Journal of The Korea Society of Computer and Information
Vol. 24 No. 6, pp. 1-9, June 2019

www.ksci.re.kr
https://doi.org/10.9708/jksci.2019.24.06.001

Hot Data Verification Method Considering Continuity and Frequency of Write Requests Using Counting Filter

Seung-Woo Lee*, Kwan-Woo Ryu*

Abstract

Hard disks, which have long been used as secondary storage in computing systems, are increasingly being replaced by solid state drives (SSDs), due to their relatively fast data input / output speeds and small, light weight. SSDs that use NAND flash memory as a storage medium are significantly different from hard disks in terms of physical operation and internal operation. In particular, there is a feature that data overwrite can not be performed, which causes erase operation before writing. In order to solve this problem, a hot data for frequently updating a data for a specific page is distinguished from a cold data for a relatively non-hot data. Hot data identification helps to improve overall performance by identifying and managing hot data separately. Among the various hot data identification methods known so far, there is a technique of recording consecutive write requests by using a Bloom filter and judging the values by hot data. However, the Bloom filter technique has a problem that a new bit array must be generated every time a set of items is changed. In addition, since it is judged based on a continuous write request, it is possible to make a wrong judgment. In this paper, we propose a method using a counting filter for accurate hot data verification. The proposed method examines consecutive write requests. It also records the number of times consecutive write requests occur. The proposed method enables more accurate hot data verification.

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

I. Introduction

모바일 컴퓨팅 분야는 크게 성장 중에 있으며 스마트 폰, 스마트 워치, 디지털 카메라 등과 같은 모바일 컴퓨팅 기기의 수요는 이미 전통적인 데스크탑 컴퓨팅 기기의 수요를 넘어선 것으로 알려졌다[1]. 이러한 모바일 컴퓨팅 분야에서 사용되는 기기는 필수적으로 전력소모가 적고, 가볍고, 충격에 강해야 하며, 특히 사용자들이 만들어 내는 많은 양의 데이터를 효율적으로 저장, 관리할 수 있는 저장장치가 필수적으로 요구된다.

최근 모바일 컴퓨팅 기기에 저장장치로 널리 사용되고 있는 NAND 플래시 메모리 기반 저장장치는 데스크탑 컴퓨팅 분야에서 저장장치로 사용되어온 하드디스크 보다 더 모바일 컴퓨팅 환경에 적합하다. 대표적인 낸드 플래시 메모리 기반 저장장치로는 eMMC(embedded multimedia card)와 SD(Secure Digital), 그리고 솔리드 스테이트 드라이브(solid-state drive) 등이 있다. 낸드 플래시 메모리는 플로팅 게이트 트랜지스터를 이용하여 데이터를 기록하며 이는 페이지와 블록 단위로 구성된다. 또한 데이터 쓰기와 읽기 명령은 페이지 단위, 데이터 삭제 명령은 블록 단위로 동작하는 특징이 있다. 특히 낸드 플래시 메모리는 하드웨어 구조로 인해 데이터 제자리 덮어쓰기(in-place updates)가 불가능함으로 데이터 업데이트 요청 시 쓰기 전 지우기(erase-before-write) 동작이 먼저 요구된다. 앞서 언급한 것처럼 데이터 삭제 명령은 블록단위로 동작함으로 데이터 업데이트 요청 시 업데이트 된 페이지

• First Author: Seung-Woo Lee, Corresponding Author: Kwan-Woo Ryu
*Seung-Woo Lee (zpa007@knu.ac.kr), Dept. of Computer Science, Kyungpook National University
*Kwan-Woo Ryu (kwryu@knu.ac.kr), Dept. of Computer Science, Kyungpook National University
• Received: 2019. 03. 18, Revised: 2019. 04. 22, Accepted: 2019. 06. 05.

를 포함한 블록 내의 모든 페이지를 비어있는 새로운 블록에 복사해야하는 오버헤드가 발생하며 저장장치 내에 비어있는 블록의 수가 줄어들수록 쓰기 전 지우기 동작으로 인한 오버헤드는 증가하게 된다. 이러한 문제는 낸드 플래시 메모리 기반 저장장치 사용에 있어 반드시 고려해야할 중요한 문제로 알려져 있다[2].

이러한 문제를 해결하기 위해서는 특정 페이지를 대상으로 발생한 쓰기요청 중 데이터 업데이트 동작이 빈번히 발생하는 쓰기요청(hot data)과 상대적으로 그렇지 않은 쓰기요청(cold data)을 정확히 구분하여 각각 블록에 저장, 관리함으로 해결할 수 있다. 이러한 쓰기요청 분류기법을 hot data identification 기법이라 하며 현재까지 다양한 hot data identification 기법이 알려졌다.

그 중 가장 단순한 기법은 단순 비트배열과 해시함수를 이용하여 특정 시간동안 발생한 쓰기요청 발생 사실을 단순 누적 기록하고 그 기록 값이 특정기준 이상일 경우 이를 hot data로 판단하는 MHF(Multi Hash Function Framework)기법이다. MHF 기법은 알고리즘의 단순성으로 인해 운용에 장점이 있으나 쓰기요청 발생 사실의 단순 기록만으로 hot data 여부를 판단함으로 특정 시간동안에 한차례 발생한 집중적인 쓰기요청을 hot data로 판단하는 문제가 있다.

또 다른 기법으로 쓰기요청 발생 사실을 확률기반 자료구조인 블룸필터를 이용하여 연속적으로 기록하고 각 블룸필터에 기록된 값의 연속성을 판단 기준으로 하는 BHF(Bloom filter Hash Function Framework)기법이다. 하지만 BHF 기법은 쓰기요청 발생 시 마다 이를 기록하기 위해 새로운 블룸필터를 생성해야함으로 메모리 사용량 증가의 문제가 있으며 항목삭제가 불가능한 단점이 있다. 또한 쓰기요청 발생의 연속성만을 기준으로 hot data 여부를 판단함으로 연속성이 짧은 특정 쓰기요청 패턴이 자주 발생하는 경우 이를 hot data로 판단하지 못하는 문제점이 있다[3-5].

본 논문에서 보다 정확한 hot data 검증을 위해 쓰기요청 발생 사실을 카운팅 필터를 이용하여 연속적으로 기록하며 또한 하나의 완료된 특정 쓰기요청 패턴이 특정 시간구간에 얼마나 자주 발생하는지를 카운터 값으로 함께 기록함으로 hot data 판단 시 쓰기요청의 연속성과 특정 쓰기요청 패턴의 발생 횟수를 모두 고려한 보다 정확한 hot data identification 기법을 제안한다.

