http://dx.doi.org/10.7840/kics.2016.41.10.1167

플래시 변환 계층에서 시간적 지역성을 이용하여 쓰기 요청을 처리하는 효율적인 페이지 레벨 매핑 알고리듬

이 해 룡◆, 황 선 영°

An Efficient Page-Level Mapping Algorithm for Handling Write Requests in the Flash Translation Layer by Exploiting Temporal Locality

Hai-Long Li◆, Sun-Young Hwang°

요 약

본 논문에서는 플래시 메모리의 FTL에서 페이지 매핑 기법을 기반으로 소거횟수를 줄이는 알고리듬을 제안한다. 제안된 알고리듬은 버퍼에서 매 쓰기요청들의 가중치들을 유지하고 이용하여 현재 쓰여질 요청의 시간적 지역성의 정도를 판단한다. 시간적 지역성을 효율적으로 이용하여 핫 요청을 판단하기 위해 현재 쓰여질 요청은 실험적으로 정한 기준점보다 높은 시간적 지역성을 가져야 한다. 반면 LRU 알고리듬을 이용한 FTL에서는 새로 쓰여질 요청을 항상 시간적 지역성이 높은 요청으로 판단하여 데이터를 순차적으로 저장하지만 제안된 알고리듬을 사용하여 판단된 핫 요청들의 데이터는 핫 블록에 집중적으로 저장한다. 핫 블록에 저장된 데이터들은 웜 블록의 데이터들보다 자주 업데이트되어 Garbage Collection 수행 시 핫 블록들 중 무효한 페이지가 많은 블록이 주로 희생블록으로 선택되므로 소거연산의 시작을 지연시켜 전체 소거횟수를 줄인다. 임의적인 요청을 위주로 하는 실제 I/O시스템에서 추출한 트레이스 파일들을 적용하여 검증한 결과, 기존의 LRU 알고리듬을 사용하는 경우에 비해 소거횟수는 9.3% 줄어들었다.

Key Words : NAND flash memory, FTL, Page-level mapping, Temporal locality, Random request

ABSTRACT

This paper proposes an efficient page-level mapping algorithm that reduces the erase count in the FTL for flash memory systems. By maintaining the weight for each write request in the request buffer, the proposed algorithm estimates the degree of temporal locality for each incoming write request. To exploit temporal locality deliberately for determination of hot request, the degree of temporal locality should be much higher than the reference point determined experimentally. While previous LRU algorithm treats a new write request to have high temporal locality, the proposed algorithm allows write requests that are estimated to have high temporal locality to access hot blocks to store hot data intensively. The pages are more frequently updated in hot blocks than warm blocks. A hot block that has most of invalid pages is always selected as victim block at Garbage Collection, which results in delayed erase operation and in reduced erase count. Experimental results show that erase count is reduced by 9.3% for real I/O workloads, when compared to the previous LRU algorithm.

※ 본 연구는 삼성전자(주)의 지원으로 수행 하였습니다(No. 201470063.03).
◆ First Author : Sogang University Department of Electronic Engineering, lhl2014@sogang.ac.kr, 학생회원
° Corresponding Author : Sogang University Department of Electronic Engineering, hwang@sogang.ac.kr, 종신회원
논문번호 : KICS2016-08-003, Received August 11, 2016; Revised September 12, 2016; Accepted September 28, 2016

Ⅰ. 서 론

최근 웨어러블 기기의 상용화가 빠르게 이루어지면서 주요 메모리로 사용되는 플래시 메모리는 지속적으로 발전되고 있고 일반용 컴퓨터와 서버에서 대용량 저장장치로도 많이 사용된다. 특히 현재 On-Line Transaction Processing (OLTP)를 하는 기업용 서버에서는 SSD로 대체해나가고 있다[1,2]. 대량의 데이터를 정확하고 빠르게 처리하기 위해 사람들은 더 좋은 성능을 가진 SSD를 필요로 하고 있는데 향상된 SSD를 설계하기 위해 고려해야 할 특징들이 있다.

