KIPS Trans. Comp. and Comm. Sys.
Vol.11, No.12 pp.419~428
ISSN: 2287-5891 (Print), ISSN 2734-049X (Online)

https://doi.org/10.3745/KTCCS.2022.11.12.419

2R++: Enhancing 2R FTL to Identify Warm Pages

Hyojun An† · Sangwon Lee††

ABSTRACT

Since in-place updates for pages are not allowed in flash memory, all new page writes should be written in an out-of-place manner. The old overwritten pages are invalidated. Such invalidated pages eventually trigger the costly garbage collection process. Since the garbage collection causes numerous read and write operations, it is one of the flash memory's major performance issues. In 2R, it modified the garbage collection algorithm, which applies the I/O characteristics of the On-Line Transaction Process workload to improve the Write Amplification Factor. However, this algorithm has a region pollution problem. Therefore, in this paper, we developed 2R++ that additionally separates pages with long access intervals to solve the region pollution problem. 2R++ introduces an extra bit per block to separate warm pages based on a second chance mechanism. Prevents warm pages from being misidentified as cold pages to solve region pollution problem. We conducted the experiments on TPC-C and Linkbench to make the performance comparison. The experiment showed that 2R++ achieved a Write Amplification Factor improvement of 57.8% and 13.8% compared to 2R, respectively.

Keywords : Flash Memory, Garbage Collection, Page Separation

2R++: Warm Page 식별을 통한 2R FTL 개선

안 효 준† · 이 상 원††

요 약

플래시 메모리는 in-place 수정이 불가능한 특성을 가지기 때문에 out-of-place 방식으로 쓰기 작업을 수행한다. 덮어쓰기가 발생한 오래된 페이지는 유효하지 않은 페이지로 전환된다. 유효하지 않은 페이지들은 높은 오버헤드를 가진 가비지 컬렉션 과정을 유발한다. 가비지 컬렉션은 많은 읽기, 쓰기 작업을 유발하기 때문에 플래시 메모리의 주요 성능 이슈 중 하나이다. 2R에서는 OLTP(On-Line Transaction Process) 워크로드의 I/O 특성을 가비지 컬렉션 알고리즘에 적용하여 WAF(Write Amplification Factor)를 개선하였다. 본 논문에서는 접근 간격이 긴 페이지들을 추가로 분리하는 2R++를 통해 2R에서 발생하는 지역 오염 문제를 해결했다. 2R++는 블록 당 추가 bit를 도입해 second chance mechanism 기반으로 warm 페이지를 분리해서 warm 페이지가 cold 페이지로 오인 식별되는 것을 방지한다. TPC-C와 Linkbench에 대해 알고리즘 별 성능 비교 실험을 진행하였고, 그 결과 2R++의 WAF는 2R대비 각각 57.8%, 13.8%의 개선을 이루어냈음을 확인했다.

키워드 : 플래시 메모리, 가비지 컬렉션, 페이지 분리

1. 서 론

하드디스크의 뒤를 이어 메인 저장장치의 지위를 이어받은 플래시 메모리 저장장치는 하드디스크와 비교하여 매우 빠른 속도를 가지지만 in-place 업데이트가 불가능하다는 특성을 가진다. 플래시 메모리는 기존의 데이터를 유효하지 않은 데이터로 표시하고 빈 공간에 수정된 데이터를 새로 작성하는 확장의 형태로 데이터를 수정한다. 플래시 메모리는 수정된 데이터에 접근하기 위한 주소 매핑 기법과 유효하지 않은 페이지들을 다시 빈 페이지로 바꾸는 가비지 컬렉션 등을 제공하고 FTL(Flash Translation Layer)에서 이를 관리한다.

가비지 컬렉션은 플래시 메모리에 공간이 부족할 때 새로운 공간을 만들기 위한 동작이다. 블록 단위로 유효하지 않은 페이지들을 지우고, 유효한 페이지들은 새로운 빈 블록으로 옮기고 이 과정에서 사용자가 요청한 I/O와 별개로 추가적인 I/O를 발생시킨다. 추가 I/O는 플래시 메모리의 주요 성능 이슈 중 하나로 이를 줄이기 위해 이전부터 많은 관련 연구들

※ 이 성과는 정부(과학기술정보통신부)의 재원으로 한국연구재단의 지원을 받아 수행된 연구임(No. 2022R1A2C2008225).
※ 이 논문은 2022년도 정부(과학기술정보통신부)의 재원으로 정보통신기획평가원의 지원을 받아 수행된 연구임(No.2015-0-00314,비휘발성 메모리 기반 개방형 고성능 DBMS 개발).
† 비 회 원 : 성균관대학교 소프트웨어학과 석사과정
†† 정 회 원 : 성균관대학교 소프트웨어융합대학 교수
Manuscript Received : August 24, 2022
First Revision : October 7, 2022
Accepted : October 31, 2022
* Corresponding Author : Sangwon Lee(swlee@skku.edu)

이 수행되었다. 본 논문의 기반이 되는 2R FTL은 OLTP 워크로드의 패턴을 분석하고 가비지 컬렉션에 적용한 FTL로 상용 플래시 메모리에서 주로 사용되는 그리디 FTL과 비교하여 WAF를 크게 개선하였다[1]. 하지만 2R에서는 긴 접근 간격을 가진 페이지들을 분리하지 못하기 때문에 페이지들이 지역을 오가면서 지역 오염 문제를 발생시킨다.

