Journal of Digital Contents Society Vol. 13 No. 1 Mar. 2012(pp. 75-81)

플래시 SSD의 내구성을 검증하기 위한 FTL 오프라인 알고리즘

정호영,* 이태화,** 차재혁,***

요 약

SSD (Solid-State Drives)는 고성능, 저전력, 내구성과 경량 등의 특징을 가지고 있어 빠른 속도로 하드 디스크를 대체하고 있다. SSD는 하드디스크와 같은 블록 저장장치로 에뮬레이트하는 계층인 FTL (Flash Translation Layer) 을 가지고 있다. 가비지 컬렉션(Garbage Collection)은 FTL의 주요한 기능으로서 SSD의 수명과 성능에 큰 영향을 끼친다. 그러나 아직까지 새로운 알고리즘을 검증하기 위한 사실상의 표준이 없는 상황이다. 본 논문에서는 이 문제를 해결하기 위해 트레이스 기반의 오프라인 최적 알고리즘을 제안한다. 제안한 알고리즘은 언제나 최소 횟수의 지우기 연산을 보장한다. 추가적으로 본 논문에서는 TPC 트레이스를 사용하여 제안한 알고리즘의 유효성에 대해 검증하였다.

An Offline FTL Algorithm to Verify the Endurance of Flash SSD

Hoyoung Jung*, Taehwa Lee**, Jaehyuk Cha***

Abstract

SSDs(Solid State Drives) have many attractive features such as high performance, low power consumption, shock resistance, and low weight, so they replace HDDs to a certain extent. An SSD has FTL(Flash Translation Layer) which emulate block storage devices like HDDs. A garbage collection, one of major functions of FTL, effects highly on the performance and the lifetime of SSDs. However, there is no de facto standard for new garbage collection algorithms. To solve this problem, we propose trace driven offline optimal algorithms for garbage collection of FTL. The proposed algorithm always guarantees minimal number of erase operation. In addition, we verify our proposed algorithm using TPC trace.

Keywords : Flash Memory, FTL, Garbage Collection, Offline Algorithm

1. 서론

최근 낸드 플래시 메모리(NAND Flash Memory) 기반 SSD(Solid State Drives)는 기존의 저장 장치인 하드디스크를 대체하는 저장 장치로 주목받고 있다[1]. 이는 SSD가 갖고 있는 빠른 속도, 저전력, 휴대성, 내구성과 같은 다양한 장점들 때문이다. 하지만 SSD의 내부 저장 소자인 플래시 메모리는 덮어쓰기가 불가능하고 블록당 지우기 횟수에 한계를 가지고 있다. 또한 플래시 메모리의 읽기/ 쓰기 단위는 512bytes에서 4KB 크기인 페이지 단위로 이루어지지만 지우기는 일정한 개수의 페이지들의 집합인 블록 단위로 이루어진다. 플래시 메모리의 물리적인 특성을 보완하고 하드 디스크와 같은 블록 저장장치로 사용하기 위해 SSD는 FTL(Flash Translation Layer)이라는 시스템 소프트웨어 모듈을 갖고 있다. FTL의 주요 기능은 논리 블록(logical block)과 물리 페이지(physical page)로의 사상(mapping), 가비지 컬렉션, 배드 블록 관리 등이며 FTL의 주요 기능들은 SSD의 성능에 큰 영향을 끼치게 된다[2].

※ 제일저자(First Author) : 정호영
접수일:2012년 01월 17일, 수정일:2012년 03월 15일
완료일:2012년 03월 23일
* LG전자 선임연구원
hoyoung@hanyang.ac.kr
** 한양대학교 석사과정
*** 한양대학교 정보통신학과 교수 (교신저자)
▣본 연구는 한국연구재단 정부지원연구 (20110004993)지원으로 수행되었음.

SSD의 성능을 향상시키기 위해 기존에 많은 연구들이 이루어졌으며 특히 가비지 컬렉션(garbage collection) 에 대해서도 활발한 연구가 이루어지고 있다. 가비지 컬렉션은 SSD에서 쓰기 요청을 처리할 수 있는 내부 저장 공간이 부족할 때 이루어지는 일련의 작업으로서, 플래시 메모리의 블록에 대한 지우기 연산 및 블록 내의 유효한 페이지들의 복사 과정을 수반한다. 가비지 컬렉션 알고리즘과 같은 FTL의 처리 알고리즘들은 SSD의 지우기 발생 횟수, 쓰기 증폭률(write amplification factor), 마모 평준화(wear leveling) 등에 큰 영향을 끼치는 중요한 알고리즘이다. 그러나 기존의 많은 연구에도 불구하고 알고리즘의 성능을 평가하기 위한 절대적인 기준의 부재로 인해 기존의 연구들은 상대적인 평가에 의해 알고리즘의 성능을 평가해 왔다[3][4][5][6][7].