NAND 플래시 메모리는 장점이 많고 기술 발전도 빠른 매체이긴 하나, 데이터 덮어쓰기가 불가능하여 기존의 데이터를 소거한 후 쓰기를 해야 한다[3,4]. 페이지단위로 읽고 쓰는 연산을 하고 블록단위로 소거연산을 하며 소거연산은 읽기와 쓰기연산보다 느리므로 속도차이도 고려하여 데이터를 저장해야 한다. 메모리를 사용할수록 수명이 줄어드나 매 블록들의 제한된 소거횟수를 초과할 경우 남은 수명이 급격히 줄어든다. 이러한 NAND 플래시 메모리의 특징으로 인해 여러 저장장치 관리기법을 요구하고 이를 효율적으로 구현하기 위해 그림 1과 같이 Flash Translation Layer (FTL)이 필요하다. FTL은 플래시 메모리와 파일 시스템 사이에 존재하며 주로 주소변환, Garbage Collection, wear leveling 역할을 수행한다[3,5-12]. 특정 페이지의 데이터를 수정하고자 할 때 새로운 데이터를 다른 빈 페이지에 기록하고 이전 데이터는 무효화시키며 새로운 데이터의 주소 매핑정보를 유지하기 위해 여러 주소변환기법이 FTL설계의 주요 이슈로 되었고 주로 페이지 매핑, 블록 매핑, 하이브리드 매핑을 사용한다[3,10]. 페이지 매핑은 페이지 단위로 매핑 테이블을 만들어 물리적인 페이지 번호와 논리적인 페이지 번호를 매핑한다. 한 번의 매핑 테이블의 검색으로 실제 플래시 메모리의 물리적인 페이지 번호를 알아서 빠르게 동작하여 크기가 작고 임의적인 특성을 가진 데이터 처리에 높은 성능을 보이나 플래시 메모리의 용량이 증가하는 만큼 매핑정보를 저장하기 위해 많은 양의 RAM을 사용한다. 반면 블록 매핑은 물리적인 블록 번호와 논리적인 블록 번호를 매핑하여 메모리 블록의 수만큼 테이블을 RAM에 저장한다. 하이브리드 매핑은 블록 매핑을 이용하여 RAM의 사용량을 줄이고 블록 내에서 페이지 매핑을 하여 두 가지 매핑의 장점을 이용한다. 블록 매핑과 하이브리드 매핑은 RAM의 사용량을 줄이지만 빈 블록을 얻기 위해 무효한 데이터가 많은 블록을 소거하는

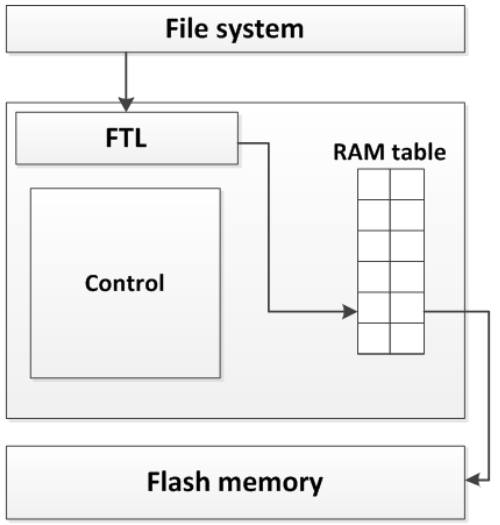

그림 1. FTL을 사용할 때 시스템 구조.
Fig. 1. Architecture using FTL.

Garbage Collection 수행 시 오버헤드가 페이지 매핑보다 크다. Garbage Collection 은 주요하게 유효한 페이지들을 이동하는 오버헤드와 소거연산에 의한 오버헤드가 존재한다[9]. 이러한 오버헤드는 데이터 분류단계와 데이터 할당을 통해 줄일 수 있다. 데이터 분류를 위해 데이터 특징을 판단하는 과정이 필요하고 일반용 컴퓨터나 서버에 사용되는 데이터들은 시간적 지역성이나 공간적 지역성을 가지고 있다[13]. 시간적 지역성은 최근 참조된 데이터는 가까운 장래에도 계속 참조될 가능성이 높은 특징이 있고 공간적 지역성은 현재 접근중인 데이터의 위치와 인접된 데이터들은 계속 참조될 가능성이 높은 특징이 있다.

기업용 서버에서 많은 사용자들이 동시에 사용하여 데이터가 임의적으로 접근되는 경향성이 있다[14]. SSD의 성능을 향상시키기 위해 SSD의 특성과 데이터의 특징도 함께 고려하면 더 좋은 성능을 기대할 수 있다[15]. 기업용 서버에서 추출한 실제 I/O 워크로드는 임의적인 접근의 경향성을 보이면서 크기가 작고 시간적 지역성도 나타내고 있어[1,14] 본 논문에서는 임의적인 데이터를 처리하는 상황을 가정하여 연구를 수행하였고 데이터 분류와 데이터 할당을 통해 소거연산의 시작을 지연시켜 전체 소거횟수를 줄인다.

본 논문의 구성은 다음과 같다. 2절에서는 기존의 데이터 분류방법과 연구 동기에 대해 설명하고, 3절에서는 본 논문에서 제안된 알고리듬을 설명한다. 4절에서는 기존의 알고리듬과 성능비교를 한 실험결과를

보인다. 마지막으로 5절에서는 결론을 제시한다.

Ⅱ. 관련연구 및 동기

본 절에서는 기존의 데이터 분류방법과 연구 동기에 대해 서술하였다.

