Journal of The Korea Society of Computer and Information
Vol. 22 No. 2, pp. 1-7, February 2017
www.ksci.re.kr
https://doi.org/10.9708/jksci.2017.22.02.001

Hot Data Identification For Flash Based Storage Systems Considering Continuous Write Operation

Seung-Woo Lee*, Kwan-Woo Ryu**

Abstract

Recently, NAND flash memory, which is used as a storage medium, is replacing HDD (Hard Disk Drive) at a high speed due to various advantages such as fast access speed, low power, and easy portability. In order to apply NAND flash memory to a computer system, a Flash Translation Layer (FTL) is indispensably required. FTL provides a number of features such as address mapping, garbage collection, wear leveling, and hot data identification. In particular, hot data identification is an algorithm that identifies specific pages where data updates frequently occur. Hot data identification helps to improve overall performance by identifying and managing hot data separately. MHF (Multi hash framework) technique, known as hot data identification technique, records the number of write operations in memory. The recorded value is evaluated and judged as hot data. However, the method of counting the number of times in a write request is not enough to judge a page as a hot data page. In this paper, we propose hot data identification which considers not only the number of write requests but also the persistence of write requests.

▸ Keyword : Flash Memory, FTL, Garbage Collection, Hot Data Identification

I. Introduction

지금까지 컴퓨터 시스템에 저장매체로 널리 사용되어온 하드디스크 드라이브(hard disk drive)는 점차 낸드 플래시 메모리 기반의 SSD(Solid State Drive)로 대체되고 있으며 이러한 SSD는 하드디스크 드라이브와 비교해 상대적으로 더 빠른 엑세스 속도, 저전력, 휴대성과 같은 여러 장점이 있다. 또한 최근 SSD의 가격하락으로 인하여 보급 속도는 지금보다 더욱더 빨라질 것으로 예상된다. 하지만 이러한 장점을 가진 SSD에 저장매체로 사용되는 낸드 플래시 메모리는 하드웨어 특성으로 인해 발생하는 몇 가지 문제점을 가지고 있다. 특히 데이터 업데이트 동작 시 제자리 덮어쓰기(in-place updates)의 불가능함으로 인해 발생하는 추가적인 쓰기 전 지우기(erase-before-write)동작은 낸드 플래시 메모리를 저장매체로 사용하는 SSD가 해결해야할 중요한 문제로 알려져 있다[1]. 이러한 문제를 해결하기 위해서는 데이터 업데이트 동작이 빈번히 발생하는 페이지(hot data page)와 상대적으로 그렇지 않은 페이지(clod data page)를 구분하여 각각 물리 블록에 매핑 저장함으로써 해결할 수 있으며 이러한 데이터가 기록된 페이지를 분류하는 기법을 hot data identification기법이라 한다. 현재까지 알려진 기법으로는 쓰기연산의 호출의 빈도를 기록하여 이를 기준으로 hot data를 판단하는 MHF(Multi Hash Function Framework)기법과 쓰기연산 요청에 대한 지속 여부를 판단하여 구분하는 기법 등이 있다[2][3][4].

MHF기법의 경우 특정 논리페이지주소(Logical Page Address)에 대한 쓰기연산 요청의 빈도를 별도에 메모리에 기록하고 그 기록된 값이 일정기준 이상일 때 해당 논리페이지를 hot data로 판단하는 방법이다. 하지만 이러한 쓰기연산 요청

• First Author: Seung-Woo Lee, Corresponding Author: Kwan-Woo Ryu
*Seung-Woo Lee (zpa007@knu.ac.kr), Dept. of Computer Science and Engineering, Kyungpook National University
**Kwan-Woo Ryu (kwryu@knu.ac.kr), Dept. of Computer Science and Engineering, Kyungpook National University
• Received: 2016. 11. 09, Revised: 2016. 12. 12, Accepted: 2017. 01. 19.

빈도를 단순히 카운트하는 방법만으로 hot data로 판단하는데 한계가 있다. 또한 쓰기연산 요청의 지속성을 기준으로 하는 기법의 경우 쓰기연산 요청 사실을 특정 시간 간격을 기준으로 순차적으로 기록하고 쓰기연산 요청 발생의 연속성을 평가하여 hot data로 판단하는 방법이다[5]. 이러한 지속성을 기준으로 하는 방법의 경우 그 구현과 운용이 복잡한 단점이 있다.