이러한 문제를 해결하기 위해 본 논문에서는 최소 횟수의 지우기 발생을 보장하는 FTL의 오프라인 알고리즘을 제안한다. 제안한 알고리즘은 새로운 온라인 알고리즘에 대해 기존의 상대평가에서 벗어나 성능향상의 절대적인 지표를 제공해 줄 수 있을 것이다.

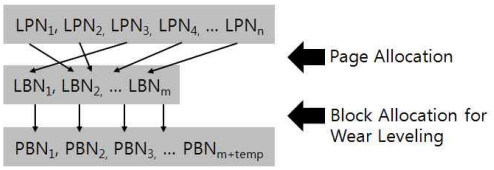
본 논문의 구성은 다음과 같다. 2장에서는 플래시 메모리의 FTL의 기본 개념과 용어를 설명하고, 시뮬레이션에서의 성능 평가 척도를 설명한다. 3장에서는 지우기 횟수를 최소화 할 수 있는 오프라인 최적화 알고리즘에 대해 자세히 설명한다. 4장에서는 3장에서 소개한 알고리즘의 문제점을 제시하고 이를 극복하기 위한 마모 평준화에 대한 개선안을 추가적으로 소개한다. 5장에서는 TPC 트레이스 기반의 시뮬레이션 실험을 통해 제안한 알고리즘이 수학적인 최적의 값과 동일함을 증명하였다. 마지막으로 6장에서 최종적인 결론을 맺고 향후 연구 과제를 설명한다.

2. 관련연구

2.1 FTL (Flash Translation Layer)

이 장에서는 알고리즘의 이해를 위해 필요한 FTL 메커니즘을 설명한다. FTL은 앞서 언급한 바와 같이 파일시스템으로부터 받은 논리 주소를 물리 주소로 변환하는 사상 기능을 수행한다. 파일시스템에서 논리 주소에 대한 읽기 요청이 들어오게 되면 FTL은 사상 테이블을 통해 논리 주소에 해당하는 물리 주소를 찾아서 읽기연산을 수행하게 된다. 플래시 메모리에서 덮어쓰기의 경우 데이터를 직접 덮어쓸 수 없기 때문에 변경된 데이터를 저장한 새로운 공간을 확보하여 변경된 데이터를 새로운 공간에 저장하고 기존의 데이터는 유효하지 않음을 표시한다. 새로운 쓰기 요청에 대해 더 이상 내부적인 저장 공간이 없게 되면 블록 지우기 연산이 발생하며, 이때 블록 내에 존재하는 유효한 페이지들은 새로운 페이지에 복사한 후 블록을 지워야 하기 때문에 추가 쓰기가 발생된다.

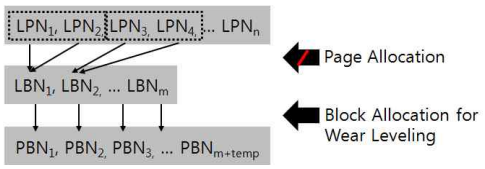
FTL의 사상하는 방법에는 크게 페이지 매핑(page mapping)과 블록 매핑(block mapping)의 두 가지 방법이 있다[8][9][10]. 페이지 매핑의 경우 사상테이블의 엔트리가 실제 페이지 개수만큼 존재하여 각 페이지를 1:1로 사상하게 된다. 페이지 매핑은 iops 등의 성능 척도에 대해 높은 성능을 보이지만, 매핑 테이블을 위한 저장 공간으로 인한 높은 비용이 단점이 되고 있다. 반면 블록 매핑의 경우 블록 단위로 사상을 하기 때문에 매핑 테이블의 저장 공간이 페이지 매핑에 비해 매우 적게 필요한 반면 주소 연산에 따른 성능 저하를 단점으로 가지고 있어 최근의 고성능 SSD의 경우 대부분 페이지 매핑에 기반을 둔 사상 알고리즘을 사용하는 경향이 있다[11][12].