2.1 기존의 데이터 분류방법

NAND 플래시 메모리에서 덮어쓰기가 불가능하여 특정 페이지의 데이터를 수정하고자 할 때 새로운 데이터를 다른 빈 페이지에 기록하고 이전 데이터는 무효화시킨다. Garbage Collection 수행 시 유효한 페이지를 이동하는 오버헤드와 소거연산에 의한 오버헤드에 의해 속도가 느려져 데이터 분류단계에서 데이터의 특징을 판단하여 데이터 할당에 이용하면 오버헤드를 줄일 수 있다. 시간적 지역성을 가진 데이터들은 가까운 장래에 자주 업데이트되기에 이전 데이터들은 무효화과정이 빈번하다. 이러한 데이터들을 집중적으로 같은 블록에 저장하면 효율적인 Garbage Collection 이 가능하다. 공간적 지역성을 가진 데이터들은 참조되면 접근중인 데이터와 인접한 데이터들을 계속 참조하므로 함께 관리하면 Garbage Collection 오버헤드가 줄어든다. 파일 시스템에서 받은 쓰기요청은 순차적인 쓰기와 임의적인 쓰기로 나타난다[13]. 순차적인 쓰기요청은 공간적 지역성을 가지고 있고 데이터를 업데이트하는 빈도가 시간적 지역성을 가진 임의적인 요청보다 적어, 혼합해서 데이터가 할당되면 Garbage Collection 오버헤드가 늘어나 성능이 저하된다[1,16]. 이러한 문제를 해결하기 위해 데이터들의 특징을 판단하는 방법들이 연구되었다. 전형적인 방법인 LRU 알고리듬[7,17]은 최근에 접근한 데이터들을 항상 시간적 지역성이 높다고 판단하여 접근 최신성만 고려하여 데이터정보를 유지하나, 실제 데이터는 시간적 지역성이 종종 적게 나타나기도 한다. 데이터 접근 빈도로 판단하는 알고리듬은 시간적 지역성을 판단할 때 데이터가 자주 업데이트되는 시간외에도 시간적 지역성이 높다고 판단하는 단점이 있다[6]. LAST[13]에서는 데이터의 특성을 크기에 의해 판단하나 크기가 작은 순차적인 데이터와 임의적인 데이터들이 혼합하여 나타날 때 오버헤드가 크다.

2.2 연구동기

FTL 알고리듬은 NAND 플래시 메모리 특성상 소거시간이 데이터를 읽거나 쓰는 시간에 비해 길어 소거연산을 적게 할수록 성능이 좋아진다. 소거횟수의 증가는 NAND 플래시 메모리의 수명감소의 원인이기도 하다. 실제 일반용 컴퓨팅시스템에서는 그림 2와 같은 멀티 태스킹[13]을 한다. 큰 순차적인 쓰기요청은 파일 시스템을 통해서 작은 쓰기요청으로 분리하고 다른 태스크의 요청들과 혼합하여 FTL에 전달한다. 기업용 서버 경우 많은 사용자들이 같은 시스템에서 동시에 읽고 쓰기를 요청하여 결과적으로 실제 I/O 워크로드에서 접근 패턴은 임의적으로 나타나고 크기가 작고 시간적 지역성을 보이며 전체 요청 중 큰 비중을 차지한다[1]. 일반용 컴퓨팅시스템과 기업용 서버시스템에서 하이브리드 매핑 기법을 이용할 때 임의적인 요청 때문에 오버헤드가 큰 전체 합병연산[10-12]은 존재할 수 밖에 없다. 전체 합병연산을 수행하지 않고 오직 교환연산과 부분 합병연산만을 할 수 있도록 페이지 매핑 기법을 기반으로 하는 알고리듬이 연구되고 있다[7]. 최근에 제안된 페이지 매핑기법을 기반으로 된 알고리듬인 DFTL방법[7]은 매핑정보를 저장하는 RAM의 양을 줄이기 위해 전체 매핑정보는 플래시 메모리에 저장하고 일부분만 RAM에 저장하여 사용된다. 쓰기요청의 시간적 지역성을 고려하여 LRU 알고리듬[17]을 사용한다. 이 알고리듬은 새로 쓰여질 모든 요청들은 시간적 지역성이 높다고 판단하여 최근에 쓰여진 매핑정보를 RAM에 기억하고 데이터들을 저장한다. 실제 쓰기요청들은 종종 시간적 지역성이 적게 나타나서 기존의 LRU 알고리듬을 사용한 FTL에서는 시간적 지역성이 높은 데이터와 기타 데이터들이 혼합하여 데이터블록에 저장되므로 효율적으로 Garbage Collection 을 수행하지 못한다. 높은 시간적 지역성을 가진 요청들을 효율적으로 판단하고 페이지

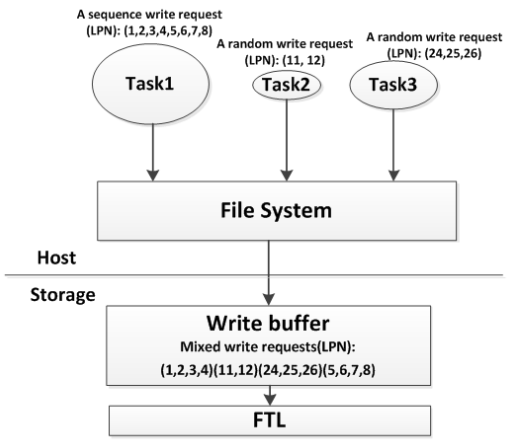

그림 2. 실제 쓰기요청의 표현.
Fig. 2. Behaviors of real write requests.

할당정책에 사용하면 좋은 결과를 기대할 수 있다. 이를 개선하기 위해 쓰기요청들이 저장된 버퍼에서 정해진 크기의 윈도우내의 정보를 이용하여 페이지 점수를 계산하고 특정기준점수와 비교하여 쓰기요청의 핫 여부를 판단하는 알고리듬을 제안하여 페이지 할당정책에 사용하였다.

Ⅲ. 제안한 알고리듬

본 절에서는 제안한 알고리듬에 관해 기술한다.

