차이를 보였다. 그리고 SBET가 BET보다 수명이 더 길다는 것을 알 수 있다. BET의 'k' 값이 0일 때

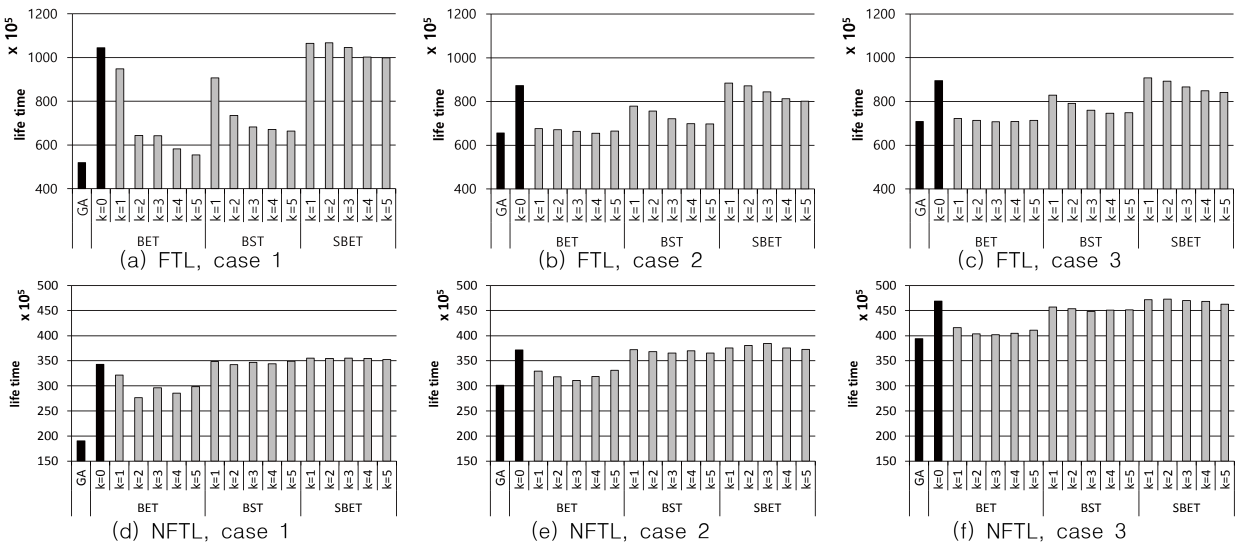


Fig. 8. Life time of flash memory for each wear leveling technique in FTL and NFTL. (*T*=10)

와 SBET의 '*K*' 값이 1 이상일 때를 비교하면, SBET가 BET와 대부분 수명이 비슷하고 (a), (d)와 같이 일부 상황에서는 BET의 '*K*' 값이 0인 것 보다 SBET가 오히려 수명이 더 긴 경우도 있었다. 결국 SBET는 '*K*' 값이 증가하여도 성능 저하가 적었으며, 상대적으로 다른 기법보다 균등한 성능 분포를 보였다. 결과적으로 SBET가 BET보다는 최대 80%, BST보다 최대 53% 더 수명을 연장하였다.

V. Conclusions

본 논문에서는 플래시 메모리의 수명을 연장하기 위해 SBET(Sampling-based Block Erase Table) 기법을 제안하였다. SBET는 기존의 BET 기법에 XOR 기반의 라운드 로빈 방식을 적용하여 메모리의 사용량을 줄일 때 발생하는 성능 저하를 최소화 하였다. 성능 저하의 원인을 숨겨진 콜드 블록 문제로 정의하고 이는 하나의 비트가 여러 블록들과 연관되어 발생한다고 지적하였다. 이를 해결하기 위해 각 블록의 지우기 이력을 저장하는 비트 배열 테이블의 각 비트가 각 블록에 일대일로 대응될 수 있도록 하였다. 실험에서 BET의 개선을 위해 연구된 BST 기법과 기존의 BET 기법을 구현하고 제안한 기법과 이들을 상호 비교하였다. 그 결과 SBET는 플래시 메모리의 수명을 BET보다 최대 80% 연장하였고 BST보다 최대 53% 연장하였다. 그리고 메모리의 사용량을 줄여도 다른 기법보다 상대적으로 균등한 수명 연장 효과를 보여 메모리의 제한이 많은 센서 노드나 사물 인터넷 단말기에도 적용할 수 있음을 확인하였다.

SBET 기법은 최대 사용할 수 있는 메모리 크기가 정해져 있어 메모리 자원에 대한 제한이 적은 장치에서는 최대 사용 메모리 크기를 늘려 성능을 더욱 향상시킬 수 있는 방법이 필요하다. 따라서 이를 해결하는 방법을 향후 연구과제로 남긴다.