본 논문에서는 가비지 컬렉션이 수행되는 페이지들에 추가적인 기회를 주어 지역 오염 문제를 해결하고, 실험을 통해 성능 향상에 대한 분석을 진행한다. 본 연구는 QEMU 기반의 DRAM-backed NVMe SSD Emulator인 FEMU 환경에서 수행되었으며, 대표적인 OLTP 벤치마크인 TPC-C와 Linkbench를 사용하여 각 FTL에 대한 성능 비교를 진행하였다.

본 논문의 구성은 다음과 같다. 2장에서는 논문의 배경에 대해서 설명한다. 3장에서는 새롭게 작성된 FTL에 대해서 설명하고 4장과 5장에서는 실험 환경과 실험 결과에 대해서 설명하고 분석한다. 마지막으로 6장에서 관련 연구와 본 논문을 비교해보고 7장에서 실험의 결과를 정리하여 결론을 내림으로써 논문을 마무리한다.

2. 배 경

2.1 OLTP 워크로드의 쓰기 특성

OLTP는 데이터 기입 및 수정 등 대량의 트랜잭션을 수행하는 데이터 처리 유형이다. OLTP에서는 수많은 I/O가 발생하기 때문에 OLTP의 특성을 이해하고 저장장치를 설계하는 것은 저장장치의 성능 향상에 중요한 역할을 한다. OLTP의 대표적인 세 가지 쓰기 특성을 아래에서 설명한다.

1) 시간적 지역성

시간적 지역성은 대부분의 워크로드에서 볼 수 있는 특징으로 최근에 접근한 페이지는 빠른 시일 내에 재접근이 이루어질 가능성이 높다. 시간적 지역성으로 인해 페이지들은 접근 빈도가 높은 hot 상태와 접근 빈도가 낮은 cold 상태가 비교적 명확하게 나누어진다.

2) Skewness

데이터들은 독립적으로 각각의 개별 접근 확률을 가지고 접근되며, 접근이 자주 발생하는 정도에 따라 hot, warm, cold 페이지 등으로 나눌 수 있다. Skewness는 OLTP에서 대표적으로 확인할 수 있는 특징이다[2]. OLTP의 Skewness를 직접 확인하기 위해 TPC-C, Linkbench, fio에서 360GB의 쓰기를 수행하고 각 페이지에서 발생한 쓰기 횟수를 기록한 후, 그 결과를 Fig. 1에 정리하였다.

fio는 마이크로 벤치마크로 zipf distribution 옵션을 통해 zipf의 법칙 따르는 skewness를 생성할 수 있다. zipf의 법칙은 총 N 개의 요소들 중 k번째 요소의 빈도를 구하는 공식으로 각 요소들의 빈도는 서로 독립적이다.

Fig. 1은 누적 분포 함수로, 전체 페이지를 접근이 많은 수로 정렬하여 x축에 배치한 후, 페이지 별로 접근한 횟수를 누적으로 더하여 y축에 나타내었다. TPC-C 벤치마크는 20%의 페이지에서 90%가, Linkbench에서는 20%의 페이지에서 80%의 접근이 발생하여 대부분의 접근이 일부 페이지에 치중되는 것을 확인할 수 있다.