3.1 제안한 알고리듬의 전체 흐름

FTL에서 핫 데이터는 높은 시간적 지역성을 나타내고 웜 데이터는 낮은 시간적 지역성을 나타낸다. 핫 데이터들은 자주 업데이트되어 집중적으로 저장하면 블록을 효율적으로 사용할 수 있으므로 웜 데이터와 구별하여 저장하고, 이들 블록이 핫 블록, 웜 블록이며, 전체 페이지 매핑정보를 저장하는 블록들은 전환 블록이다. 핫 블록에 저장된 페이지는 핫 매핑 테이블(HMT)의 매핑정보를 통해서 매핑하고 웜 블록에 저장된 페이지는 웜 매핑 테이블 (WMT)의 매핑정보를 통해서 매핑하며 HMT와 WMT는 모두 RAM에 기억한다. HMT와 WMT에 없는 페이지 매핑정보는 전환 블록의 주소를 기억하고 있는 글로벌 트랜즈레션 디렉터리(GTD)를 통해서 찾는다.

제안한 알고리듬은 페이지 할당정책(page allocation)에 사용하기 위해 핫/웜 데이터를 효율적으로 인식하고 그림 3과 같은 과정을 보인다. 유지된 버퍼 윈도우내의 쓰기 요청 정보를 이용하여 현재 쓰여질 요청의 시간적 지역성을 효율적으로 판단할 수 있는 점수를 계산하고 실험적으로 정한 특정기준점수와 비교하여 핫 여부를 결정한다. 판단된 시간적 지역성이 높은 요청은 핫 블록에 데이터를 저장하고 핫 매핑 테이블에 매핑정보를 업데이트한다.

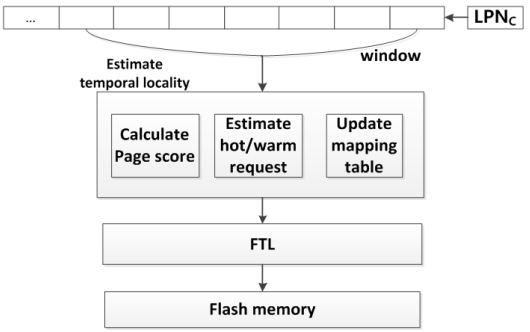

그림 3. 제안한 알고리듬의 전체 과정.
Fig. 3. Overall process of the proposed algorithm.

3.2 제안한 알고리듬의 구체적인 동작

제안한 알고리듬은 그림 3과 같이 핫/웜 데이터를 효율적으로 인식하고 페이지 할당정책에 사용하기 위해 현재 쓰여질 쓰기 요청의 핫 여부를 판단한다. 기존의 LRU 기반 알고리듬은 최근에 발생한 요청을 핫 데이터로 판단하여 우선적으로 유지한다. 이 방법은 쓰기 요청의 접근 최신성은 반영하나 접근 빈도는 적절히 반영하지 못하는 단점이 있다. 제안한 알고리듬은 쓰기 요청의 핫 여부를 결정하기 위해 접근 빈도와

```
{
Step1:
   LPNc = current LPN;
   LPNi = LPNi's in the buffer window ( i = 1, 2 ... window_size );
   Weighti = weight of LPNi ( 1, 2 ... window_size );
   Intervali = window_size - i;     /* distance from the last entry in the buffer. */
   Scorec = 0; /* Scorec: Score of the current page */
   Insert LPNc to the buffer;
Step2:
   for each LPNi ( i = 1, 2 ... window_size )
   {
      if ( LPNi = LPNc ) {
      Scorec = Scorec + Weighti / (1 + Intervali) ;
   }
Step3:
   if Scorec ≥ Threshold {
      write_hot_block(LPNc);
      /* Allocate the current request block at the hot block region */
      update_hot_mapping_table( LPNc, Scorec );
      /* Update the entry and score of LPNc in the hot mapping table. Make an entry and score of LPNc if non-existent */
   }
   else {
      write_warm_block(LPNc);
      /* Allocate the current request block at the warm block region */
      update_warm_mapping_table( LPNc );
      /* Update the entry of LPNc in the hot mapping table. Make an entry of LPNc if non-existent */
   }
   Shift the buffer window by one towards the direction of new inserted LPN;
   Update Weighti and Intervali of all the entries in the buffer window;
}
```

그림 4. 시간적 지역성이 높은 쓰기요청을 분할하는 전반적인 알고리듬.
Fig. 4. Overall separation algorithm for hot write requests.

접근 최신성을 동시에 반영하였다. 그림 4는 시간적 지역성이 높은 쓰기 요청을 분할하는 전반적인 알고리듬이다.