(그림 1) 페이지 매핑의 기본 구조

페이지 매핑의 경우 (그림 1)과 같은 구조를 가지므로 논리 페이지에서 물리 페이지로의 할당이 자유롭고 마모 평준화를 위한 블록 할당도 직접 제어할 수 있다. 플래시 메모리 각 블록은 제한된 지우기 회수를 가지고 있으며, 따라서 SSD의 지우기 작업은 마모 평준화 알고리즘을 통해 전체 플래시 메모리 블록들에 대해 고르게 분산시켜야 한다. 페이지 매핑은 페이지의 블록

간 할당이 자유로우므로 지우기 횟수를 최적화 하기 위해 페이지 할당을 블록 매핑보다 용이하게 적용할 수 있다.



(그림 2) 블록 매핑의 기본 구조

블록 매핑의 경우에는 인접한 논리 페이지 번호(LPN: logical page number) 가 그룹으로 묶여서 물리 블록에 할당되기 때문에 페이지 할당이 자유롭지 못하다. 그렇기 때문에 본 논문에서 제시하는 알고리즘 중 3장의 지우기 횟수 최적화 알고리즘은 페이지 매핑에만 적용 가능하며 4장에서 제시하는 마모 평준화 알고리즘은 두 매핑 방식 모두에 적용 가능한 알고리즘이다.

2.2 가비지 컬렉션 및 마모평준화 정책

[3][4][5][6] 과 같은 기존의 연구들은 다양한 알고리즘과 정책 등을 사용해서 플래시 메모리의 성능과 수명을 높이려 해 왔다.

[3] 논문은 FTL의 메타데이터 저장영역을 활용하여 소거횟수가 낮은 블록에 소거를 집중시키는 방식을 사용하여, 간단한 알고리즘으로 지우기 연산의 마모평준화를 개선하였다. [4] 논문은 기존의 알고리즘들의 장점을 취합한 새로운 가비지 컬렉션 알고리즘을 제시하였다.

이들 논문은 각각 독창적인 아이디어를 제시하고 해당 아이디어를 구현 및 실험을 통해 성능을 평가하고 있다. 그러나 이들 연구는 대부분 성능을 평가하기 위해 제시한 알고리즘을 적용/미적용 시의 성능을 평가하거나 타 알고리즘 대비 개선도를 백분율로 나타내는 방법을 사용하여 성능을 평가하고 있다. 이것은 페이지 교체 정책의 MIN 알고리즘과 같이 최적의 성능을 알 수 있는 기준이 되는 오프라인 알고리즘이 없기 때문이다. 따라서 본 논문에서는 기준이 될 수 있는 최적의 오프라인 알고리즘을 제안하였다.

2.3 성능 평가 척도

일반적으로 하드디스크와 같은 디스크의 성능 실험에서는 IO 대역폭(IO bandwidth) 초당 IO 회수(IOPS: IO per seconds) 등이 중요한 성능 평가 척도로 활용되고 있다. SSD의 경우에 앞서 말한 두 가지 또한 중요하지만 수명을 고려하는 척도 또한 중요하다[13]. SSD의 각 블록은 지우기 횟수가 정해져 있어서 제한된 지우기 횟수에 도달하게 되면 배드 블록이 되며 이 경우 더 이상 SSD는 신뢰할 수 있는 저장장치로 활용할 수 없다. 따라서 SSD의 수명은 SSD의 성능을 평가하는 주요한 요소이며 이를 위한 평가 척도로는 지우기 횟수와 지우기 횟수의 블록별 분포가 있다. 같은 횟수의 쓰기 요청이라도 FTL의 내부 알고리즘에 따라 이들 지우기 횟수 및 분포는 확연이 달라진다. 본 논문에서도 지우기 횟수와 분포를 성능 평가 척도로 활용한다.

3. 오프라인 최적 알고리즘

이 장에서는 트레이스에 기반을 둔 오프라인 최적 알고리즘을 제시하고 설명한다. 제시하는 알고리즘은 트레이스 전체를 읽고 이를 분석하여 쓰기 요청에 의해 페이지들이 무효화(invalid)되는 순서를 파악하여 무효화되는 순서대로 페이지들을 그룹화하고 이들 페이지 그룹을 같은 물리 블록에 할당하는 것이다. 이렇게 모아진 블록은 이후에 블록을 삭제할 경우 블록 안에 유효한 페이지가 전혀 없으므로 추가적인 쓰기 발생이 일어나지 않고, 이에 따라 최소의 지우기 횟수를 보장하게 된다.