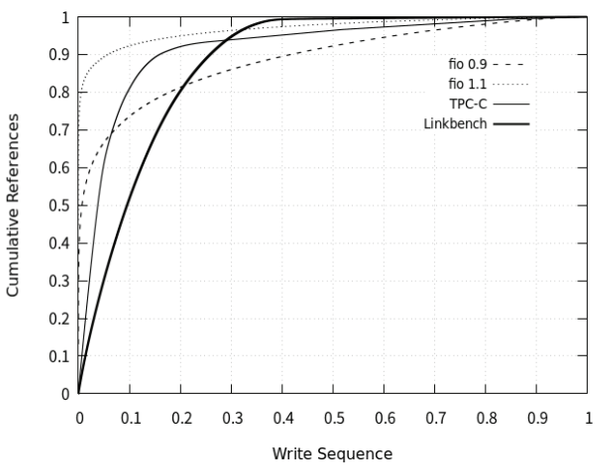

Fig. 1. Cumulative Distribution Function of Skewness

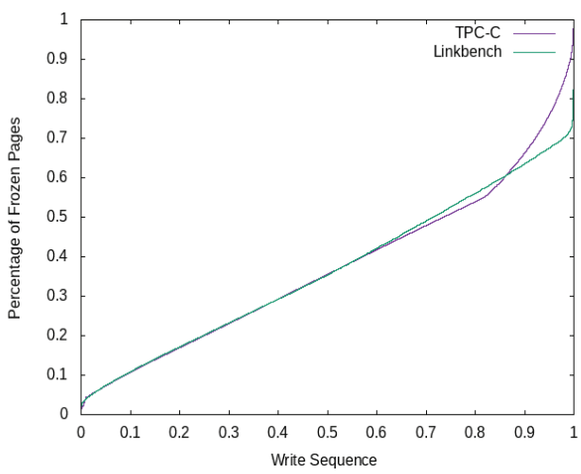

Fig. 2. Ratio of Frozen Pages

Skewness에 따라 페이지는 일정한 간격을 가지고 접근이 발생하며 간격에 따라 hot, warm, cold 등으로 나눌 수 있다.

3) 동결 페이지

더 이상 쓰기 접근이 발생하지 않는 페이지들을 동결 페이지라고 부르며, OLTP에는 많은 데이터들이 시간이 지남에 따라 동결 페이지로 전환된다. Fig. 2는 OLTP 벤치마크의 진행에 따라 저장 장치에 존재하는 동결 페이지의 비율의 변화를 나타내고 있다. 그래프에서 OLTP 벤치마크가 60% 진행되었을 때 전체 공간의 40%가 동결 페이지임을 확인할 수 있다. TPC-C의 경우, 이전의 기록을 저장하고 관리하는 order_line, order, history 테이블에 의해 동결 페이지가 생성된다[2].

2.2 가비지 컬렉션

플래시 메모리는 데이터의 덮어쓰기가 불가능한 특성을 가진다. 플래시 메모리에서 데이터의 업데이트가 발생하면 플래시 메모리는 기존의 데이터가 저장된 페이지를 유효하지 않은 페이지로 표시하고 새로운 위치에 수정된 데이터를 작성한다. 그리고 새로운 위치에 저장된 데이터에 접근할 수 있도록 논리 주소와 물리 주소의 연결을 업데이트하는 주소 매핑을 진행한다. 이를 out-place 업데이트라고 하며, out-place 업데이트 과정에서 생성된 유효하지 않은 페이지들은 바로 지워지지 않고 저장 장치에 남아 공간을 차지하다 추후 데이터를 저장할 공간이 부족해지면 지워져 빈 페이지로 변환된다. 플래시 메모리의 읽기와 쓰기는 페이지 단위로 진행되는 반면, 지우기는 블록 단위로 진행되기 때문에 가비지 컬렉션 과정을 거쳐야 한다. 가비지 컬렉션은 지우기를 진행할 블록을 선정하고, 선택한 블록에 위치한 유효 페이지들을 다른 블록으로 옮긴다. 유효한 페이지들을 옮기는 과정을 카피백이라고 한다. 카피백은 요청한 쓰기 작업 이외에 추가로 발생하는 쓰기 작업으로 플래시 메모리의 성능 저하를 유발한다. 마지막으로 블록을 빈 블록으로 바꾸는 지우기 작업을 완료하면 가비지 컬렉션 과정이 종료된다.

가비지 컬렉션에서 어떤 블록을 희생 블록으로 선택하느냐에 따라 카피백의 횟수가 크게 달라지고, 이는 플래시 메모리의 성능과 직결된다. 일반적인 상용 플래시 메모리에서는 주로 그리디 FTL을 사용한다. 그리디 FTL은 유효한 페이지의 수가 가장 적은 블록을 희생 블록으로 선택하는 FTL로 간단하고 빠르다는 장점이 있다. 하지만 워크로드의 접근 패턴에 대한 고려가 존재하지 않기 때문에 곧 다시 접근할 희생 블록을 선택하는 경우 큰 성능 저하가 발생한다. 이를 방지하기 위해 OLTP의 접근 패턴을 고려하여 작성된 FTL이 2R FTL이다[1].

2.3 2R FTL

2R FTL은 OLTP의 접근 패턴의 특성을 이용하는 가비지 컬렉션 알고리즘이다. 서로 다른 skewness를 가진 데이터가 같은 블록 내에 존재하면 블록에 유효한 페이지와 유효하지 않은 페이지가 서로 혼재되어 가비지 컬렉션의 희생 블록의 utilization을 해친다. Utilization이 높은 블록에서 발생하는 높은 카피백은 성능의 저하로 이어진다. 2R에서는 normal region에 쓰기를 진행하고 cold region에 카피백을 진행하여 데이터를 나누어 저장함으로써 skewness로 인한 문제를 해결한다[1]. Fig. 3은 2R의 동작 방식을 간단하게 나타낸 그림이다[1)].