먼저 쓰기 요청의 시간적 지역성을 판단할 때 사용되는 페이지 점수(Page Score)를 계산하기 위해 윈도우내에 쓰기 요청의 정보를 유지한다. 윈도우내의 쓰기 요청들은 모두 가중치(Weight)와 위치차이(Interval)정보를 가지고 있다. 가중치는 요청들의 접근 최신성을 반영하고 버퍼 윈도우내에서 후에 삽입된 쓰기 요청의 가중치일수록 더 크게 주며 위치차이는 버퍼에서 마지막으로 삽입된 쓰기 요청과 버퍼에 저장된 요청사이의 순번차이이다.

$$Score_C = \sum_{i=0}^{window\,size} \frac{Weight_i}{Interval_i}$$

다음 현재 쓰여질 요청(LPN_C)의 페이지 점수를 주어진 식에 따라 계산한다. LPN_C는 마지막으로 버퍼에 삽입된 쓰기 요청이다. 주어진 식에서 $Weight_i$는 버퍼 윈도우내에서 LPN_C와 같은 값 LPN을 갖는 요청의 가중치이고 $Interval_i$는 버퍼에 저장된 LPN_C와 같은 값 LPN을 갖는 요청사이의 위치차이이므로 $\frac{Weight_i}{Interval_i}$는 그 LPN의 접근 최신성을 반영한다. $Score_c$는 LPN_C의 페이지 점수이고 버퍼 윈도우내의 같은 값 LPN을 갖는 요청들의 $\frac{Weight_i}{Interval_i}$를 더한 합이므로 요청들의 접근 최신성과 접근 빈도가 동시에 적절히 반영되었다. 버퍼 윈도우내에서 같은 값 LPN을 갖는 요청이 없을 경우 위치차이는 무한히 크다고 가정하여 페이지 점수는 0이다. 그림 5는 버퍼에서 윈도우크기가 7일 때 현재 쓰여질 LPN_C가 8, 2, 7의 페이지 점수를 각각 구하는 과정을 보인다.

마지막으로 쓰여질 요청의 페이지 점수는 실험적으로 정한 특정기준점수와 비교하여 핫 여부를 결정하고 페이지 할당정책에 사용된다. 특정기준점수를 실험적으로 정하는 과정은 실험결과부분에서 추가적으로 보여준다. 특정기준점수보다 크거나 같으면 핫 요청으로 판단하여 핫 블록에 데이터를 저장하고 핫 매핑 테이블에 매핑정보를 업데이트하며 작으면 웜 요청으로 판단하여 웜 블록에 데이터를 저장하고 웜 매핑 테이블에 매핑정보를 업데이트 한다. 핫 매핑 테이블에는 핫 요청의 엔트리와 페이지 점수를 저장하고 웜 매핑 테이블에는 웜 요청의 엔트리를 저장한다. 매핑 테이블의 교체정책에서 핫 매핑 테이블은 쓰기 요청의 페이지 점수가 높은 요청의 엔트리를 우선적으로 유지하고 낮은 요청의 엔트리는 웜 매핑 테이블로 이동시킨다. 웜 매핑 테이블은 최근에 사용된 요청의 엔트리를 우선적으로 유지하고 오래 사용되지 않은 요청의 엔트리를 전환블록에 업데이트한다.

LRU기반 알고리듬은 접근의 최신성만 고려하고 현재 쓰여질 요청은 항상 높은 시간적 지역성을 가지고 있다고 판단하여 데이터 블록에 순차적으로 저장하여 실제 데이터 블록에는 핫/웜 데이터들이 혼합하여 나타나지만 제안한 알고리듬은 접근의 최신성과 접근의 빈도를 동시에 반영하여 요청의 핫 여부를 판단하고 페이지 할당정책에 사용하기에 쓰기요청의 시간적 지역성을 효율적으로 판단하여 핫 데이터들을 핫 블록에 집중적으로 저장할 수 있다. 쓰기요청의 시간적 지역성을 신중히 고려하여 판단된 핫 데이터들

No.	1	2	3	4	5	6	7
LPN	7	7	5	3	7	2	4
Weight	1	2	3	4	5	6	7
Interval	7	6	5	4	3	2	1

LPN_C

If LPN_C = 8, no same LPN in window. Score(8) = 0.

If LPN_C = 2, no same LPN in window. Score(2) = $\frac{W_2}{I_2}$ = 3.

If LPN_C = 7, no same LPN in window. Score(7) = $\frac{W_3}{I_3}+\frac{W_6}{I_6}+\frac{W_7}{I_7}$ = 2.14.

그림 5. 윈도우내에서 페이지 점수를 구하는 예.
Fig. 5. An example calculating page scores in window.

Write Request															
No.	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
LPN	33	1	2	10	11	7	4	5	7	33	3	10	11	7	33	10

Window size = 10, Reference Point = 1.2										
Weight	10	9	8	7	6	5	4	3	2	1
LPN	11	10	7	17	33	21	20	19	4	3
Interval	1	2	3	4	5	6	7	8	9	10

HMT	Hot blocks	Hot blocks	Hot blocks	Warm blocks	Warm blocks
10	33	33	33	1	5
11	10	10	10	2	3
7	11	11		7	
33	7	7		4	

(a)

Data blocks	Data blocks	Data blocks	Data blocks
33	11	7	11
1	7	33	7
2	4	3	33
10	5	10	10

(b)

그림 6. 제안한 알고리듬과 기존의 알고리듬을 이용한 FTL에서 데이터를 저장하는 예, (a) 제안한 알고리듬 (b) 기존의 알고리듬.
Fig. 6. An example of data placement using proposed algorithm and LRU algorithm in FTL. (a) The proposed algorithm (b) LRU algorithm.