본 논문에서는 쓰기연산 요청의 빈도를 측정하고 측정된 결과 값을 사전에 미리 정한 시간단위 마다 별도에 메모리에 누적 기록함으로써 hot data 판단 시 쓰기연산 요청의 빈도와 지속성을 함께 고려하는 기법을 제안한다.

II. Preliminaries

2.1 Related works

2.1.1 FTL(Flash Translation Layer)

컴퓨터 시스템의 저장매체로 널리 사용되어 온 하드디스크는 데이터의 읽기, 쓰기의 작업 단위가 트랙과 섹터이며 오랜 기간 동안 이러한 구조의 하드디스크를 저장매체로 이용하여 컴퓨터 시스템은 발전하였다. 이러한 하드디스크를 저장매체로 사용하는 컴퓨터 시스템의 경우 응용프로그램과 운영체제 그리고 물리적인 하드디스크에 데이터 입출력을 처리하는 파일시스템까지 모두 트랙과 섹터 단위로 동작하므로 문제없이 동작할 수 있었다.

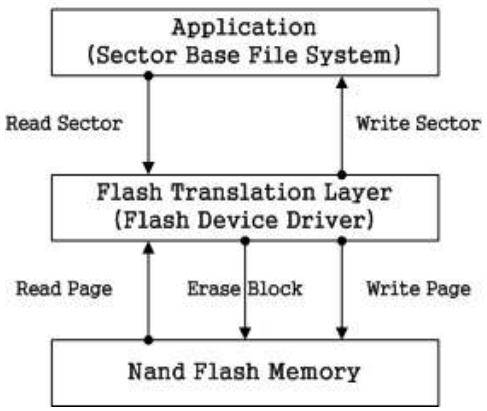

Fig. 1. Flash Memory-based Storage Systems

하지만 SSD에 저장매체로 사용되는 낸드 플래시 메모리는 물리적으로 트랙과 섹터가 존재하지 않으며 대신 블록과 페이지를 데이터 처리의 단위로 사용한다. 이러한 물리적인 구조를 가지는 낸드 플래시 메모리를 효과적으로 사용하기 위해서는 블록과 페이지 기반의 새로운 파일 시스템이 필요하다. 그러나 새로운 파일시스템을 개발하기보다 기존에 사용되어온 파일시스템을 수정 없이 섹터 기반 파일시스템과 페이지 기반의 낸드 플래시 메모리 사이에 플래시 변환 계층(Flash Translation Layer)을 만들어 논리적으로 섹터 구조를 구축하는 방법을 사용한다.

이러한 플래시 변환 계층은 물리적인 낸드 플래시 메모리의 하드웨어 특성을 고려하여 필수적으로 주소 매핑(Address Mapping), 가비지 컬렉션(Garbage Collection), 마모도 평준화(Wear leveling) 등 부가적인 다양한 기능들을 제공한다[6][7].

2.1.2 Features of nand flash structure and memory read, write, and delete operations

낸드 플래시 메모리의 셀들은 블록과 페이지 단위로 구성되며, 일반적으로 하나의 블록은 256개의 페이지로 구성되고 하나의 페이지는 4바이트 크기를 가지는 것이 일반적이다. 이러한 구성의 낸드 플래시 메모리는 데이터 읽기 연산과 쓰기 연산이 페이지 단위로 동작하는 특징이 있다. 특히 페이지에 대한 쓰기 연산에 경우 해당 페이지에 이미 데이터가 기록되어 있는 상태에서 데이터의 변경 즉 데이터 업데이트가 요청되었을 때 해당 페이지에 바로 덮어쓰기 되는 것이 아니라 데이터를 내부 레지스터로 복사하고 레지스터에서 변경한 후 비어있는 새로운 페이지에 쓰여 지게 된다. 그리고 데이터가 기록되어있는 이전 페이지는 무효(invalid) 상태로 마킹되며 이러한 페이지들이 점차 늘어나면 적절한 시점에 가비지 컬렉션에 의해 삭제 연산이 실행된다. 이와 같은 현상은 낸드 플래시 메모리의 물리적인 구조에 의해 발생하는 특징이다.