시간 지역성을 특성으로 가지는 페이지들은 한 번 접근하면 이른 시일 내에 다시 접근할 확률이 높다. Cold와 normal region은 각각 cold 블록과 normal 블록의 집합이며, 2R에서는 쓰기가 발생한 모든 페이지를 normal region에 위치시켜 시간적 지역성을 만족한다. 최근에 접근한 시간 지역성이 강한 페이지들이 위치한 블록을 희생 블록으로 선정할 경우, 카피백 되어 옮겨진 페이지에서 다시 접근이 발생할 수 있다. 이는 불필요한 카피백으로 최근에 접근한 일정 비율의 블록은 희생 블록으로 선정하지 않음으로써 문제를 해결한다. 실험을 통하여 알아낸 비율의 최적 값은 20%이다[1].

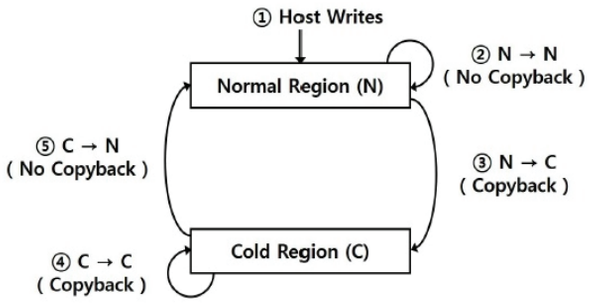

Fig. 3. Basic Operation of 2R

동결 페이지는 더 이상 접근이 발생하지 않는 페이지로 동결 페이지는 영구히 유효한 페이지로 존재한다. 다른 페이지들과 같은 블록에 위치해 희생 페이지로 선택될 때마다 추가적인 카피백을 발생시킨다. 동결 페이지로 인한 카피백이 발생하지 않도록 동결 페이지만 모여 있는 동결 블록을 생성하여 문제를 해결한다. 2R에서는 희생 블록을 여러 개 선정한 후 유효한 페이지들을 병합하는 다중 희생 블록 선택 방식을 채택하였다[1]. 선택한 블록들에 위치한 유효한 페이지들이 블록 하나의 크기보다 커질 때까지 추가 블록을 선택한다. 추가 희생 블록은 처음 선택한 희생 블록이 위치한 region에서만 선택한다. Cold region에서 발생하는 다중 희생 블록 선택 정책은 cold region에서도 여전히 유효한 더 cold한 페이지들끼리 동일한 블록에 배치되도록 만든다. 이는 장기적으로 동결 블록의 생성을 유도한다[1].

2R은 시간적 지역성과 블록의 utilization을 모두 고려하기 위해서 희생 블록을 FIFO 형태로 선택한다[1]. 쓰기가 완료된 시간 순으로 정렬된 블록들의 집합에서 FIFO 형태로 희생 블록을 선택한다. FIFO로 선택한 블록의 utilization이 특정 임계값보다 높을 경우, 희생 블록으로 선택하지 않고 지나간다. 이는 불필요한 많은 카피백이 발생하는 것을 방지하고 유효한 페이지들이 유효하지 않은 페이지로 전환될 추가적인 기회를 제공한다[1]. 희생 블록 선택은 이전 가비지 컬렉션이 완료된 시점에서 다시 재개된다.

OLTP의 접근 패턴에 대한 충분한 고려와 반영으로 2R FTL은 그리디 FTL 대비 높은 WAF 감소 효과를 보였다.

2.4 Warm 페이지에 의한 2R FTL의 cold region 오염 문제

하지만 2R에서 쓰기 간격이 길어 쓰기와 카피백이 번갈아

1) Fig. 3은 "2R: efficiently isolating cold pages in flash storages" 논문의 원작자의 허락을 받아 첨부되었음.

발생하는 warm 페이지는 normal region과 cold region을 오가면서 cold region의 utilization을 저하시키는 지역 오염을 발생시킨다.

Fig. 4는 2R FTL을 진행하는 동안 발생하는 cold region 오염에 대해서 나타낸 그래프이다. 1G에서 10G까지 데이터베이스가 증가하는 동안 TPC-C와 Linkbench에 대해 2R FTL을 수행하였다. 카피백으로 인해 cold region으로 이동한 페이지들 중 쓰기 요청에 의해 다시 normal region으로 돌아온 페이지의 비율을 나타내었다. 다시 돌아오는 페이지의 비율이 TPC-C에서는 67.71%, Linkbench에서는 97.75%로 cold region이 역할을 제대로 수행하지 못하고 있다. Warm 페이지는 cold region에서 덮어 쓰기를 통해 유효하지 않은 페이지가 되면서 cold region의 utilization을 떨어뜨린다. Cold region에서의 utilization 저하는 cold region에서의 희생 블록 선택으로 이어진다. 이는 동결 페이지들의 카피백을 발생시키고 2R에서 WAF가 증가하게 되는 주요 이유가 된다.