II. Preliminaries

1. Related works

1.1 FTL(Flash Translation Layer)

하드디스크와 낸드 플래시 메모리 기반 저장장치는 서로 물리적인 동작방식과 그 운영방식이 전혀 다르다. 하드디스크는 데이터 저장을 위해 자기장의 원리를 이용한 플래터를 사용하며 데이터 읽기 쓰기 작업 단위로 트랙과 섹터 개념을 이용한다. 반면 낸드 플래시 메모리는 데이터 저장을 위해 전자적인 플로팅 게이트를 사용하며 데이터 읽기, 쓰기 작업 단위로 페이지와 블록 개념을 사용한다.

이러한 낸드 플래시 메모리 사용에 있어 반드시 고려해야할 점은 그동안 하드디스크를 저장장치로 이용하며 발전한 데스크탑 컴퓨팅 시스템에서 사용되어온 대부분의 운영체제 및 데이터 입출력을 담당하는 파일시스템들이 트랙과 섹터 단위로 동작한다는 것이다. 이러한 운영체제 및 파일시스템을 별다른 수정 없이 동작구조가 전혀 다른 낸드 플래시 메모리와 함께 사용하기 위해서는 기존 물리적인 섹터 기반 파일시스템과 낸드 플래시 메모리 사이에 논리적인 섹터 구조를 구축하는 방법을 이용한다. 이를 FTL(Flash Translation Layer)이라고 한다.

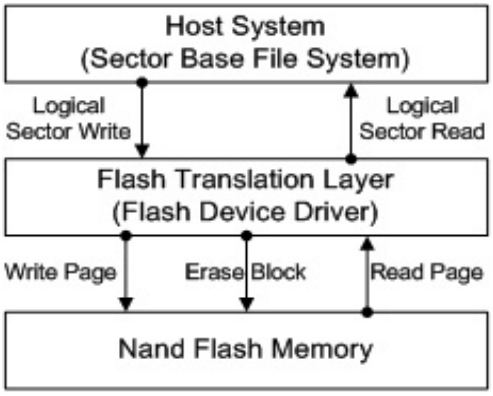

Fig. 1. Flash Memory based Storage System

FTL은 파일시스템에 논리적 주소를 낸드 플래시 메모리의 물리적 주소로 연결하기 위한 주소 매핑(Address Mapping), 데이터 업데이트 과정에서 발생하는 무효화된 페이지를 포함하는 블록을 선택하여 유효한 페이지들을 다른 블록에 복사한 후에 해당 블록을 삭제하여 재사용 할 수 있도록 해주는 가비지 컬렉션(Garbage Collection), 마모도 평준화(Wear leveling) 등 부가적인 다양한 기능들을 함께 제공한다[6].

1.2 NAND flash structure and memory read, write and delete operations

플래시 메모리에서 데이터를 저장하는 최소 단위는 셀(Cell)이다. 셀은 플로팅 게이트 트랜지스터로 구현되며 컨트롤 게이트와 산화막을 이용하여 플로팅 게이트의 전자를 채우거나 비우는 방식으로 데이터를 기록한다. 이러한 셀들이 병렬 구조로 연결된 것을 NOR 플래시 메모리 구조라 하며 직렬 구조로 연결된 것을 NAND 플래시 메모리 구조라 한다. 일반적으로 모바일 기기의 저장장치로 구조가 단순하고 고집적화에 유리한 낸드 플래시 메모리가 사용된다. 이러한 낸드 플래시 메모리의 셀들은 일정한 단위로 페이지와 블록 단위로 그룹화 된다. 예를 들어 삼성전자 SSD 840 EVO의 경우 2048KB의 블록 크기를 가지고 하나의 블록은 256개의 8KB

크기 페이지들로 구성된다. 또한 낸드 플래시 메모리는 페이지 단위로 데이터 읽기 또는 쓰기 작업을 실행하며 블록단위로 데이터 삭제 작업을 실행하는 특징이 있다.

이러한 낸드 플래시 메모리 사용에 있어 반드시 고려할 점은 이미 데이터를 기록한 페이지를 대상으로 데이터 업데이트 요청이 발생할 경우 플로팅 게이트의 이미 채워져 있는 전자를 비우고 다시 채우는 과정을 거쳐야 함으로 구조적으로 데이터 덮어쓰기(Overwrite)가 불가능하다는 것이다. 이러한 데이터 업데이트 요청은 해당 페이지에 기록된 데이터를 내부 레지스터로 읽어온 뒤 해당 레지스터에서 업데이트 요청된 데이터와 기존 데이터를 병합한 후 새로운 페이지에 쓰기 작업이 진행되며 기존 페이지는 무효 처리된다.

특정 페이지를 대상으로 한 빈번한 데이터 업데이트 요청은 결국 지속적으로 다수의 무효 페이지를 발생시키며 이러한 무효 페이지가 여러 블록에 흩어져 발생하는 것은 바람직하지 않다. 왜냐하면 쓰기 전 지우기 동작 과정 또는 가비지 컬렉션 과정 중 블록 삭제가 발생할 때 블록 내에 존재하는 적은 수의 무효페이지로 인해 다수의 유효 페이지를 새로운 블록으로 옮겨 써야하는 추가 작업이 발생하기 때문이다. 이러한 문제를 해결하기 위해는 특정 페이지를 대상으로 빈번히 발생하는 쓰기요청(hot data)을 정확히 구분하여 이 때 발생하는 무효페이지를 특정 블록에 집중적으로 기록함으로써 hot data 와 cold data를 별도의 블록에 저장 관리할 수 있으며 이를 위해 반드시 정확한 hot data identification 기법이 필수적으로 요구된다.

1.3 Bloom Filter와 Counting Filter

블룸필터(Bloom filter)란 u개 요소를 갖는 집합 U에서 s개의 요소로 이루어진 부분집합 S의 요소별 키 값을 표현하기 위해 사용되는 확률 기반 자료구조이며 이는 1970년 Burton Howard Bloom에 의해 발표되었다[7-8].