삭제 연산의 경우 블록 단위로 동작하기 때문에 단독으로 페이지 삭제가 불가능하며 페이지 삭제 시에는 해당 페이지가 속한 블록 전체를 삭제해야하는 특징이 있다. 앞서 언급한 것과 같이 낸드 플래시 메모리의 하드웨어 특성으로 인해 덮어쓰기가 불가능으로써 발생하는 무효 페이지의 수가 점차 증가할수록 상대적으로 낸드 플래시 저장매체의 저장 가능 공간은 점점 줄어들게 되며 이러한 문제를 해결하기 위해는 사전에 무효 페이지 발생을 최소화 할 수 있는 hot data identification 기법이 필수적으로 필요하다.

2.1.3 Hot data identification

hot data란 특정 페이지에 데이터 업데이트가 상대적으로 빈번히 발생하는 것을 뜻하며 상대적으로 그렇지 않은 경우를 cold data 라고 한다[8][9]. hot data와 cold data가 같은 블록 내에 저장되어있을 경우 해당 블록에 hot data 페이지에 빈번한 업데이트 동작 발생으로 인해 무효 페이지가 점차 증가하게 되고 결국 가비지 컬렉션 동작이 발생하게 된다. 이 때 cold data 페이지인 유효페이지의 데이터를 유지 하기위해 유효 페이지를 새로운 블록에 복사해야 하는 부가적인 쓰기 연산 요청이 발생하게 되는 것이다.

이러한 문제를 최소화하기 위해서는 쓰기 연산 요청의 패턴을 분석하여 hot data와 cold data를 미리 구분하고 각각 다른 블록에 저장함으로써 해결이 가능하다. 이러한 기법을 hot data identification 기법이라고 하며 이를 통해 가비지 컬렉션

동작 시 수행되는 블록 삭제 연산으로 인한 오버헤드를 최소화하여 전체적인 플래시 메모리의 성능을 향상시킬 수 있다. 현재 이와 관련된 많은 기법이 연구되고 있으며 그 중 대표적으로 멀티해시함수를 이용한 MHF기법이 있다. MHF 기법은 하나 이상의 해시함수를 이용하여 쓰기 연산이 요청된 페이지의 주소 값을 해싱 한다. 해싱의 결과 값은 쓰기연산 요청에 횟수를 기록하기 위한 카운트와 매칭되어 있다.

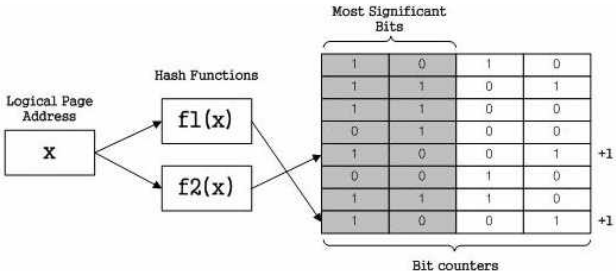

Fig. 2. Multi Hash Function Framework

Fig. 2.와 같이 플래시 변환 계층으로부터 특정 논리 페이지 주소(Logical Page Address)에 대한 쓰기연산 요청이 발생하면 해당 논리 페이지 주소의 값을 해싱하고 결과 값과 매칭된 특정 카운터에 bit값을 1bit씩 증가시킨다. 이후 카운터에 기록된 값 중 상위 2bit에 기록된 값이 0이 아닐 경우 해당 논리 페이지 주소에 기록된 데이터를 hot data로 판단하는 구조이다. 이러한 구조의 MHF 기법은 알고리즘의 단순성으로 인해 운영의 오버헤드가 낮은 장점이 있지만 hot data 판단 시 쓰기연산 요청의 빈도만을 고려함으로써 특정시간 구간에 집중적인 쓰기연산 요청이 발생한 뒤 더 이상 쓰기연산 요청이 발생하지 않는 경우까지 hot data로 판단하는 문제가 있다.

III. The Proposed Scheme

3.1 The Framework

본 논문에서 제안한 hot data identification의 구조는 Fig. 3.과 같다. 플래시 변환 계층으로부터 쓰기연산 요청이 발생할 경우 해당 논리 페이지 주소 값을 해싱하기 위한 해시 함수들과 그 결과 값과 대응하는 4bit 크기의 메모리, 그리고 이러한 다수의 메모리들로 이루어진 하나의 전체 테이블로 구성된다.