은 자주 업데이트되어 Garbage Collection 수행 시 핫 블록들 중 무효한 페이지가 많은 블록이 희생블록으로 선택되므로 소거연산의 시작을 지연시켜 전체 소거횟수를 줄인다. 그림 6은 제안한 알고리듬과 기존의 LRU 알고리듬을 이용한 FTL에서 데이터를 저장하는 과정을 보인다. 그림 6(a)는 제안한 알고리듬을 이용한 FTL에서 데이터를 저장하는 과정을 보이고 그림 6(b)는 LRU 알고리듬을 이용한 FTL에서 데이터를 저장하는 과정을 보인다. 제안한 알고리듬을 이용한 FTL에서 페이지를 할당할 때 블록들을 더 효율적으로 이용하므로 무효화된 페이지들이 더 집중적으로 나타난다.

Ⅳ. 실험결과

제안한 알고리듬을 검증하기 위해 Disksim[18]시뮬레이터에 플래시 모듈을 추가하여 SSD환경을 가상으로 조성한 트레이스 구동 시뮬레이터 Flashsim[19]을 사용하여 시뮬레이션을 수행하였다. 실험에 사용된 플래시 메모리 모델의 사양은 표 1에 제시되었으며 4GB(=524,288페이지)의 용량의 NAND 플래시 메모리를 사용하였다. 실제 I/O 워크로드를 입력으로 하여 시뮬레이션을 수행하였다. financial1[20]과 financial2[20]는 큰 금융기업의 온라인 무역처리시스템에서 추출하였고 TPC-C[21]와 TPC-E[21]는 마이크로소프트회사에서 각각 TPC-C 벤치마크와 TPC-E 벤치마크를 실행한 기기에서 추출하였으며 CAMRSHMSA[21]는 Microsoft Research Center(MRC)에서 사용하는 기업용 서버로부터 추출하였다. 또 DisplayAdsPayload[21], MSN[21], RADIUS_SQL[21] 는 마이크로소프트 프로덕션들의 서버에서 추출하였다. 그 특성들은 표 2와 같다. 제안한 알고리듬은 쓰여질 요청의 시간적 지역성을 효율적으로 계산하여 핫/웜 데이터를 판단하고 페이지 할당정책에 사용하여 소거연산의 시작을 지연

표 1. 실험에 사용된 NAND 플래시 메모리의 파라미터
Table 1. Parameter of NAND flash memory used for simulation.

Organizations of NAND flash memory	Block size	1MB
	Page size	8KB
	Number of pages per block	128
Each operation of access time	Read operation	250us
	Write operation	1.3ms
	Erase operation	1.5ms

표 2. 기업용 workload의 특성
Table 2. Characteristic of Enterprise-Scale Workload.

Workload info. / Workload type	Total Request Num	Average Request Size (KB)	Read (%)	Sequentiality (%)
financial	5,334,986	9.8	23.1	1.1
financial2	3,699,194	7.7	82.3	1.0
TPC-C	6,834,303	17.1	64.5	0
TPC-E	1,372,870	21.0	7.6	0
CAMRSHMSA	3,993,315	16.7	35.5	0.1
DisplayAdsPayload	48,717	92.8	45	14.6
MSN	1,685,121	28.2	79.6	1.4
RADIUS_SQL	380,245	68.8	17	19.2

시킨다. 데이터의 핫 여부는 유지된 버퍼에서 윈도우 내의 쓰기요청의 정보를 이용하여 현재 쓰여질 요청의 페이지 점수를 계산하고 특정기준점수와 비교하여 높으면 핫 데이터로 낮으면 웜 데이터로 판단하므로 버퍼 윈도우내의 정보가 많을수록 데이터를 효율적으로 판단한다. 그림 7은 특정기준점수를 변하지 않고 실제 I/O 워크로드를 입력으로 시뮬레이션을 하였을

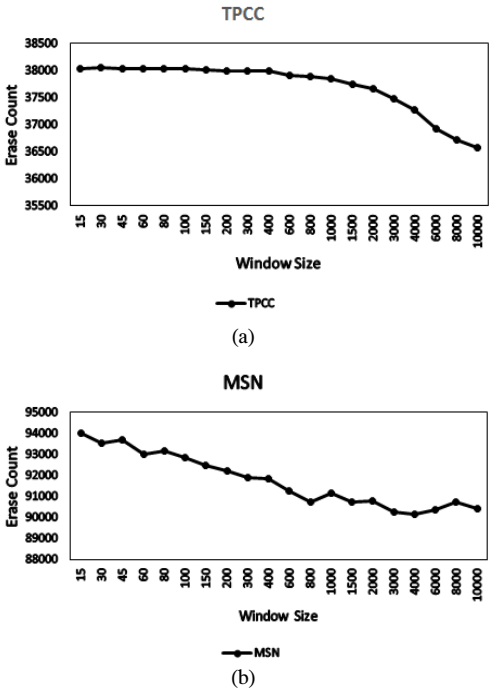

그림 7. 버퍼 윈도우의 크기와 소거횟수사이 관계. (a) TPC-C (b) MSN.
Fig. 7. Buffer window size and erase count in different real I/O tracefiles. (a) TPC-C (b) MSN.