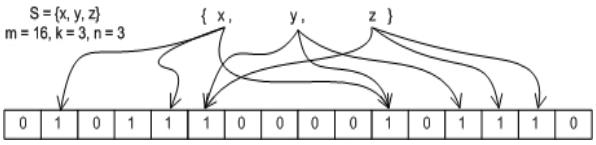

Fig. 2. Bloom filter

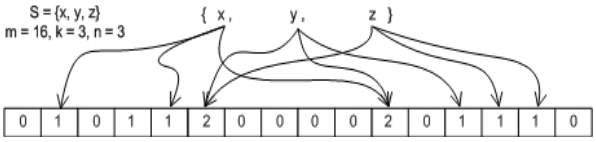

Fig. 3. Counting filter

블룸필터의 동작과정은 다음과 같다. 최초 블룸필터는 m크기의 1차원 비트배열이며 모든 비트는 0으로 설정된다. 또한 k개의 서로 다른 해시 함수가 존재하며 각 해시 함수는 부분집합 S에 속하는 요소를 입력 값으로 균등 확률 값을 반환하고 이 반환 값에 해당하는 1차원 비트배열의 비트 값을 0에서 1로 변경한다. 이후 집합 U에 포함된 요소들 중 부분집합 S에도 포함되는 요소인지 검사하는 과정은 k개의 해시 함수의 입력 값으로 해당 요소를 전달하고 그 반환 값에 해당하는 블룸필터의 비트 값이 모두 1이라면 이는 해당 요소가 부분집합 S에도 속한다고 판단 할 수 있다. 이러한 블룸필터는 부분집합 S에 요소를 추가하는 명령어와 특정 요소가 부분집합 S에 속하는지 검사하는 명령어는 지원하지만 항목에서 특정 요소를 삭제하는 것은 지원하지 않는다. 특히 특정 요소가 부분집합에 속한다고 판단되더라도 실제로는 특정 요소가 부분집합에 속하지 않을 수 있는데 이것을 긍정오류라 하며 반대로 특정 요소가 부분집합에 속하지 않는 것으로 판단되었지만 실제로는 특정 요소가 부분집합에 속할 수 있는데 이를 부정오류라 하는데 블룸필터에서는 부정오류는 절대로 발생하지 않는다는 특징이 있다.

한편 블룸필터 활용 시에 긍정오류 발생확률을 줄이기 위해 비트배열의 크기를 적어도 부분집합에 요소개수의 15배 크기로 설정하고 더불어 해시함수의 개수 또한 3개 이상으로 설정할 경우 긍정오류 발생 확률은 전체 1% 미만인 것으로 알려져 있다. 아래 표1.은 비트백터 크기와 해시함수 개수에 따른 긍정오류 비율을 나타낸다.

Table 1. Rate of positive error according to bit vector size and number of hash functions

m/n	k=1	k=2	k=3
12.0	7.99%	2.36%	1.08%
13.0	7.40%	2.03%	0.86%
14.0	6.89%	1.77%	0.72%
15.0	6.45%	1.56%	0.60%

이러한 블룸필터는 빠른 속도로 특정 요소의 부분집합 포함 여부를 조사할 수 있는 방법이다. 하지만 블룸필터는 부분집합의 항목을 수정할 수 없다는 단점이 존재한다. 또한 항목 제거가 불가능함으로 항목집합 S가 동적으로 변화하는 경우에는 항목이 변경될 때마다 새로운 비트배열을 생성해야함으로 이러한 방식은 현실적으로 사용하기 어렵다. 이러한 블룸필터의 단점을 극복한 것이 카운팅 블룸필터이다.

카운팅 필터(Counting filter)는 1998년 L. Fan 등에 의해 제안되었으며 항목 변경 시 필터를 재생성해야 하는 블룸 필터의 단점을 극복하기 위해 단일비트를 사용하는 블룸필터와는 다르게 이를 n비트로 확장하여 카운터로 사용하며 항목 추가 시 해당 n비트의 값을 1씩 증가시키고 항목 삭제 시 값을 1씩 감소시킨다. 또한 항목 검색 시 해당 n비트의 값이 0이 아닌지 확인하게 된다[9-10]. 이러한 동작방식의 카운팅 필터는 항목 삽입과 삭제에 대해 자유로우며 카운터 값을 저장하는 n비트를 이용하여 항목 삽입 삭제에 대한 사실을 표현할 수 있게 된다.

본 논문에서는 hot data identification을 위해 전체 LPN 중 hot data로 판정된 LPN을 표현하기 위해 카운팅 필터를 이용한다. 또한 hot data 판정 역시 카운팅 필터의 n비트를 이용하여 hot data 기록과 판정을 동시에 진행함으로써 효과적인 메모리

절약과 hot data 판정과정의 단순성을 동시에 확보하였다.

1.4 Problems of existing hot data identification technique

FTL은 하드웨어 시스템 큐에 삽입된 읽기 또는 쓰기 요청된 LPN(Logical Page Address)을 순차적으로 가져와 사상 테이블을 통해 LPN에 대한 PPN(Physical Page Address) 정보를 얻고 이후 플래시 컨트롤러에 실제 입출력 명령을 전달한다. hot data identification 기법은 이러한 과정 중 쓰기요청 된 LPN을 대상으로 hot data 판단 과정을 진행한다. 지금까지 알려진 hot data identification기법은 크게 쓰기요청 발생빈도를 기준으로 하는 MHF(Multi Hash Function Framework)기법과 쓰기요청 발생의 연속성을 기준으로 하는 BHF(Bloom filter Hash Function Framework)기법으로 구분할 수 있다.