3.1 데이터 정의

본 절에서는 알고리즘에 사용되는 데이터 타입 및 함수를 정의한다. 제안하는 알고리즘에 사용되는 트레이스는 <요청순번, 논리페이지번호>의 형식으로 구성되는 요소들의 집합으로 구성되어 있으며, 본 절에서 알고리즘은 이를 분석하고 처리하여 실제 플래시 메모리에 사상시키는 알고리즘이다. 필요한 자료구조는 아래와 같다.

x= 요청순번, lpn: 논리페이지 번호
w= 트레이스 기본 요소 (x, lpn)으로 구성
T = 트레이스 w의 전체 집합
gnum= 무효그룹들을 구별하기 위한 그룹번호를 저장하기 위한 변수
getRequest(T)= T에서 한 요소를 제거하고 이를 반환
lpn(w)= w 요소에 해당하는 lpn 값을 반환
seq(w)= w 요소의 순번 x를 반환
$Group\_LPN_i$= {x|x∈ T and LPN(x) = i }
Group_LPN
={$Group\_LPN_0$,$Group\_LPN_1$, …
$Group\_LPN_{MAX\_LPN}$}
GVP= 유효 페이지들의 그룹
GIP_i= gnum을 이용해 분리된 무효페이지들의 그룹
GIP= { GIP_0, GIP_1, GIP_2 … GIP_{MAX_GNUM} }
GUP= GVP, GIP로 분류되기 전 Undefined 그룹

3.2 데이터 초기화

알고리즘에 사용되는 데이터들은 아래와 같이 초기화된다.

Group_LPN=
{$Group\_LPN_0$={},…$Group\_LPN_{MAX\_LPN}$={}}
gnum = 1

3.3 알고리즘

Step 1. 요청의 논리 페이지 번호별 그룹화

while((w = getRequest(T)) != NULL)
  $Group_LPN_{LPN(w)}$에 seq(w)를 추가한다.

Step 1의 과정에 의해 Group_LPN은 같은 논리 번호의 요청 순번들을 모아서 이를 그룹화한다. 플래시 메모리의 특성상 유효 페이지가 무효 페이지로 변경되기 위해서는 같은 논리 페이지 번호에 대한 재 쓰기가 이루어져야 한다. Step 1 의 과정이 없을 경우 Step 2의 알고리즘에서는 한 논리 페이지 번호에 대한 무효화 시점을 확인하기 위해 매번 전체 트레이스를 스캔해야 한다. 각 트레이스 요청마다 전체 트레이스를 스캔해야 하기 때문에 이 경우 Step 2의 알고리즘 복잡도는 $O(n^2)$이 된다.

그러나 Step 1의 분류 작업을 통해 각 논리 페이지 번호별로 요청을 묶어서 그룹화하면 LPN의 그룹만을 확인하면 되므로 알고리즘의 복잡도는 O(n)이 된다.

Step 2. 요청의 그룹화

Step 1의 결과를 활용해서 각 요청을 유효 페이지 그룹(GVP)과 무효페이지 그룹들(GIP_i)로 다시 그룹화하게 되는데 이 때 요청을 4가지 경우로 나누어 처리한다. 나누어지는 기준은 다음과 같다.

1. T에 해당 논리 페이지 번호가 유일하게 존재하는 경우: 해당 페이지는 무효화되지 않기 때문에 바로 GVP에 추가된다.

2. w가 해당하는 LPN 그룹의 마지막 요청인 경우: 이 경우 w 자신은 유효 페이지가 되고 같은 그룹의 이전 요청은 무효 페이지가 된다.

3. w가 해당하는 LPN 그룹의 처음 요소인 경우: 이 경우는 해당 요소는 유효/ 무효를 구분할 수 없으므로 GUP에 추가된다.