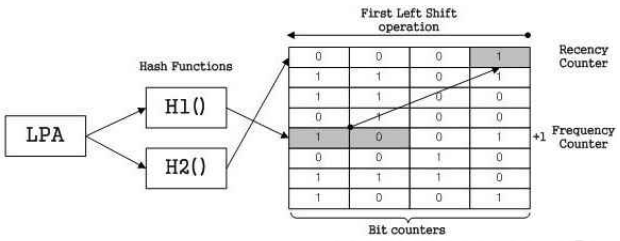

Fig. 3. Our Framework and It's Operations

기본적인 동작구조는 다음과 같다. 먼저 쓰기연산 요청에 대한 빈도를 측정하기 위해 해시 함수 H1을 이용하여 논리 페이지 주소 값을 해싱하고 그 결과 값과 대응하는 4bit 크기의 메모리에 비트 값을 1bit 증가시킨다. 또한 쓰기연산 요청에 대한 지속성을 측정하기 위해 해시 함수 H2를 이용하여 동일한 논리 페이지 주소 값을 해싱하고 그 결과 값과 대응하는 4bit 메모리에 지속성을 측정하기 위한 기록을 시작한다.

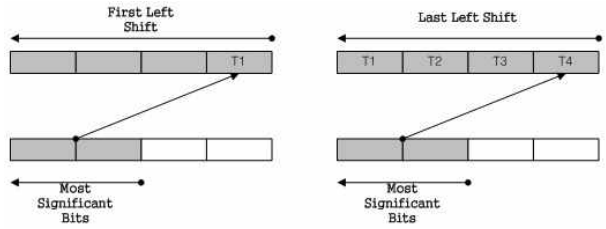

Fig. 4. Operation process when Left Shift

쓰기연산 요청에 지속성을 측정하기 위해 사용되는 4bit 메모리는 최초 대기 상태이며, 기록이 시작되는 시점에 동작상태로 전환된다. 대기상태에서 동작상태로 전환 된 4bit 메모리는 사전에 미리 정해둔 시간 간격으로 총 4차례 left shift하며 Fig. 4.와 같이 left shift 시 마다 해시함수 H1과 대응하는 카운트에 기록된 빈도 측정값을 참조하여 빈도 측정 카운터의 4bit 중 상위 2bit의 값이 0이 아닌 경우 1 반대의 경우에는 0을 기록한다. 동작상태 중인 4bit 카운터는 4차례 left shift 이후 다시 대기상태로 전환되며 이때 해당 논리 페이지 주소에 대한 hot data 여부를 판단하는 과정이 시작된다.

3.2 Frequency of write operations

본 논문에서는 특정 논리 페이지 주소에 대한 쓰기연산 발생의 빈도를 측정하기 위해 Algorithm 1.과 같이 쓰기연산 요청이 발생한 논리 페이지 주소를 해시함수를 이용하여 해싱하고 그 결과 값과 매칭되는 4bit 메모리에 값을 1bit 증가시킨다. 또한 4bit 메모리에 기록된 값은 시간이 지남에 따라 초기화 되어 4bit 모두 0값으로 설정된다.

Algorithm 1. *Frequency of write operation(lpn).*

```
Input: lpn
Output: none
1 k := the number of hash functions
2 threshold := threshold value for hot data identification
3 for i=1 to k
4    entry=hashing[i] (lpn)
5    increase hashing_table[entry] by 1
6 end for
end
```

이러한 동작 과정은 기존 MHF기법에서 알려진 빈도측정 알고리즘과 유사하다. 하지만 본 논문에서 제안한 빈도 측정 알고

리즘의 경우 MHF기법에서 발생하는 문제점인 쓰기연산 요청의 빈도가 기록된 카운터의 초기화 과정에 불명확성으로 인해 발생하는 hot data 판단오류 문제를 해결하기 위해 4bit에 기록된 단순 카운터의 값뿐만 아니라 해당 논리 페이지 주소에 대한 쓰기연산 요청의 지속성을 측정하는 알고리즘의 결과를 함께 고려하여 최종 hot data로 판단한다.