때 버퍼 윈도우의 크기와 소거횟수사이 관계를 보인다. 버퍼 윈도우의 크기가 클수록 데이터의 핫 여부를 효율적으로 판단하여 핫 데이터를 집중적으로 저장할 수 있기에 소거연산의 시작을 지연시켜 소거횟수를 감소시킨다. 그림 8은 버퍼 윈도우의 크기를 변하지 않고 특정기준점수와 소거횟수사이 관계를 보인다. 버퍼 윈도우의 크기의 정보를 이용하여 계산된 쓰여질 요청의 페이지 점수는 특정기준점수와 비교하여 핫 여부를 판단하므로 특정기준점수가 높을수록 실제 핫 데이터로 판단되어야 할 요청들을 웜 데이터로 판단할 수 있고 낮을수록 실제 웜 데이터로 판단되어야 할 요청들을 핫 데이터로 판단할 수 있어 핫 블록 혹은 웜 블록에 핫/웜 데이터가 혼합하여 나타나기에 소거연산을 효율적으로 진행하지 못한다. 실제 소거횟수는 특정기준점수의 결정으로 영향이 크지 않으나 특정기준점수가 낮을수록 소거횟수가 평균적으로 적으므로 낮은 특정기준점수를 실험적으로 정한다. 버퍼 윈도우내의 쓰기요청의 정보들이 소거횟수에 큰 영향을 주나 저장해야할 정보의 양이 많을수록 검색하고 계산하는 시간이 길어지는 우려가 존재한다. 본 실험에서는 버퍼 윈도우의 크기를 100으로 하고 특정기준점수

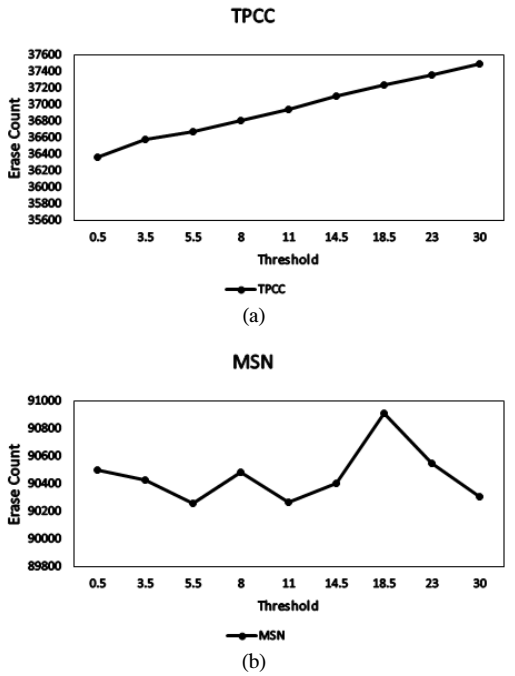

그림 8. 특정기준점수와 소거횟수사이 관계. (a)TPC-C (b)MSN.
Fig. 8. Threshold and erase count in different real I/O tracefiles. (a) TPC-C (b) MSN.

표 3. 실제 workload에 대한 성능비교
Table 3. Comparison results by real workload.

Comparison items / Tracefiles	Erase Count		
	DFTL	Proposed	Comparison(%)
financial1	59,139	57,175	-3.3
financial2	10,162	9,187	-9.6
TPC-C	48,488	38,037	-21.6
TPC-E	21,660	16,679	-23
CAMRSHMSA	49,664	44,963	-9.5
DisplayAdsPayload	2,721	2,654	-2.5
MSN	94,121	93,603	-0.6
RADIOUS_SQL	13,681	13,079	-4.4

를 3.5로 하며 실험적으로 결정하였다. 표 3에서는 다양한 실제 I/O 워크로드에서 추출한 트레이스파일들을 입력으로 시뮬레이션하여 소거횟수를 비교한 결과를 제시하였다. 제안한 알고리듬을 적용한 FTL은 LRU 기반 DFTL보다 소거횟수가 평균 9.3%줄어들었다.

V. 결론 및 추후연구

본 논문에서는 페이지 매핑기법을 기반으로 데이터 분류단계와 데이터 할당을 통해 Garbage Collection에서 생기는 오버헤드를 줄여 전체 소거횟수를 감소시키는 방법을 제안하였다. 데이터 분류를 위해 데이터 특징을 판단하며 기존에는 주로 데이터 접근 최신성, 접근 빈도, 데이터 크기 등을 각각 이용하는 방법들이 제안되었다. 제안된 알고리듬은 데이터 접근 최신성과 접근 빈도를 동시에 반영하여 데이터 특징을 판단하고 데이터 할당에 사용하였다. 버퍼 윈도우내의 정보를 이용하여 쓰여질 요청의 페이지 점수를 계산하고 특정기준점수와 비교하여 핫 여부를 판단하며 페이지 할당정책에 사용하여 소거연산의 시작을 지연시켜 전체 소거횟수를 줄인다. 기존의 알고리듬과 비교하여 소거횟수는 평균 9.3% 감소하였다. 추후, 데이터 크기도 함께 고려하여 데이터를 세분화하게 분류하여 데이터 할당하는 연구가 더 필요하다. 데이터 접근 최신성, 접근 빈도 외에 데이터 크기도 동시에 고려하여 데이터를 분류하면 보다 좋은 성능을 기대할 수 있다.