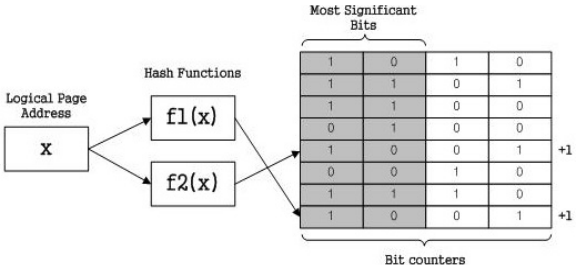

Fig. 4. Multi Hash Function Framework

MHF기법은 해시함수를 이용하여 쓰기 요청된 LPN을 해싱하고 해싱 값에 대응하는 비트배열에 값을 순차적으로 0에서 1로 변경함으로 해당 쓰기요청을 기록한다. 그리고 더 이상 쓰기요청이 없다면 기록된 값은 정해진 시간 값에 맞춰 가장 먼저 기록된 값부터 순차적으로 0으로 초기화된다. 이러한 초기화 과정을 통해 비트배열에 기록된 값이 항상 마지막 쓰기요청에 의해 기록된 값임을 나타낸다. MHF기법은 단순 비트배열이 4비트로 구성될 경우 상위2비트 값에 변화가 hot data 판단기준이 된다. 이러한 MHF기법은 동작구조의 단순함으로 운용에 장점이 있지만 정확한 hot data 판단에 있어 문제점이 존재한다. MHF기법은 특정 LPN을 대상으로 집중적인 쓰기요청이 발생한 이후 더 이상 쓰기요청이 발생하지 않는 경우에도 특정시간 동안 기록된 상위2비트의 쓰기요청 발생빈도 값만을 고려하여 hot data로 판단한다. 하지만 데이터 업데이트 과정에서 특정 LPN에 집중적인 쓰기요청이 발생하는 것은 일반적인 경우이며 hot data 판단에 있어 단순히 특정시간 동안 기록된 쓰기요청의 발생빈도만을 고려하는 것은 이러한 집중적인 특정 쓰기요청 패턴이 더 이상 발생하지 않는 경우에도 이를 hot data로 판단하기 때문이다. 하지만 MHF기법은 단순 비트배열을 이용하여 특정 시간동안 쓰기요청에 발생빈도만을 기록하여 판단함으로 이러한 집중적인 특정 쓰기요청 패턴이 지속적으로 발생하는 것인지 판단할 수 없다.

BHF기법은 해시함수를 이용하여 쓰기 요청된 LPN을 해싱하고 해싱 값에 대응하는 현재 선택된 블룸필터의 비트 값을 0에서 1로 변경한다. 또한 더 이상 쓰기요청 발생이 없다면 현재 선택된 블룸필터의 비트 값은 0으로 유지된다. BHF기법은 쓰기요청 발생 사실을 기록하기 위해 다수의 블룸필터를 라운드 로빈 방식으로 선택하게 된다. 이러한 BHF기법은 별도의 기록 값 삭제과정 없이 가장 오래전 기록된 블룸필터부터 순차적으로 삭제하며 삭제 후 새로운 블룸필터를 계속해 생성한다. BHF기법은 다수의 블룸필터에 기록된 쓰기요청의 발생의 연속성을 기준으로 hot data를 판단한다. 하지만 hot data 판단 시 쓰기요청 발생 사실의 연속성만을 고려할 경우 짧은 연속성을 가진 특정 쓰기요청 패턴이 자주 발생하는 경우 이를 정확히 hot data로 판단하지 못하는 문제가 발생한다.

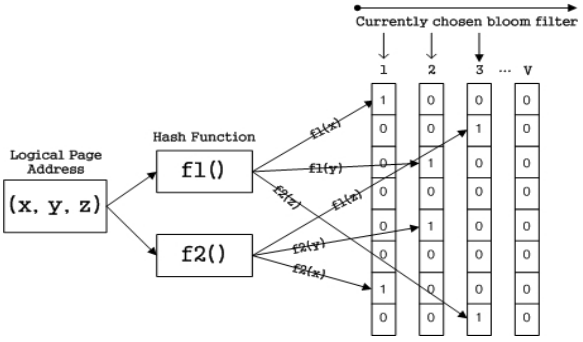

Fig. 5. Bloom filter Hash Function Framework

III. The Proposed Scheme

3.1 The operation structure of the proposal verification method

제안 검증기법은 특정 LPN을 hot data로 판단하기 위해 필요한 값을 기록하는 과정과 이 값을 이용하여 hot data 여부를 판단하는 과정 그리고 hot data로 판단된 LPN을 카운팅 필터 항목에 기록하여 유지하는 과정 마지막으로 특정 LPN이 hot data로 이미 기록되었는지 검색하는 과정 순으로 동작한다. 본 논문은 hot data 판단을 위해 값을 기록하고 이를 이용하여 hot data 여부를 판단하는 과정을 새롭게 제안하며 기록유지와 검색과정은 일반적인 카운팅 필터의 동작구조를 그대로 따른다. 제안 검증기법은 이를 위해 카운팅 필터를 이용한다. 이는 이전 기법에서 사용된 단순비트배열 또는 다수의 블룸필터와 비교하여 상대적으로 더 적은 메모리를 사용하지만 더 많은 hot data 항목을 기록할 수 있고 더 빠른 속도로 검색을 수행할 수 있다. 카운팅 필터 검색에 필요한 시간은 해시 함수의 개수에 의해 결정되므로 검색비용은 $O(k)$이며 집합 내 원소 개수와 무관하다. 제안 검증기법의 경우 4개의 해시 함수를 사용한다.

제안 검증기법의 동작은 시스템 큐를 폴링하여 쓰기 요청된 LPN을 카운팅 필터 항목에 추가함으로써 시작한다. 카운팅 필터 항목에 추가된 LPN은 해시함수를 이용해 해싱되고 반환된 값에 해당하는 카운팅 필터의 각 n비트에 hot data 판단과정에 필요한 값들이 기록된다. 이후 기록된 값을 이용하여 hot data 판단과정을

진행하며 이를 통해 hot data로 확인된 LPN은 카운팅 필터 항목에 계속 유지하고 hot data가 아닌 경우 항목에서 삭제한다. 이후 특정 LPN에 대한 쓰기요청 발생 시 해당 LPN이 카운팅 필터 항목에 존재하는지 검색하는 과정이 진행된다. 해당 LPN이 카운팅 필터 항목에 이미 기록되어 있는 경우에는 해당 LPN에 대한 hot data 판단과정을 재실행하지 않는다. 이렇게 hot data로 판단된 LPN은 FTL과 연동하여 별도의 사상알고리즘을 적용함으로 무효 페이지로 인한 오버헤드를 해결할 수 있다. 본 논문에서는 FTL 연동부분은 언급하지 않는다. 다음으로 hot data 판단과정 중 특정 LPN에 대한 쓰기요청을 hot data로 판단하는데 필요한 값을 기록하는 과정에 대해 설명한다.