쓰기연산 요청의 지속성을 함께 고려하는 이유는 쓰기연산 요청에 횟수를 단순히 카운트하는 방법만으로는 해당 논리 페이지 주소를 hot data로 정확히 판단하는 데 충분하지 않기 때문이다. 예를 들어 특정 시간 구간에 집중적인 쓰기연산 요청이 발생한 뒤 더 이상 쓰기연산 요청이 발생하지 않는 쓰기요청 패턴을 생각할 수 있다. 이러한 경우 MHF기법은 단순히 카운터에 기록된 쓰기연산 요청에 대한 빈도 값만을 기준으로 해당 논리 페이지를 hot data로 판단하지만 실제로 해당 논리 페이지는 이후 더 이상 쓰기연산이 발생하지 않는 cold data 임에도 불구하고 hot data로 판단되는 문제가 발생한다.

이러한 문제는 쓰기연산 요청 발생에 대한 카운트 값의 기록 시점과 그 기록된 값의 초기화 시점이 명확하지 않음으로 발생함으로 이러한 문제점을 해결하기 위해 본 논문에서는 쓰기연산 요청의 발생 빈도를 카운터하고 그 결과를 사전에 미리 정한 시간단위로 별도의 메모리에 누적 기록하여 이를 바탕으로 hot data 여부를 판단하는 알고리즘을 제안한다.

3.3 Duration measurement for write operation

Algorithm 2. *Duration Counter(lpn).*

```
Input: lpn
Output:none
1 k := the number of hash functions
2 threshold := threshold value for hot data identification
3 for i=1 to k
4    entry=hashing[i] (lpn)
5    if idel state is 0
6       for times during a certain time
           the most signification bit of Frequency counter
           are not 0
7          increase hashing_table[entry] by 1
        end for
8    end if
9 end for
end
```

특정 논리 페이지를 hot data로 정확히 판단하기 위해서는 특정 논리 페이지에 대한 쓰기연산 요청의 빈도와 지속적인 쓰기연산 요청을 함께 고려하여야한다. 이를 위하여 쓰기연산 요청에 대한 빈도 값이 기록된 카운터 값의 초기화 여부와 관계없이 특정 시간 단위 마다 쓰기연산 요청의 빈도 측정값을 별도에 메모리에 누적 기록하여 쓰기연산 요청의 지속성 여부를 판단한다.

본 논문에서 쓰기연산 요청의 지속성을 측정하기 위하여 Algorithm 2.와 같이 쓰기연산 요청 발생 시 최초 대기상태에 4bit 카운터가 동작상태로 전환되며 4bit에 마지막 bit 값을 0으로 초기화 시킨 후 해당 쓰기연산 요청에 대한 지속성 값을 bit 값으로 기록하게 된다. 이때 기록되는 값은 해시함수 H1과 대응하는 4bit 메모리에 상위 2bit의 값이 0이 아니면 1로 기록하고 그 반대의 경우에는 0을 기록하게 된다. 이후 4차례 left shift 동작 마다 이를 반복하게 된다. 이후 hot data page 여부를 판단하기 위한 hot data 판단과정을 시작한다.

3.4 Hot data judgment process

특정 논리 페이지에 대한 hot data 판단은 쓰기연산 요청에 대한 빈도 측정값을 특정 시간마다 누적 기록한 뒤 그 기록된 값의 지속성을 평가함으로써 이루어진다. 특히 누적 기록한 각 bit의 값은 그 값의 기록 시점을 기준으로 지속성 측면에서 서로 다른 의미를 가진다. 본 논문에서는 쓰기연산 요청에 대한 지속성을 판단하기 위해 4bit 메모리에 상위2bit와 하위2bit의 값이 모두 1일 때 hot data로 판단한다.

Fig. 5.와 같이 빈도 측정값을 누적 기록하기 위해 각 bit를 T1, T2, T3, T4로 구분하였다. T1의 기록된 값은 T2의 기록된 값보다 지속성 측면에서 더 높은 가중치를 가진다. 왜냐하면 지속성이란 얼마나 더 쓰기연산 요청이 오래 지속되었는지를 뜻하기 때문이며 상대적으로 더 이전에 빈도 측정값이 기록된 T1의 값은 T2의 값보다 쓰기연산 요청을 기록한 시점이 더 앞서기 때문이다.