4. 1,2,3에 해당하지 않는 경우: 해당하는 LPN 그룹의 이전 요소는 무효페이지가 되고, 자신은 GUP가 된다. Step 2를 수행하면 모든 요청들이 어떤 그룹에 속하게 되는지 알 수 있게 된다.

```
bszie: 물리 블록에 저장할 수 있는 페이지의 최대 크기
while( (w = getRequest(T)) != NULL ){
if (size(Group_LPN_lpn(w) ==1)
  GVP에 w를 추가
  continue;
if( Group_LPN_lpn(w)의 첫번째값==seq(w))
  GUP에 w를 추가
  continue;
GUP에서 lpn(w)인 요소 삭제
GIP_gnum에서 lpn(w)인 요소 추가
if size(GIP_gnum) == bsize
  gnum++
if (Group_LPN_lpn(w)의 마지막값== req(w))
  GVP에 w를 추가
else
  GUP에 w 추가
}
```

Step 3. 페이지 할당

Step 2의 결과로 GIP[gnum] 그룹들과 GVP 그룹이 생성된다. Step 3은 실제 요청들을 물리 블록에 사상하는 단계로 GVP 그룹과 GIP_1그룹들을 각각 같은 물리 블록에 할당한다. 그렇게 되면 각 GIP 그룹의 페이지들은 같은 블록에 할당되어 있고 따라서 삭제 시에 모두 동시에 무효페이지가 되기 때문에 유효 페이지에 대한 추가 쓰기 작업이 전혀 발생하지 않게 된다.

```
while( (w = getRequest(T)) != NULL ) {
  if (w in GIP[g])
    if ( GIP[g] is in mapping table)
     b = block num of GIP[g]
    else b = 비어 있는 블록 번호 할당
  else  //w in VGP
    b =Valid 블록 번호 반환
  b 블록에 w에 해당하는 쓰기 연산 수행
}
```

4. 마모 평준화 개선 알고리즘

3장에서 설명된 알고리즘을 적용하면 총 지우기 횟수가 최소가 된다. 하지만 SSD의 경우 특정 블록에 지우기가 집중되면 수명이 짧아지기 때문에 이를 균등하게 배분할 필요가 있다.

3장의 알고리즘을 진행하게 되면 총 지우기 횟수는 수학적인 최소값이 되지만 분산값, 즉 각 블록의 지우기 횟수의 분포는 좋지 않다. 이 원인은 같은 논리 번호에 대한 쓰기 요청 간격이 긴 페이지들과 유효 페이지 그룹들이 지우기 횟수가 낮은 블록에 할당되어 긴 시간 동안 이를 차지하고 있기 때문이다. 실제 시험에 사용한 <표 1>의 데이터를 분석한 결과 일반적인 페이지들보다 요청 간격이 약 10여배 이상 긴 페이지들과 한번 쓰기를 요청한 후 이후에 재쓰기를 요청하지 않는 페이지들이 상당수 존재하는 것으로 판명되었으며 이러한 페이지들은 결국 마모 평준화에 악영향을 미치고 있다. 이들의 영향을 감소시키고 총 지우기 횟수에 영향을 끼치지 않고 마모 블록당 지우기 값을 균등하게 배분하기 위해 추가적으로 마모 균등화 개선 알고리즘을 제안하였다.

각 그룹의 요청번호를 분석하면 해당 페이지가 삭제되는 지점과 현재 들어온 지점의 차이를 알아 낼 수 있다. 제안하는 알고리즘은 이를 이용하여 유효페이지그룹의 페이지와 앞서 설명한 수명이 긴 페이지들을 지우기 횟수가 높은 블록에 할당하여 빨리 지워지는 페이지들이 지우기 횟수가 적은 블록에 할당될 수 있도록 하는 알고리즘이다.

```
Step 3.에서 빈 블록을 찾아 매핑 후 반환하는
과정에서 반환할 블록을 상황에 맞게 다른 블
록을 반환한다.

if Valid 그룹의 페이지라면
    지우기 횟수가 가장 높은 블록을 반환한다.
else if 기대되는 그룹번호와 페이지 그룹번호
의 차이가 기준치 이상이라면
    지우기 횟수가 가장 높은 블록을 반환한다.
else
    지우기 횟수가 가장 적은 블록을 반환한다.
```

5. 실험 및 성능 평가

이 장에서는 앞서 3,4장에 걸쳐 소개된 알고리즘을 구현하여 트레이스로 실험한 결과를 나타낸다.

5.1 실험 환경

실험에 사용된 시스템은 인텔 i7 860, 16GB 메모리의 가상 머신 환경에서 Ubuntu 10.04를 운영체제로 사용한다. 실험에 사용한 트레이스는 다음과 같다. 실험에 사용한 트레이스의 전체 크기가 너무 커서 본 실험에서는 이를 축소하여 사용하였다. 실험에 사용한 트레이스는 실제 트레이스의 1/100 크기로 축소하여 사용하였다.