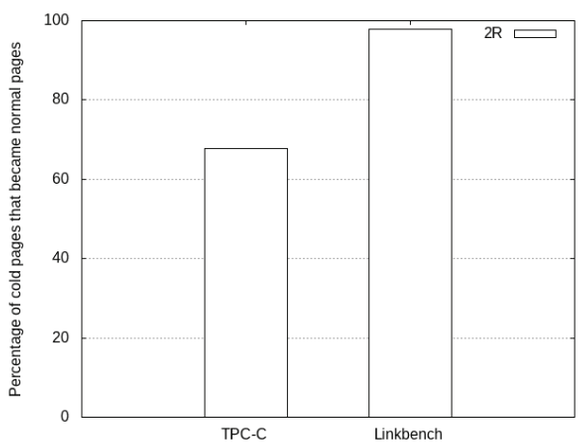

Fig. 4. Ratio of Pages Returned from Cold Region to Normal Region

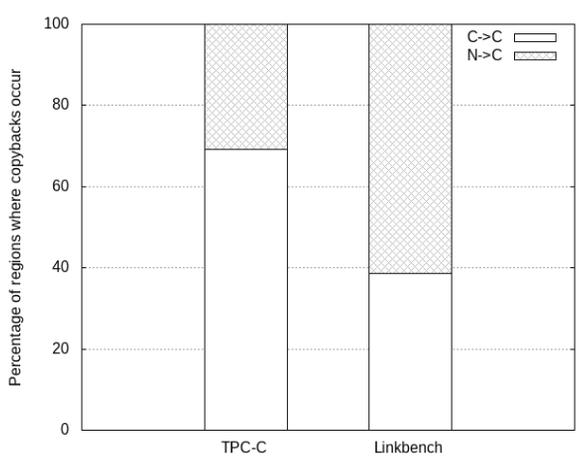

Fig. 5. Copybacks in the Normal Region and the Cold Region Respectively

각 지역에서 발생한 카피백이 전체 카피백에서 차지하고 있는 비율을 Fig. 5에 나타내었다. N->C는 normal region에서 cold region으로 발생한 카피백을 의미하고, C->C는 cold region에서 cold region으로 발생한 카피백을 의미한다. TPC-C에서 C->C가 차지하는 비율은 69.21%, Linkbench에서 C->C가 차지하는 비율은 38.53%이다. TPC-C와 Linkbench 모두에서 cold region내에서 발생하는 카피백이 차지하는 높은 비율을 통해 2R이 cold 페이지와 warm 페이지를 정확히 분리하지 못하고 있는 것을 확인 할 수 있다.

3. 2R++

3.1 2R++ 설계

쓰기와 카피백이 번갈아 발생하는 페이지들은 지역 오염을 유발하고 cold region의 utilization을 해친다. 이는 2R의 WAF가 증가하는 주요 원인이다. 본 논문에서는 이를 해결하기 위해 2R++이라는 추가 알고리즘을 제안한다. 2R++에서는 normal region과 cold region의 적절한 분리보다 cold region에서 쓰기가 최대한 발생하지 않도록 페이지들을 배치하는 것을 목표로 한다. 2R++에서는 희생 블록으로 적절한 블록을 선택하는 것보다 희생 블록으로 선정되면 안 되는 블록을 만드는 것을 설계 목표로 한다[3].

2R++는 2R과 비슷한 동작방식을 가진다. 쓰기가 완료된 순서대로 블록들을 관리하고 이를 희생 블록 선택에 사용한다. 순서대로 블록을 탐색하며 블록의 utilization이 특정 임계값보다 낮을 경우, 희생 블록으로 선택한다. 임계값이 높으면 utilization이 높은 블록이 희생 블록으로 선택되어 많은 카피백이 발생할 수 있고, 임계값이 낮으면 희생 블록을 선택할 수 없는 문제가 발생할 수 있기 때문에 본 논문에서는 이 임계값을 40%로 설정하였다.

첫 번째 희생 블록을 선택했다면 같은 region에 위치한 추가적인 희생 블록을 선택한다. 희생 블록은 블록들에 위치한 유효하지 않은 페이지들의 합이 블록 하나의 크기에 도달할 때까지 선택된다. 선택된 희생 블록들에 위치한 유효 페이지들은 빈 블록으로 카피백된다. 이 과정은 skewness가 비슷한 페이지들을 함께 카피백하여 블록의 일관성을 높이고 동결 블록의 생성을 유도한다.

최근에 접근이 발생한 20%의 블록은 희생 블록으로 선택하지 않음으로써 높은 시간적 지역성을 가진 블록들이 카피백되어 cold region의 utilization을 떨어뜨리는 문제를 미리 방지한다.