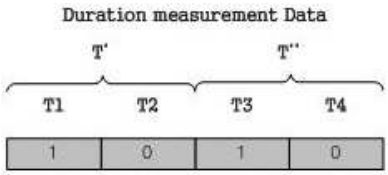

Fig. 5. Duration measurement data

또한 상위2bit와 하위2bit를 구분하여 평가함으로써 쓰기연산 요청의 지속성을 판단할 수 있다. 이를 위해 Fig. 5.와 같이 T1과 T2를 하나의 연속적인 시간 구간으로 설정하고 T3와 T4를 하나의 연속적인 시간 구간으로 설정한다.

hot data 판단과정은 Algorithm 3. 과 같다.

Algorithm 3. Hot data judgment process *(lpn).*

```
Input: Duration Counter Value
Output: none
1 threshold := if each 2bit balue is 1
2 if Duration measurment Data ( 't1' || 't2' is 1
          && 't3' || 't4' is 1 )
          update hot data list
3 end if
end
```

해시 함수 H2에 대응하는 4bit 메모리에 기록된 쓰기연산

요청에 대한 빈도 측정값이 누적 기록된 상위 2bit의 값이 0이 아니고 하위 2bit에 값이 0이 아닐 경우 해당 쓰기연산이 요청된 논리 페이지를 hot data로 판단한다. 예를 들어 1 1 1 1 의 패턴의 경우 T1과 T2 시간 구간과 T3와 T4 시간 구간에 연속하여 쓰기요청이 기록되었으므로 hot data로 판단하며 1 0 0 1 의 패턴에 경우도 연속되는 구간에서 모두 쓰기요청이 발생하였다고 판단할 수 있다. 반면 1 1 0 0 의 패턴에 경우 hot data로 판단하지 않는다. 이유는 연속되는 시간구간 중 한 구간에서만 쓰기연산 요청이 발생하였기 때문이며 이는 지속성 측면에서 쓰기연산 요청이 지속적으로 발생하지 않음을 의미하기 때문이다. 마지막으로 1 1 0 0 쓰기 패턴의 경우 T' 구간을 기준으로 쓰기연산 요청이 발생하였고 이후 쓰기연산 요청이 지속적으로 발생하지 않는 쓰기패턴으로 판단할 수 있다.
0 0 1 1 패턴에 경우도 마찬가지로 hot data로 판단하지 않는다. 본 논문에서 제안한 알고리즘의 경우 특정 lpn에 대한 hot data 판단 과정에서 발생하는 연산량이 상대적으로 MHF기법보다 더 많이 요구된다. 특히 쓰기연산 요청에 대한 빈도 값을 누적 기록하는 과정에서 발생하는 오버헤드를 생각할 수 있다. 하지만 이것으로 인해 발생하는 오버헤드는 hot data를 정확히 판단하지 못함으로 발생하는 추가적인 가비지 컬렉션 동작오버헤드에 비해 훨씬 낮다. 결론적으로 이러한 오버헤드 문제는 hot data identification 기법을 적용하는 시스템의 특성을 고려한 최종 사용자 조정 작업에 더 많은 노력이 필요하다.

IV. Experimental

이 장에서는 2장에서 소개한 MHF기법과 3장에서 제안한 기법을 각각 시뮬레이션 프로그램으로 구현하고 일반적인 pc사용환경에서 발생하는 workload를 기록한 trace파일을 대상으로 쓰기연산 요청에 대한 hot data 발생 비율과 블록 내 무효페이지 발생 비율에 대해 실험하여 그 결과를 분석한다.

4.1 Experimental environment and Trace file spec

Table 1. Trace file characteristics

속성	값
write 요청 횟수	290,400
최소 오프셋	8
최대 오프셋	326,960
최소 크기	32
최대 크기	4,096

실험에 사용된 시스템은 인텔 i5-4690, ddr3 4GB 메모리로 구성되며 Ubuntu 14.04를 운영체제로 사용한다. 또한 실험에 사용된 trace파일의 경우 일반적인 컴퓨터 사용 환경에서 발생하는 workload를 trace파일로 사용하였으며 표. 1.과 같이 쓰기연산 요청횟수는 약 290,000회 이다. 또한 실험에서 쓰기연산 요청 이외에 연산에 대해서는 고려하지 않는다.