<표 1> TPC 트레이스 특성

속성	값
이름	TPC
Write 수	11,648,888
최소 오프셋	8
최대 오프셋	32,696,064
최소 크기	32
최대 크기	4,096

5.2 실험 및 평가

<표 2> 실험 결과

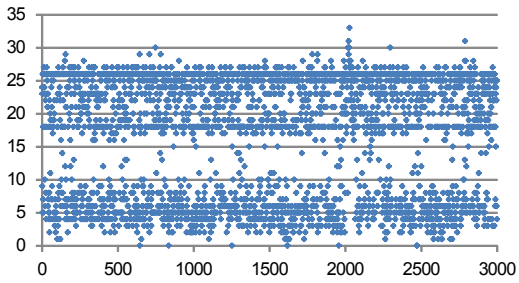
속성	값
(1) Invalid group	150,832
(2) Valid group	5,095
(3) Garbage Collection	176,919
(4) 최소 지우기 횟수: (3) +(2)	182,014

위 결과에서 Garbage Collection 호출횟수가 곧 지우기 횟수를 나타낸다. 여기서 나온 총 지우기 횟수는 176,919이다. 그러나 발생한 지우기 횟수 뿐만 아니라 유효 그룹들도 이후에 지우기를 발생시키므로 <표 2> 의 지우기 횟수에서 유효 그룹 횟수를 더한 값(<표 2> 의 (4) 항)이 최소 지우기 횟수가 된다. 이 수치를 검증하기 위해서는 다음과 같은 공식에 의한 값과 비교를 해보면 알 수 있다.

$$Erase\ Count_{\min} = \lceil \frac{Write\ Count}{\#\ of\ pages\ per\ block} \rceil$$

이 값을 구해보게 되면 11,648,888 / 64가 되고 이를 올림하면 182, 014가 되어 이는 알고리즘 수행 결과와 동일한 값이 된다. 총 지우기 횟수의 경우 앞에 설명과 같이 3장, 4장의 알고리즘 모두 같은 결과를 갖으며 따라서 두 알고리즘 모두 지우기의 최소 횟수를 보장하는 것을 알 수 있다.

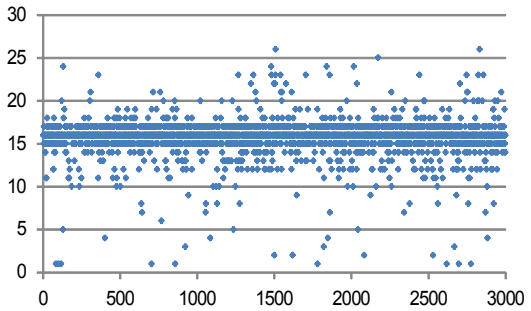
4장의 알고리즘을 적용할 경우 분포도의 경우 확실히 개선된 모습을 보여준다. 3장 알고리즘의 분포도는 다음 그래프와 같다.



(그림 3) 3장 알고리즘의 블록별 지우기

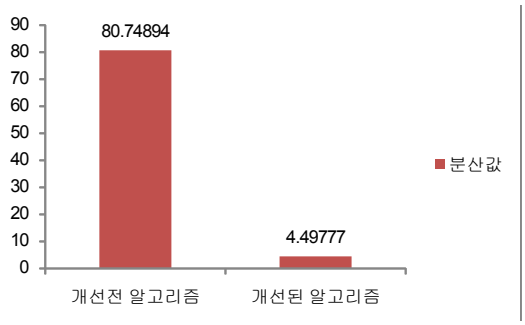
(그림 3)과 (그림 4)의 그래프의 x축은 각 블록번호이며 y축은 각 블록 발생한 지우기 연산의 누적된 횟수를 나타낸다. (그림 3)에서는 각 블록이 최소 0부터 최대 33까지의 지우기 값을 가지고 있으며 블록별로 편차가 크다는 것을 알 수 있다.

개선한 알고리즘을 적용할 경우의 분포도는 (그림 4)의 그래프와 같다.



(그림 4) 평준화 알고리즘의 블록별 지우기

(그림 4)는 (그림 3)에 비해 전체적으로 중간층의 밀도가 높으며 즉 편차가 크게 줄어든 것을 알 수 있다. 이는 총 지우기 횟수에 영향 없이 분산도를 줄인 데에 큰 의미가 있다. 분포도를 나타내는 분산값의 비교 또한 다음 (그림 5)와 같이 크게 향상 된 것을 볼 수 있다.