2R++의 페이지 배치방식에 대해 Fig. 6에 나타내었다. 2R++는 2R이 블록 당 1bit의 추가 오버헤드를 사용하던 것에서 1bit를 추가해 총 2bit를 사용하여 페이지 배치 방식에 추가적인 알고리즘을 적용한다. 추가된 2bit는 warm 페이지

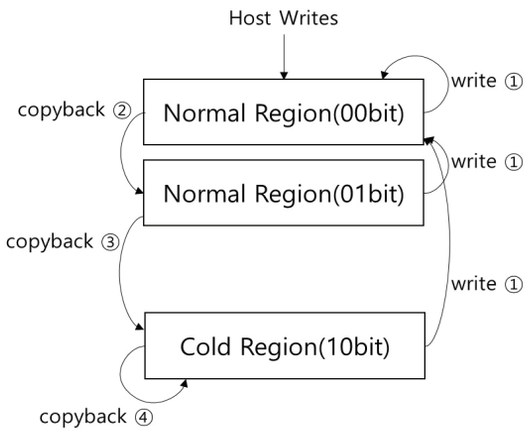

Fig. 6. Basic Operation of 2R++

2R++ Algorithm

```
1:  procedure WRITE (P)
2:      /* CurBlk: current normal block for write */
3:      if no free page in CurBlk then
4:          CurBlk = GET_NEW_BLOCK(W);
5:      end if
6:      if P is already existing in any block then
7:          mark invalid in original block
8:      end if
9:      write P to CurBlk;
10: end procedure
11:
12: procedure GET_NEW_BLOCK(S)
13:     if S == W AND no free block then
14:         GARBAGE_COLLECT();
15:     end if
16:     if S == W then
17:         set FreeBlock.bit = 00; /* It means normal block to write */
18:     elif S == N then
19:         set FreeBlock.bit = 01; /* It means normal block to copyback */
20:     elif S == C then
21:         set FreeBlock.bit = 10; /* It means cold block to copyback */
22:     end if
23:     return FreeBlock;
24: end procedure
25:
26: procedure GARBAGE_COLLECT()
27:     /* victim selection using the FIFO policy */
28:     select N victim blocks from one region
29:     /* ColBlk: current normal block for copyback */
30:     /* NorBlk: current normal block for copyback */
31:     if page in 00 normal block then
32:         copy valid pages in NorBlk;
33:         if no free page in NorBlk then
34:             NorBlk = GET_NEW_BLOCK(N);
35:         end if
36:     elif page in 01 normal block or cold block then
37:         copy valid pages in ColBlk;
38:         if no free page in ColBlk then
39:             ColBlk = GET_NEW_BLOCK(C);
40:         end if
41:     end if
42:     erase N victim blocks;
43: end procedure
```

Fig. 7. Algorithm of 2R++

와 cold 페이지들을 구분하기 위한 second chance의 구현을 위해 사용된다. 쓰기가 요청된 모든 페이지는 위치하고 있던 블록에 상관없이 00 값을 가진 normal 블록에 배치된다(Fig. 6①). 00과 01 값을 가진 블록들은 모두 normal region에 위치한 normal 블록들로 가비지 컬렉션 단계의 다중 희생 블록 선택 단계에서 함께 선택될 수 있다. 00 블록에서 카피백되는 페이지는 01의 값을 가진 normal 블록에, 01의 normal 블록에서 카피백되는 페이지는 10의 cold 블록에 배치된다(Fig. 6②, ③). Cold region에서 카피백된 페이지는 다시 cold region에 배치된다(Fig. 6④).

Cold region에 배치되기 전에 01 normal 블록에 한 번 카피백되는 과정은 warm 페이지에 second chance를 부여함으로써 warm 페이지가 cold region에 잘못 배치되는 것을 막는다. 이는 cold 페이지들의 정확한 분리로 이어진다.

Fig. 7은 2R++의 의사코드이다. CurBlk은 쓰기가 발생한 페이지를 배치할 블록을 가리키는 포인터이다. 쓰기로 인해 CurBlk이 가리키는 블록이 가득 차면 GET_NEW_BLOCK() 함수를 호출해 Curblk이 가리킬 빈 블록을 가져온다. Curblk이 가리키는 블록은 normal 블록이며 bit 값은 00이다. 빈 블록이 없을 경우, 빈 블록을 만들기 위해 GARBAGE_COLLECT() 함수를 호출한다. 함수가 호출되면 희생 블록 선택 정책에 의해 희생 블록이 선택된다. 선택된 블록이 00bit 값을 가진 블록일 경우, NorBlk에 페이지가 카피백된다. NorBlk은 카피백된 페이지를 배치할 01bit 값을 가진 normal 블록을 가리키는 포인터이다. 선택된 블록이 bit값이 01인 normal 블록이거나, cold 블록일 경우, ColBlk에 페이지가 카피백된다. ColBlk은 카피백된 페이지를 배치할 cold 블록을 가리키는 포인터이다.