3.2 A value recording process for judging the hot data of the proposal verification technique

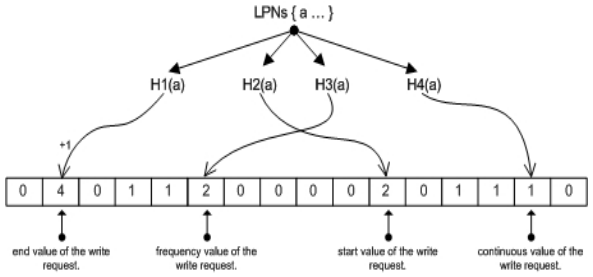

Fig. 6. operation process of the proposed method

제안 검증기법은 특정 LPN에 대한 쓰기요청 발생 시 해시함수 H1에 n비트의 값을 1씩 증가시키며 더 이상 쓰기 요청이 발생하지 않는 경우에는 미리 설정한 시간 간격으로 1씩 감소시킨다. 특히 H1에 n비트의 값이 최초 증가 또는 감소 동작 후 최초 증가하는 경우에는 H1에 n비트의 값을 H2에 n비트의 저장한 뒤 증가하며 H1에 n비트의 값이 감소하는 경우 최초 감소 시에만 H3에 n비트의 기록된 값을 1씩 증가시킨다. 하지만 H1에 n비트의 값이 연속하여 감소할 경우 H3에 n비트의 값은 1씩 감소한다. 또한 H4에 n비트의 값이 직전 H4에 n비트의 값과 보다 작을 경우에도 H3에 n비트의 값이 1감소된다. H4에 n비트의 값은 hot data판단과정 중 별도로 연산되어 결과 값이 기록된다. 이렇게 기록된 다양한 값들의 의미는 3.3절에서 상세히 설명한다. 마지막으로 이러한 값을 이용하여 hot data 여부를 판단하는 과정은 3.4절에서 상세히 설명한다.

3.3 The continuity of write requests and the number of occurrences of a particular write request pattern

쓰기요청의 연속성이란 특정 LPN에 대한 쓰기요청 발생 유무를 연속하여 기록한 것을 말한다. 이는 쓰기요청이 짧은 시간동안 연속 발생한 경우와 상대적으로 긴 시간동안 연속 발생한 경우로 나뉜다.

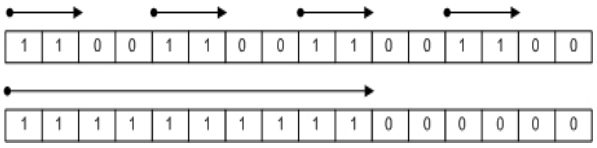

Fig. 7. continuity of write request

또한 특정 쓰기요청 패턴의 발생 횟수란 앞서 언급한 쓰기요청의 연속성이 특정 시간동안 몇 번 발생했는지 기록한 값을 말한다. 보다 정확한 hot data 판단을 위해서는 쓰기요청의 연속성과 특정 쓰기요청 패턴의 발생 횟수를 함께 고려해야한다. 이전기법의 경우 hot data 판단 시 쓰기요청의 연속성만을 고려함으로 쓰기요청이 한번만 긴 시간동안 연속 발생할 경우에도 이를 hot data로 판단하는 문제가 있다. 또한 쓰기요청 발생 사실을 단순 비트 값으로 기록함으로 짧은 연속성을 가진 쓰기요청이 자주 발생하는 경우와 긴 연속성을 가진 쓰기요청이 지속적으로 발생하는 경우를 정확히 판단할 수 없다.

이러한 문제를 해결하기 위해서는 쓰기요청의 연속성과 더불어 특정 쓰기요청 패턴의 발생 횟수를 함께 고려하여야하며 제안 검증기법은 이를 위해 카운팅 필터에 H1과 H2에 n비트에 쓰기요청의 연속성을 기록하고 H3과 H4에 n비트에 특정 쓰기요청의 패턴의 발생 횟수를 기록한다.

```
Algorithm 1. Continuity of writing operation.(lpn).
Input: lpn
Output: none

1 : While(True){
2 :   if(write operation occurred)
3 :     if( initial increase or first increase after decline)
4 :       nbit of h2 = nbit of h1
5 :     nbit of h1 += 1
6 :   else
7 :     if(nbit of h1 - 1 < 0)
8 :       continue
9 :     if(first decrease)
10 :      nbit of h3 += 1
11 :    nbit of h1 -= 1
12 : }
```

제안 검증기법의 경우 쓰기요청의 연속성을 확인하기 위해 특정 LPN에 대한 쓰기요청 발생사실을 해시함수 H1에 n비트에 기록한다. 하지만 이 값은 특정 LPN에 대한 쓰기요청 발생 유무를 단순 누적 기록한 것이며 이 값만으로 쓰기요청이 연속적으로 발생한 것인지 판단할 수 없다. 예를 들어 현재 H1에 n비트에 기록된 값이 3일 경우 이 값이 0 1 2 3 순으로 기록된 것인지 아니면 6 5 4 3 순으로 기록된 값인지 정확히 판단할 수 없다. 전자의 경우는 쓰기요청이 연속적으로 발생한다고 판단할 수 있다. 하지만 후자의 경우에는 더 이상 쓰기요청이 발생하지 않아 감소 중인 것으로 판단해야한다. 보다 정확한 쓰기요청의 연속성을 판단하기 위해서는 기록된 값이 최초로 증가하기 직전에 값을 별도로 기록하고 H1에 n비트의 기록된 값이 감소되기 직전까지를 쓰기요청의 연속성으로 판단해야한다.

이를 위해 H1에 n비트의 값이 최초 증가하기 직전에 값을

H2에 n비트에 저장함으로써 쓰기요청에 시작 값과 끝 값을 각각 기록하게 된다. 이를 통해 두 값의 차이를 확인하여 특정 LPN에 대한 쓰기요청의 연속성이 짧은 연속성인지 긴 연속성인지를 정확하게 판단할 수 있게 된다.