(그림 5) 개선 전후의 분산값 비교

6. 결론

본 연구에서는 기존에 이루어진 많은 FTL 알고리즘 연구 및 이루어질 연구들이 플래시 메모리의 수명에 있어서 절대적인 지표 없이 상대적인 평가들만 이루어지는 것을 보완하기 위해 진행 되었다. 알고리즘 개발 및 연구 시에 사용

되는 트레이스를 본 알고리즘에 활용하여 최적의 상태를 확인 후 자신의 알고리즘을 실험하여 비교함으로써 알고리즘의 완성도를 높일 수 있다.

참 고 문 헌

[1] S. K. Lee, S. L. Min, Y. K. Cho, "Current trends on flash memory technology," Journal of KIISE, vol.24, no.12, pp.99-106, Dec. 2006. (in Korean)

[2] Tae-Sun Chung, Dong-Joo Park, Dong-Ho Lee, Sang-Won Lee, and Ha-Joo Song, "System Software for Flash Memory: A Survey", 2004.

[3] 박정수, 민상렬, "마모 제어 영역을 활용한 플래시 메모리 마모평준화", 정보과학회논문지: 컴퓨팅의 실제 및 레터 제16권 제12호(2010.12)

[4] 이승환, 이태훈, 정기동, "플래시 메모리를 위한 페이지 비율 분석 기반의 적응적 가비지 컬렉션 정책", 정보과학회논문지 : 시스템 및 이론 제 36 권 제 5 호(2009.10)

[5] L.-P. Chang, "On efficient wear-leveling for largescale flash-memory storage systems," Proc. of the 2007 ACM symposium on Applied computing, pp.1126-1130, 2007.

[6] Y.-H. Chang, J.-W. Hsieh, T.-W. Kuo, "Improving flash wear-leveling by proactively moving static data," IEEE Transactions on Computers, vol.59, no.1, pp.53-65, Jan. 2010.

[7] Jen-Wei Hsieh, Li-Pin Chang, Tei-Wei Kuo, "Efficient On-line Identification of Hot Data for Flash-Memory Management" ACM symposium on applied computing SAC 05, pp. 838, 2005.

[8] S.-W. Lee, D.-J. Park, T.-S. Chung, D.-H. Lee, S. Park, H.-J. Song, "A log buffer-based flash translation layer using fully-associative sector translation," ACM Transactions on Embedded Computing Systems (TECS), vol.6, no.3, Jul. 2007.

[9] J. Kim, J.M. Kim, S.H. Noh, S.L. Min, and Y.Cho, "A Space-Efficient Flash Transaction on Consumer Electronics, Vol. 48, No.2, pp.366-375, 2002.

[10] Sivan Toledo, "Algorithms and Data Structures for Flash Memories" ACM Computing Surveys.

[11] J.-U. Kang, H. Jo, J.-S. Kim, J. Lee, "A superblock-based flash translation layer for NAND flash memory," Proc. of the 6th ACM & IEEE International Conference on Embedded Software, pp.161-170, 2006.

[12] X.-Y. Hu et al.,"Write amplication analysis in hash-based solid state drives", SYSTOR 2009

[13] Ioannis Koltsidas,Stratis D. Viglas, Data management over flash memory", SIGMOD '11

정 호 영

2004년 : 한양대학교 공과대학 재료공학부 (공학사)
2006년 : 한양대학교 대학원 정보통신공학과 (공학석사)
2011년 : 한양대학교 대학원 전자컴퓨터통신공학과 (공학박사)
2011년~현 재 : LG전자 CTO SW Platform (연) 선임연구원
관심분야 : 운영체제, 데이터베이스, 파일 시스템, 플래시 메모리, 병렬 프로그래밍

차 재 혁

1987년 : 서울대학교 계산통계학과 (이학사)
1991년 : 서울대학교 컴퓨터공학과 (공학석사)
1997년 :서울대학교 컴퓨터공학과 (공학박사)
1998년~2001년 : 한양대학교 컴퓨터교육과 조교수
2002년~현 재 : 한양대학교 컴퓨터공학부 교수
관심분야 : 데이터베이스, 파일시스템, 플래시 메모리, 이러닝, 콘텐츠변환 등

이 태 화

2011년 : 안양대학교 컴퓨터학과 (공학사)
2011년~현 재 : 한양대학교 대학원 정보통신공학과 (석사과정)
관심분야 : 데이터베이스, 파일시스템, 플래시 메모리