4.2 Analysis of experimental results

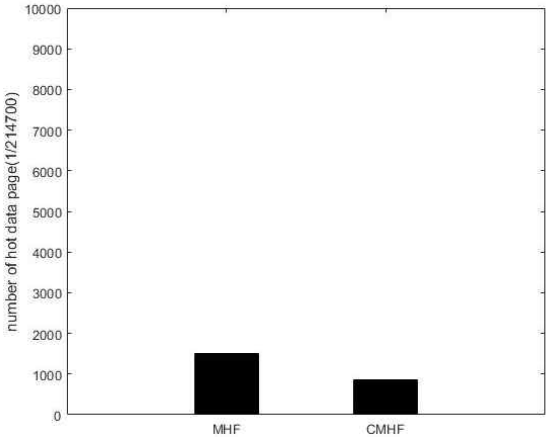

Fig. 6. Number of hot data pages when applying MHF technique and CMHF

실험결과 MHF기법 적용 시 전체 쓰기연산 요청 페이지 중 hot data로 판단된 페이지의 비율은 전체 쓰기연산 요청페이지 중 약 0.7%이고 제안한 기법 적용 시 약 0.4%이다. MHF기법보다 상대적으로 제안한 기법 적용 시 hot data 판단 비율이 낮은 이유는 특정시간 구간에 집중적인 쓰기연산 요청이 발생하고 이후 더 이상 쓰기연산 요청이 발생하지 않는 경우를 MHF기법과는 다르게 hot data로 판단하지 않기 때문이다.

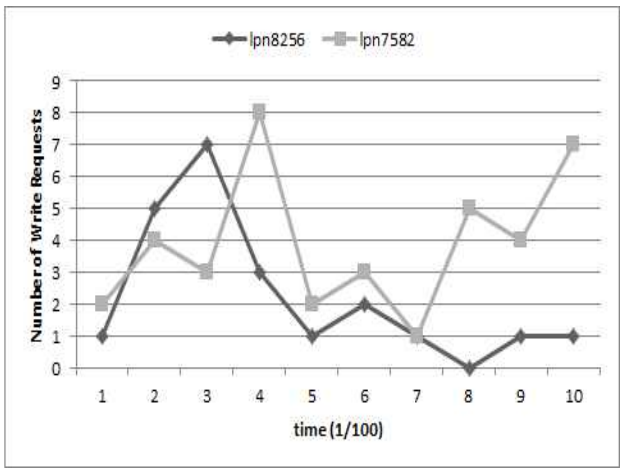

Fig. 7. Number of hot data pages when applying MHF technique and CMHF

Fig. 7.은 특정시간 구간에서 발생한 쓰기연산 요청 과정을 나타낸다. lpn 8256번에 경우 특정시간 구간에 집중적인 쓰기연산 요청이 발생하고 이후 더 이상 쓰기연산 요청이 카운터의 상위2bit에 기록될 만큼 발생하지 않음을 알 수 있다. 이러한 쓰기연산 요청의 경우 MHF기법은 단순히 쓰기연산 요청의 횟수를 카운트한 결과만을 기준으로 hot data를 판단함으로 이후 더 이상의 쓰기연산 요청이 발생하지 않는 해당 페이지를 hot

data로 판단한다. 반면 lpn 7582번에 경우 특정시간 동안 지속적인 쓰기연산 요청이 발생하고 있다. 이러한 쓰기연산 요청 페이지를 hot data로 판단하기 위해서는 쓰기연산 요청을 기록한 값이 초기화되기 이전에 별도에 메모리에 누적 기록하여 이를 평가함으로 가능하다.

4.2.1 Rate of invalid page occurrence

Fig. 8.은 MHF기법과 제안기법을 각각 적용했을 때 블록별 지우기 횟수를 나타낸다. MHF기법 적용 시 약 4340회이고 제안기법 적용 시 약 3942회이다. 제안기법을 통해 전체 쓰기연산 요청 페이지에 대한 hot data판단 후 hot data page와 cold data page를 각각 블록에 저장함으로 인해 블록 내 무효화 페이지 발생 수가 감소하였으며 이로 인해 블록에 대한 가비지 컬렉션 동작이 상대적으로 적게 발생하였음을 알 수 있다.