Algorithm 1.은 쓰기요청의 연속성을 기록하는 과정을 나타낸다. 다음으로 제안 검증기법은 특정 쓰기요청 패턴의 발생횟수를 기록하기 위해 H1에 n비트의 값이 최초 감소할 때 H3에 대응하는 n비트의 값을 1씩 증가시킨다. 이 값은 쓰기요청에 끝을 발생횟수로 누적 기록하는 것이다. 만약 H1에 n비트의 값이 연속 감소할 경우 H3에 대응하는 n비트의 값 또한 연속하여 1씩 감소시킨다. 이 값은 이전 쓰기요청의 연속성에 대한 횟수를 삭제하는 것이다. 더 이상 쓰기요청이 발생하지 않음으로 H1에 n비트의 값이 감소하여 H2에 n비트의 값보다 작거나 같아지는 경우에는 더 이상 해당 쓰기요청의 연속성이 유효하지 않음으로 특정 쓰기요청 패턴의 발생횟수를 누적하여 기록한 H3에 n비트와 쓰기요청의 연속성을 의미하는 H4에 n비트의 값을 0으로 기록함으로 해당 쓰기요청의 연속성과 특정 쓰기요청 패턴의 발생횟수에 대한 값을 모두 초기화하게 된다. 끝으로 이러한 H3에 n비트의 값만으로는 짧은 시간동안 발생한 쓰기요청에 대한 발생횟수인지 긴 시간동안 발생한 쓰기요청에 대한 발생횟수인지 판단할 수 없으므로 쓰기요청에 시작 값과 끝 값이 기록된 H1과 H2에 n비트의 값의 차이를 H4에 n비트에 누적 기록한다. 다음으로 이러한 값들을 활용하여 hot data를 판단하는 과정에 대해 설명한다.

3.4 The hot data judgment process of the proposal verification technique

제안 검증기법의 동작은 쓰기 요청된 LPN을 카운팅 필터 항목에 추가함으로써 시작하며 hot data 판단과정은 특정 LPN에 대한 쓰기요청 발생사실을 기록한 H1에 n비트의 값이 최초 감소할 때 동작한다. 하지만 H1에 n비트의 값이 H2에 n비트의 값보다 작다면 판단과정은 더 이상 진행하지 않는다.

특정 LPN에 대한 hot data 판단 기준은 짧은 연속성을 가진 쓰기요청이 자주 발생하는 경우와 긴 연속성을 가진 쓰기요청이 자주 발생하는 경우이며 그 외에 쓰기요청은 단순 데이터 업데이트 동작으로 판단한다. hot data 판단에 필요한 값은 두 가지이다.

첫째는 H1에 n비트의 값과 H2에 n비트의 값이며 이는 H4에 n비트에 서로 뺄셈되어 기록된다. 둘째는 H3의 n비트의 값이다. 이 두 가지 값의 의미는 각각 H4에 n비트의 값은 쓰기요청의 연속성을 의미하며 H3에 n비트의 값은 특정 쓰기요청 패턴의 발생 횟수를 의미한다. hot data 판단은 현재 각 n비트의 값과 직전 각 n비트의 값을 서로 비교를 통해 결정된다. 결론적으로 직전까지의 쓰기요청에 대한 H4와 H3에 기록 값과 현재의 쓰기요청에 대한 H4와 H3에 기록 값을 비교함으로 해당 LPN에 대한 쓰기요청의 변화를 검증하여 hot data 여부를 판단하는 것이다.

제안 검증기법의 hot data 판단과정을 좀 더 명확히 설명하기 위해 특정 LPN에 대한 쓰기요청 발생 시 기록되는 값들을 순차적으로 그림으로 설명한다. 이를 위해 일반적이지 않은 쓰기요청 상황을 예로 들어 설명하는 것임을 밝힌다. 먼저 Fig. 8.은 특정 LPN에 대한 쓰기요청이 최초 발생한 상황이며 이 때 각 n비트의 값은 0 0 0 0 로 초기화된다.

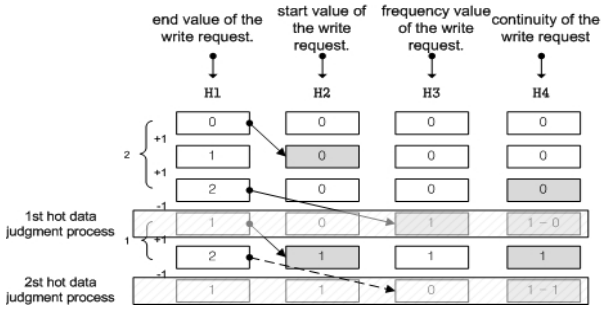

Fig. 8. Judgment Process 1

이후 두 번의 연속적인 쓰기요청이 발생하고 이후 특정 시간 간격 동안 쓰기요청 발생이 없음으로 쓰기요청 발생사실을 누적 기록하는 h1에 n비트의 값은 1씩 감소한다. 이때 첫 번째 hot data 판단과정이 동작하며 현재 각 n비트의 값은 1 0 1 1 이다. hot data 판정에 필요한 값은 h3와 h4에 n비트의 값이며 현재 h4에 n비트의 값이 1이므로 이를 통해 짧은 시간동안 연속하여 쓰기요청이 발생했음을 알 수 있으며 또한 현재 h3에 n비트의 값이 1이므로 짧은 연속성을 가진 쓰기요청이 한번 발생했음을 알 수 있다. 직전 각 n비트의 값은 2 0 0 0 이므로 현재 H3과 H4에 n비트의 값과 비교했을 때 모두 크다. 이는 쓰기요청의 연속성과 발생횟수 모두 증가함을 의미함으로 해당 LPN은 hot data로 판단한다. 하지만 이후 한 번에 쓰기요청이 더 발생한 뒤 바로 감소함으로 이 때 H4에 n비트의 값은 0이다. 이는 쓰기요청의 연속성이 종료되었음을 의미함으로 현재 H3에 n비트의 값 또한 0으로 초기화 한다. 이를 통해 각 n비트의 값은 1 1 0 0 으로 기록되며 이는 곧 카운팅 필터 항목에서 해당 LPN이 삭제됨을 의미한다. 그럼으로 이는 hot data로 판단되지 않는 경우이다.

Fig. 9.는 Fig. 8. 이후 3번의 쓰기요청과 1번에 감소 그리고 다시 3번의 쓰기요청과 1번에 감소 후 2번의 쓰기요청과 1번에 감소가 발생한 상황이다. 주목할 점은 다섯 번째 hot data 판단과정 중 h3에 n비트의 값이다. 이 값이 3으로 증가하지 않고 2로 계속하여 유지되는 이유는 현재 h4에 n비트의 값이 직전 h4에 n비트의 값보다 작음으로 1 감소하였기 때문이다. 이는 쓰기요청의 연속성이 앞선 3번에서 2번으로 바뀌었음을 의미함으로 해당 쓰기요청의 연속성을 이전 특정 쓰기요청 패턴의 발생횟수에 반영하지 않고 유지하기 위한 것이다.