References

[1] Intel Corporation, *OLTP Performance Com-*

parison: Solid state Drives vs. Hard Disk Drives, Retrieved Jan. 2009, from http://www.principledtechnologies.com/Inetel/OLTPSSDPerf0109.pdf.

[2] S. Lee, B. Moon, and C. Park, "Advances in flash memory SSD technology for enterprise database applications," in *Proc. Management of Data Conf.*, pp. 863-870, New York, NY, Jun. 2009.

[3] D. Ma, J. Feng, and G. Li, "A survey of address translation technologies for flash memories," *ACM Computing Surveys*, vol. 46, no. 3, Article 36, Jan. 2014.

[4] S. Lee and H. Bahn, "Characterizing memory references for smartphone applications and its implications," *J. Semiconductor Technol. Sci.*, vol. 15, no. 2, Apr. 2015.

[5] Q. Wei, et al., "Z-MAP: A zone-based flash translation layer with workload classification for solid-state drive," *ACM Trans. Storage*, vol. 11, no. 1, Article 4, Feb. 2015.

[6] J. Hsieh, T. Kuo, and L. Chang, "Efficient identification of hot data for flash memory storage systems," *ACM Trans. Storage*, vol. 2, no. 1, pp. 22-40, Feb. 2006.

[7] A. Gupta, Y. Kim, and B. Urgaonkar, "DFTL: A flash translation layer employing demand-based selective caching of page-level address mappings," in *Proc. Architectural Support for Program. Lang. Operating Syst. Conf.*, pp. 229-240, New York, NY, Mar. 2009.

[8] D. Kim and S. Hwang, "An efficient wear-leveling algorithm for NAND flash SSD with multi-channel and multi-way architecture," *J. KICS*, vol. 39B, no. 7, pp. 425-432, Jul. 2014.

[9] L. Chang, Y. Liu, and W. Lin, "Stable greedy: Adaptive garbage collection for durable page-mapping multichannel SSDs," *ACM Trans. Embed. Comput. Syst.*, vol. 15, no. 13, Article 13, Jan. 2016.

[10] J. Kim, et al., "A space-efficient flash translation layer for compactflash systems," *IEEE Trans. Consumer Electron.*, vol. 48, no. 2, pp. 366-375, May 2002.

[11] S. Choe, H. Cho, and T. Chung, "Garbage collection-centric FTL for improving the processing delay of flash memory," *J. KIISE*, vol. 40, no. 4, pp. 181-192, Aug. 2013.

[12] J. Kang, et al., "A superblock-based flash translation layer for NAND flash memory," in *Proc. Embedded software Conf.*, pp. 161-170, New York, NY, Oct. 2006.

[13] S. Lee, et al., "LAST: Locality-aware sector translation for NAND flash memory-based storage systems," *ACM SIGOPS Operating Syst. Rev.*, vol. 42, no. 6, pp. 36-42, Oct. 2008.

[14] S. Leutenegger and D. Dias, "A modeling study of the TPC-C benchmark," in *Proc. Management Data Conf.*, pp. 22-31, New York, NY, Jun. 1993.

[15] N. Agrawal, et al., "Design tradeoffs for SSD performance," in *Proc. USENIX Technical Conf.*, Berkeley, CA, Jun. 2008.

[16] S. Lim, S. Lee, and B. Moon, "FASTer FTL for enterprise-class flash memory SSDs," in *Proc. Storage Netw. Architecture and Parallel I/Os*, Ipp. 3-12, ncline Village, NV, Sept. 2010.

[17] R. Karedla, J. Love, and B. Wherry, "Caching strategies to improve disk system performance," *IEEE Comput.*, vol. 27, no. 3, pp. 38-46, Mar. 1994.

[18] *The DiskSim Simulation Environment Version 3.0 Reference Manual*, CMU, Jan. 2003.

[19] Y. Kim, B. Tauras, A. Gupta, and B. Urgaonkar, "FlashSim: A simulator for NAND flash-based solid-state drives," in *Proc. Advances in Syst. Simulation Conf.*, pp. 125-131, Porto, Portugal, Sept. 2009.

[20] *OLTP and Web Search Traces from Umass Trace Repository*, http://traces.cs.umass.edu/index.php/Storage/Storage.

[21] *Block I/O Traces from IOTTA Repository*, http://iotta.snia.org/tracetypes/3.

이 해 룡 (Hai-Long Li)

2013년 8월 : 합비공업대학교 전자공학과 졸업
2014년 9월~현재 : 서강대학교 전자공학과 CAD&ES 연구실 석사과정
<관심분야> Flash memory, Embedded System

황 선 영 (Sun-Young Hwang)

1976년 2월 : 서울대학교 전자공학과 학사
1978년 2월 : 한국과학원 전기 및 전자공학과 공학석사
1986년 10월 : 미국 Stanford 대학 전자공학 박사
1976년~1981년 : 삼성반도체(주) 연구원, 팀장
1986년~1989년 : Stanford 대학 Center for Integrated System 연구소 책임연구원 및 Fairchild Semiconductor Palo Alto Research Center 기술자문
1989년~1992 : 삼성전자(주) 반도체 기술 자문
1989년 3월~현재 : 서강대학교 전자공학과 교수
<관심분야> SoC설계 및 framework 구성, CAD시스템, Com. Architecture 및 DSP System Design 등