REFERENCES

[1] Lawton, George. "Improved flash memory grows in popularity." Computer, IEEE, Vol. 39, No. 1, pp. 16-18, Jan. 2006.

[2] Kwon, Se Jin, et al. "FTL algorithms for NAND-type flash memories." Design Automation for Embedded Systems, Vol. 15. No. 3-4, pp. 191-224, Dec. 2011.

[3] Chung, Tae-Sun, et al. "A survey of flash translation layer." Journal of Systems Architecture, Vol. 55, No. 5, pp. 332-343, May. 2009.

[4] Ma, Dongzhe, Jianhua Feng, and Guoliang Li. "A survey of address translation technologies for flash memories." ACM Computing Surveys (CSUR), Vol. 46, No. 3, pp. 1-39, Jan. 2014.

[5] Lee, Sungjin, and Jihong Kim. "Improving Performance and Capacity of Flash Storage Devices by Exploiting Heterogeneity of MLC Flash Memory." Computers, IEEE Transactions, Vol. 63, No. 10, pp. 2445-2458, Oct. 2014.

[6] Park, Byoungjun, et al. "Challenges and limitations of NAND flash memory devices based on floating gates." IEEE International Symposium on Circuits and Systems, pp. 420-423, May. 2012.

[7] Yang, Ming-Chang, et al. "Garbage Collection and Wear Leveling for Flash Memory: Past and Future.", Smart Computing (SMARTCOMP), 2014 International Conference on. pp. 66-73, Nov. 2014.

[8] Chang, Yuan-Hao, Jen-Wei Hsieh, and Tei-Wei Kuo. "Improving flash wear-leveling by proactively moving static data." Computers, IEEE Transactions, Vol. 59, No. 1, pp. 53-65, Jan. 2010.
[9] Kim, Seon Hwan, Jong Wook Kwak, and Park Chang-Hyeon "Wear Leveling Technique using Bit Array and Bit Set Threshold for Flash Memory," Journal of KSCI, Vol. 20, No. 11, pp. 1-8, Nov. 2015. (in Korean)
[10] Chang, Li-Pin. "On efficient wear leveling for large-scale flash-memory storage systems." Proceedings of the 2007 ACM symposium on Applied computing. ACM, pp. 1126-1130, Mar. 2007.
[11] Chang, Li-Pin, and Li-Chun Huang. "A low-cost wear-leveling algorithm for block-mapping solid-state disks." ACM SIGPLAN Notices, Vol. 46. No. 5. ACM, pp. 31-40, Apr. 2011.
[12] Murugan, Muthukumar, and David HC Du. "Rejuvenator: A static wear leveling algorithm for NAND flash memory with minimized overhead." Mass Storage Systems and Technologies, 2011 IEEE 27th Symposium on. IEEE, pp. 1-12, May. 2011.
[13] Wang, Chundong, and Weng-Fai Wong. "Observational wear leveling: an efficient algorithm for flash memory management." Design Automation Conference (DAC), pp. 235-242, Jun. 2012.
[14] Kwon, Ohhoon, et al. "FeGC: An efficient garbage collection scheme for flash memory based storage systems." Jou₩rnal of Systems and Software, Vol. 84, No. 9, pp. 1507-1523, Sep. 2011.
[15] Xu, Guangxia, Manman Wang, and Yanbing Liu. "Swap-aware garbage collection algorithm for NAND flash-based consumer electronics." Consumer Electronics, IEEE Transactions, Vol. 60, No. 1, pp. 60-65, Feb. 2014.

Authors

Seon Hwan Kim received the B.S., M.S., and Ph.D. degrees in Computer Engineering from Yeungnam University, Korea, in 2006, 2009, and 2016, respectively.
He is interested in NAND flash memory system, embedded system, and wireless sensor network.

Jong Wook Kwak received the B.S. degree in computer engineering from Kyungpook National University, Daegu, South Korea, in 1998, and the M.S. degree in computer engineering and the Ph.D. degree in electrical engineering and computer science from Seoul National University, Seoul, South Korea, in 2001 and 2006, respectively. From 2006 to 2007, he was a Senior Engineer with the system-on-chip (SoC) Research and Development Center, Samsung Electronics Company, Ltd., Suwon, South Korea. During 2011-2012, he was a Guest Researcher at the Research Institute of Advanced Computer Technology, Seoul National University. During 2012-2013, he was a Visiting Scholar at the Georgia Institute of Technology, Atlanta, GA, USA. He is currently an Associate Professor with the Department of Computer Engineering, Yeungnam University, Gyeongsan, South Korea. His current research interests include advanced processor architecture, low-power mobile embedded system design, and high-performance parallel and distributed computing.