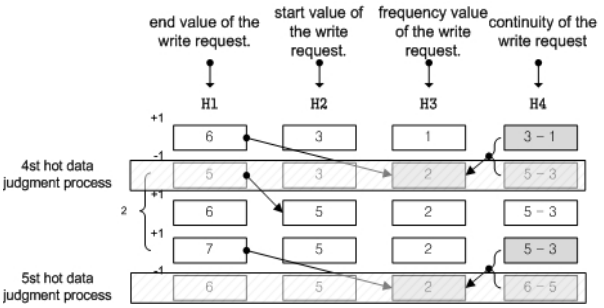

Fig. 9. Judgment Process 2

이러한 경우 현재 H4에 n비트의 값이 직전 H4에 n비트의 값보다 작지만 H3에 n비트의 값은 동일함으로 여전히 hot data 판단한다. 왜냐하면 이후 또 다시 같은 쓰기요청의 연속성이 발생할 수 있기 때문이며 이러한 경우 두 번의 연속적인 쓰기요청에 대한 발생횟수를 누적 기록함으로 이후 hot data 판단이 가능하기 때문이다. 결론적으로 직전까지의 쓰기요청에 대한 H4와 H3에 기록 값과 현재의 쓰기요청에 대한 H4와 H3에 기록 값을 비교함으로 해당 LPN에 대한 쓰기요청의 변화를 검증하여 hot data 여부를 판단하게 된다.

IV. Experimental

이 장에서는 실험을 통해 2장에서 소개한 블룸필터를 이용한 hot data 검증기법과 3장에서 제안한 hot data 검증기법을 각각 시뮬레이션 프로그램으로 구현하고 일반적인 pc사용 환경에서 생성한 워크로드를 리눅스의 blktrace를 이용하여 읽기/쓰기요청 명령을 추출한 뒤 이를 대상으로 hot data 판단비율과 블록삭제 발생비율을 분석함으로써 제안한 기법이 상대적으로 효율적임을 검증한다.

4.1 Experimental environment and Trace file spec

실험에 사용된 시스템은 인텔 i5-6500, DDR3 1600MHz 4GB 메모리로 구성되며 Centos 7.3-1611 버전의 운영체제를 사용한다. 전체 SSD의 크기는 8GB가 되도록 65,536개의 블록을 할당하였으며 또한 읽기/쓰기가 혼합된 명령들 중 쓰기요청 발생은 약 480,000회 이다. 마지막으로 실험 진행에 있어 쓰기요청 이외에 대해서는 고려하지 않는다.

Table 2. Trace file characteristics

property	value
Total Requests	1,138,396
Read request	655,198
Write request	483,198
Using block	65,536

4.2 Operation time overhead using counting filter

hot data 검증기법을 평가하는데 동작시간 오버헤드는 중요한 고려사항이다. Fig. 10.은 블룸필터기법과 제안기법 중 hot data 판단과 기록 그리고 삭제과정에 대한 동작시간 오버헤드를 cpu clock 사이클을 기준으로 비교한 것이다.

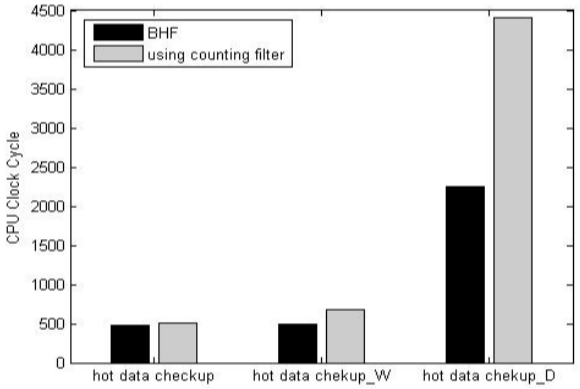

Fig. 10. CPU clock cycle comparison

결과적으로 카운팅 필터 사용 시 hot data 판단과 기록에 경우 차이가 미미하며 상대적으로 삭제과정에서 차이가 큼을 알 수 있다. 이러한 결과는 블룸필터의 경우 비트연산을 통해 비트 백터를 변경하지만 카운팅 필터의 경우 inc, dec과 같은 카운터 증가, 감소 연산과 cmp명령을 이용해 카운터 최소값과 최대값을 확인하므로 동작시간이 상대적으로 더 소요된 것으로 판단할 수 있다. 하지만 이러한 동작속도 오버헤드는 삭제연산에 대한 최적화 처리를 통해 극복할 수 있는 정도이며 보다 정확한 hot data 판단으로 인한 전체적인 성능향상으로 상쇄할 수 있다.

4.3 Analysis of experimental results

실험 결과 블룸필터를 이용한 hot data 검증기법의 경우 전체 쓰기요청 중 hot data로 판단된 쓰기요청의 비율은 전체 쓰기요청 중 약 13%이며 제안한 기법의 경우 약 9%이다. 기존 기법과 비교해 약3% 정도 hot data판단비율에 차이가 발생하는데 이는 실험에 사용한 워크로드의 특성에 영향을 받는 것이며 sql 쿼리문 사용이 많은 경우 랜덤쓰기가 상대적으로 많이 발생한다. 제안기법의 경우 순차쓰기보다 랜덤쓰기가 더 많이 발생할 경우 상대적으로 더 낮은 hot data비율을 보인다.

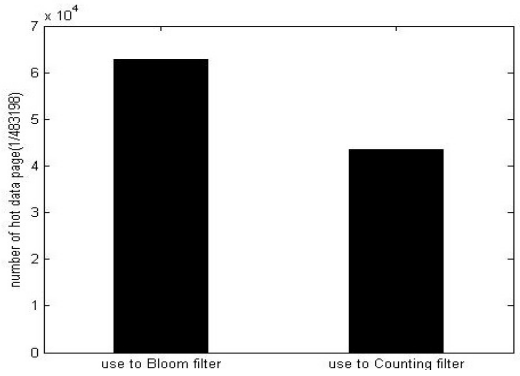

Fig. 11. Number of hot data pages when applying Bloom filter technique and Counting filter technique

이는 블룸필터를 이용한 hot data 검증기법의 경우 쓰기요청의 연속성만을 고려함으로써 집중적인 쓰기요청 이후 더 이상 쓰기요청이 발생하지 않는 경우에도 이를 hot data로 판정하기 때문이다. 반면 제안기법의 경우 쓰기요청에 연속성과 더불어 특정 쓰기요청 패턴의 발생횟수를 함께 고려함으로 집중적인 쓰기요청이 가끔 발생하는 경우와 짧은 연속성을 가진 쓰기요청이 가끔 발생하는 경우 이를 hot data로 판정하지 않기 때문이다[11].