3.2 2R++의 오버헤드

2R++에서 블록을 관리하기 위해 추가된 bit로 인해 오버헤드가 발생한다. 블록 당 2bit의 추가적인 공간을 필요로 하지만 4KB 페이지 1000개로 이루어진 블록의 경우, 4MB당 2bit의 작은 공간만을 필요로 한다.

페이지의 읽기는 페이지의 위치에 영향을 주지 않으며, 쓰기는 페이지의 위치의 변경을 유발하지만 항상 normal block에 쓰기를 진행하기 때문에 bit에 접근하지 않는다.

새로운 블록을 할당하는 경우와 가비지 컬렉션 단계에서만 bit에 대한 접근이 발생한다. Fig. 7의 CurBlk, NorBlk, ColBlk이 가리키는 블록에 빈 공간이 존재하지 않아 새로운 빈 블록을 할당하는 경우, 그 용도에 맞게 bit를 설정하는 과정을 거친다. 가비지 컬렉션을 진행하는 도중 여러 희생 블록을 선택하는 과정에서 첫 번째 희생 블록과 동일한 region의 희생 블록을 선택해야하기 때문에 각 블록의 bit를 확인해야 한다. 마지막으로 희생 블록의 선택이 완료된 이후, 각 페이지들이 위치한 블록에 따라 카피백이 진행되는 위치가 달라지기 때문에 이를 확인하기 위해 bit의 값을 확인하는 과정을 가진다.

블록의 bit로의 접근은 쓰기를 위한 새로운 블록을 할당하

는 과정을 제외하고 모두 가비지 컬렉션 과정에서 발생한다. 또한, 각 과정들은 O(1)의 시간 복잡도를 가지기 때문에 적은 오버헤드만으로 2R++를 구현할 수 있다.

4. 실험 환경

4.1 FEMU

실험은 FEMU 환경에서 수행되었다. FEMU는 플래시 메모리 연구를 목적으로 개발된 QEMU 기반의 플래시 에뮬레이터이다[4, 5]. FEMU는 실제 물리 디스크에 데이터를 저장하는 QEMU와 달리 메모리에서 I/O 작업을 수행한다. FEMU는 메모리에 데이터를 저장하고 읽기, 쓰기, 지우기 속도를 변수로 관리하여 물리적 장치의 성능에 구애받지 않고 실험을 수행할 수 있다.

FEMU는 기존 플래시 메모리와 달리 블록 단위가 아닌 라인 단위로 지우기가 발생한다. 플래시 메모리 저장장치는 여러 칩들의 집합으로 구성되며, 칩들은 다시 LUN의 집합으로 구성된다. LUN은 플레인이 모여 구성되며, 플레인은 블록 단위의 집합으로, 블록은 쓰기, 읽기의 단위인 페이지로 구성된다. 라인은 각 플레인의 동일한 위치의 블록들을 묶은 것으로 Fig. 8에 그림으로 나타내었다. 라인 단위의 지우기 동작은 블록 단위의 그것과 동일하게 동작한다.

실험은 FEMU 환경에서 제공하는 Blackbox SSD의 FTL을 수정하여 진행되었으며, Blackbox SSD는 4KB의 페이지 사이즈를 가진다. 라인 당 페이지 수와 라인의 수는 SSD의 용량에 따라 달라지며, 실험에서 사용된 8GB와 10GB의 페이지와 라인의 구성에 대해 Table 1에 나타내었다.

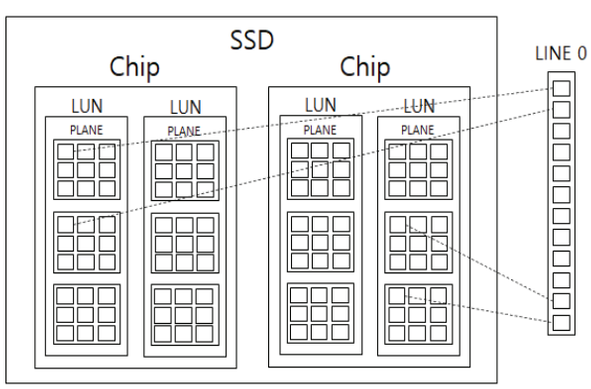

Fig. 8. Structure of FEMU SSD

Table 1. Structure of SSD

	Pages per Line	Total Lines	Total Pages
8GB	1,152	2,048	2,359,296
10GB	1,024	2,816	2,883,584

Table 2. Experiment Setup

	Size of SSD	Database Size	Total Write	OP size(%)
fio	8GB	8GB	360GB	10
TPC-C	10GB	1GB to 10GB	240GB	10
Linkbench	10GB	1GB to 10GB	2,400GB	10

4.2 fio

fio는 마이크로 벤치마크의 일종으로 본 실험에서는 skewness 설정을 위해 zipf distribution 옵션을 사용하여 실험을 진행하였다[6]. 실험은 8GB의 파일 사이즈에서 9천만 번의 쓰기(총 360GB 쓰기)를 진행하였다. 사용된 SSD의 크기는 8GB이며, 10%의 추가 OP 영역이 할당되었다(Table 2).