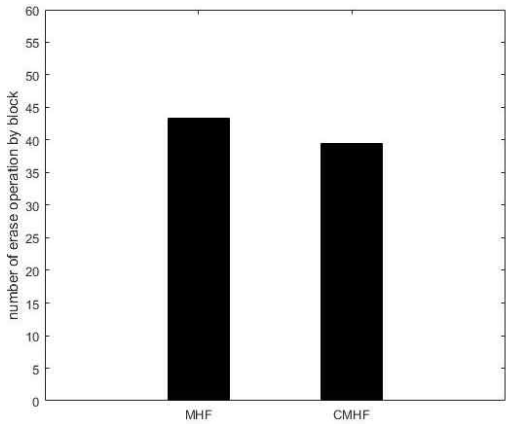

Fig. 8. number of eraser operation by block

반면 MHF기법에 경우 hot data로 판단한 페이지가 제안 기법 적용 시 보다 더 많았지만 오히려 블록 내 무효페이지 발생 비율이 더 높게 발생하였으며 이로 인한 가비지 컬렉션 동작이 더 많이 요구되었음을 알 수 있다. 이는 MHF기법이 판단한 hot data가 실제 쓰기연산 요청을 많이 요구하는 hot data가 아니었음을 의미한다.

V. Conclusions

본 논문에서는 기존 hot data identification기법으로 MHF기법에 대해 소개하였으며 MHF기법의 경우 hot data 판단 시 쓰기연산 요청에 대한 지속성을 고려하지 않음을 언급하였다. 결과적으로 제안기법과 비교해 hot data 판단 수는 더 많았지만 실제 블록별 지우기 연산 발생 횟수는 오히려 더 많이 발생하였음을 실험을 통해 알 수 있었다. 제안기법의 경우 쓰기연산 요청에 대한 지속성을 측정하기 위해 별도에 메모리를 할당하여 측정된 빈도 값을 특정시간 구간 동안 누적 기록함으로써 이를 hot data 판단에 활용하였다. 결과적으로 MHF기법과 비교해 hot data판단 수가 적음에도 블록 내 지우기 연산 요청횟수는 상대적으로 적게 발생함을 확인하였으며 실험을 통해 지속적인 쓰기요청이 발생하는 쓰기패턴의 페이지가 상당수 존재함을 확인할 수 있었다. 향후연구 과제로 본 논문에서 제안한 hot data identification기법을 적용한 완성된 낸드 플래시 매핑 알고리즘에 대한 연구를 진행할 계획이다.

REFERENCES

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

[2] Jun Liu, Shuyu Chen, Tianshu Wu, Hancui Zhang, "A Novel Hot Data Identification Mechanism for NAND Flash Memory," IEEE Journals & Magazines, Volume: 61, Issue: 4 pp.463-469, 2015.

[3] Jen-Wei Hsieh, Tei-Wei Kuo, Li-Pin Chang, "Efficient identification of hot data for flash memory storage systems", ACM Transactions on Storage (TOS), Volume 2 Issue 1, February 2006.

[4] Hyun-Seob Lee, Hyun-Sik Yun, and Dong-Ho Lee, "HFTL:Hybrid Flash Translation Layer based on Hot Data Identification for Flach Memory", IEEE Journals & Magazines, 2009.

[5] Dongchul Park; David H. C. Du, "Hot data identification for flash-based storage systems using multiple bloom filters", 2011 IEEE 27th Symposium on Mass Storage Systems and Technologies (MSST), 2011.

[6] L.-P. Chang, "On efficient wear-leveling for largescale flash-memory storage systems," Proc. of the 2007 ACM symposium on Applied computing, pp.1126-11 30, 2007.

[7] 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.

[8] Sanghyuk Jung; Yangsup Lee; Yong Ho Song, "A process-aware hot/cold identification scheme for flash memory storage systems," IEEE Journals & Magazines, Volume: 56, Issue: 2 pp.339-347, 2010.

[9] Kyuwoon Kim; Sanghyuk Jung; Yong Ho Song, "Compression ratio based hot/cold data identification for flash memory," IEEE Conference Publications, pp.33-34, 2011.

Authors

Seung Woo Lee received the B.S. degree from Kyungil University and M.S. degree from Kyungpook National University, South Korea, in 2010 and 2013, respectively.

He is currently enrolled for PhD degree in degital media lab. His current interests include embedded and flash memory based storage system.

Kwan Woo Ryu received the B.S. degree from Kyungpook National University. He received M.S. degree from KAIST, and PhD degree in University of MARYLAND, USA, in 1980, 1982, 1990, respectively.

He is currently an professor in school of computer science and engineering of Kyungpook National University. His research interests include multi-paradigm algorithm, parallel computing.