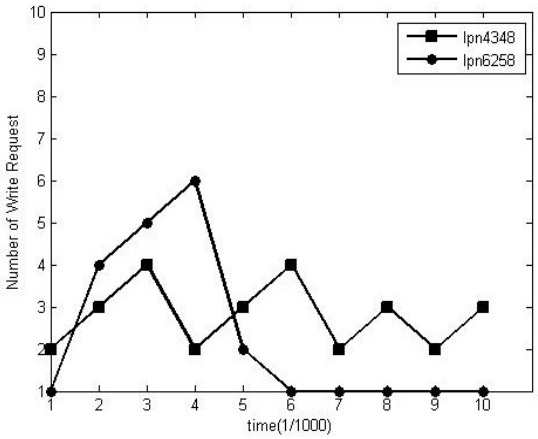

Fig. 12. Write requests with short continuity

또한 Fig. 12.에서처럼 짧은 연속성의 쓰기요청이 자주 발생하는 경우를 확인할 수 있다. LPN 4348번의 경우 쓰기요청 발생이 비교적 짧게 발생하지만 그 발생 사실이 지속적으로 발생함을 알 수 있다. 이러한 쓰기요청에 경우 블룸필터를 이용한 검증기법의 경우 단순히 쓰기요청 발생이 충분히 연속적이지 않음으로 hot data로 판정하지 않게 된다. 하지만 이러한 지속적인 짧은 쓰기요청은 많은 무효페이지를 발생시키는 원인이 됨으로 이를 정확히 hot data로 판단할 수 있어야 한다.

4.4 Rate of invalid page occurrence

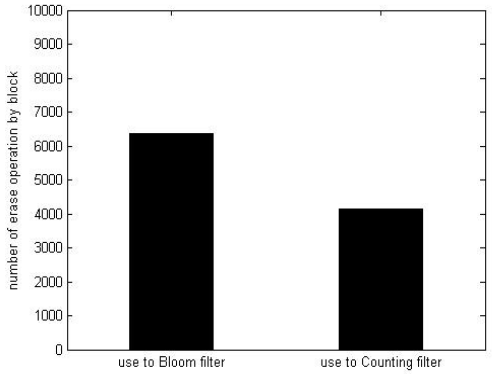

Fig. 13. Number of hot data pages when applying MHF technique and CMHF

Fig. 13.은 블룸필터 기법을 적용한 hot data 검증기법과 제안한 hot data 검증기법을 적용했을 경우 발생하는 블록삭제 요청을 나타낸다. 블룸필터 기법 적용 시 hot data 판단 비율은 상대적으로 제안기법 적용 시 보다 더 높았지만 결과적으로 블록 삭제요청이 더 많이 발생하는 것을 알 수 있으며 제안기법에 적용 시 hot data 판단 비율은 상대적으로 블룸필터를 이용한 기법 적용 시 보다 더 낮았지만 결과적으로 블록삭제 요청은 더 적게 발생하는 것을 알 수 있다. 이는 블룸필터를 이용한 hot data 검증기법에 경우 단순히 연속적인 쓰기요청에 대한 검증만으로 hot data를 판단하기 때문에 짧은 연속성의 쓰기요청을 hot data로 판단하지 못함으로 인해 오히려 블록 내 무효페이지 발생이 비율이 증가하였으며 이로 인한 블록삭제 요청이 증가함으로 결과적으로 블룸필터를 이용한 hot data 검증기법의 경우 hot data 판단 비율은 높았지만 실제 블록 삭제에 영향을 미치는 hot data가 아니었음을 알 수 있다.

V. Conclusions

본 논문에서는 기존 hot data 검증기법 중 하나인 블룸필터를 이용한 hot data 검증기법을 소개하였으며 해당 검증기법의 경우 hot data 판단 시 쓰기요청에 대한 연속성만을 고려함으로 짧은 연속성의 쓰기요청이 자주 발생하는 경우를 hot data로 판단하지 못하는 문제점이 있으며 해당 검증기법은 hot data 판단비율은 높지만 블록 삭제 사실은 상대적으로 더 많이 발생함을 실험을 통해 확인할 수 있었다. 반면 제안기법의 경우 쓰기요청의 연속성과 더불어 특정 쓰기요청의 발생횟수를 함께 고려함으로 상대적으로 hot data 판단비율은 적었지만 결과적으로 블록삭제 요청이 더 적게 발생하는 것을 확인할 수 있었다. 이는 짧은 연속성의 쓰기요청이 지속적으로 발생하는 경우를 정확히 검증한 결과로 해석할 수 있다. 향후연구 과제로 본 논문에서 제안한 검증기법으로 확인된 hot data를 관리할 수

있는 완성된 페이지 기반의 낸드 플래시 사상 알고리즘 연구를 진행할 계획이다.

REFERENCES

[1] Gartner, https://www.gartner.com/en/newsroom/press-releases/2017-10-17-gartner-says-worldwide-device-shipments-will-increase-2-percent-in-2018

[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] 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.

[4] 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.

[5] 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.

[6] Dong-chul 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).

[7] 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.

[8] Hyesook Lim, Jungwon Lee, Changhoon Yim, "Complement Bloom Filter for Identifying True Positiveness of a Bloom Filter" IEEE Communications Letters (Volume: 19 , Issue: 11 , Nov. 2015)

[9] Ori Rottenstreich, Isaac Keslassy, "The Bloom Paradox: When Not to Use a Bloom Filter" IEEE/ACM Transactions on Networking (Volume: 23 , Issue: 3 , June 2015)

[10] Peizhen Lin, Feng Wang, Weiliang Tan, Hui Deng, "Enhancing Dynamic Packet Filtering Technique with d-Left Counting Bloom Filter Algorithm" 2009 Second International Conference on Intelligent Networks and Intelligent Systems

[11] Access Pattern, http://tech.kakao.com/2016/07/17/coding-for-ssd-part-5/

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.