4.3 OLTP 벤치마크

본 논문에서는 대표적인 OLTP 벤치마크인 TPC-C와 Linkbench에서 실험을 진행하였다. 실험 환경에 대해서는 Table 2에 정리하였다.

1) TPC-C

TPC-C는 전자 상거래를 진행하면서 발생하는 DBMS 상의 트랜잭션 처리 과정을 시뮬레이션하는 벤치마크이다[2]. 본 실험에서는 FEMU 환경에서 4KB 페이지에 대한 6천만번의 쓰기(총 240GB 쓰기)를 진행했고, 데이터베이스의 사이즈는 1GB에서 10GB까지 증가하였다. 사용된 SSD의 크기는 10GB이며, 추가로 10%의 OP 영역을 할당했다.

2) Linkbench

Linkbench는 페이스북에서 제공하는 소셜 그래프 모델링에 대한 벤치마크이다[7, 8]. 본 실험에서는 FEMU 환경에서 4KB 페이지에 대한 6억 번의 쓰기(총 2400GB 쓰기)를 진행했고, 데이터베이스의 사이즈는 1GB에서 10GB까지 증가하였다. 사용된 SSD의 크기는 10GB이며, 10%의 추가 OP 영역이 존재한다.

5. 결 과

실험은 fio, TPC-C, Linkbench에 대해 각각 그리디, 2R, 2R++ FTL을 사용하여 진행되었다.

Fig. 9~11은 각각의 실험 결과에 대한 running WAF 그래프이다. WAF는 쓰기 증폭 계수로 요청한 쓰기와 가비지 컬렉션으로 발생한 카피백 쓰기를 더한 실제 쓰기 수를 요청한 쓰기 수로 나눈 값이다. 가장 이상적인 상황에서는 요청한 만큼의 쓰기만 실제 쓰기로 이어지며, WAF는 1의 값을 가진다. Running WAF는 구간별로 구한 WAF의 값으로 본 실험에서는 벤치마크의 진행을 10등분하여 각 구간마다의 실제 쓰기 수와 요청한 쓰기 수만을 이용해 running WAF를 구하였다.

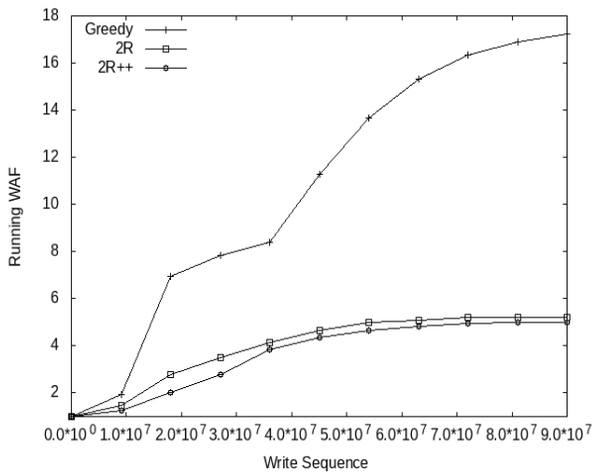

Fig. 9A. Running WAF of fio

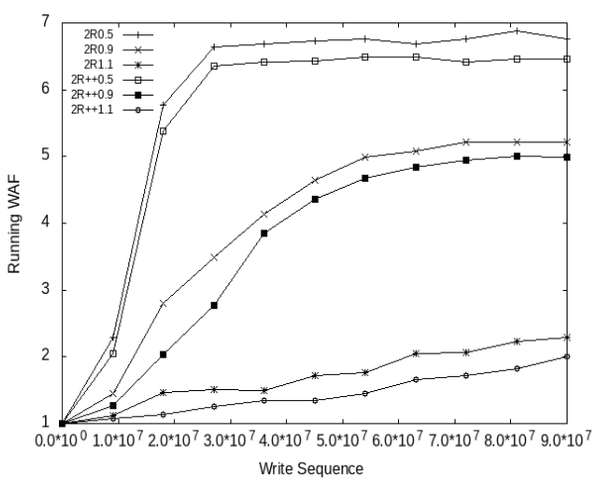

Fig. 9B. Running WAF of fio with Different Zipf Distribution

5.1 fio 실험결과 분석

Fig. 9A와 Fig. 9B는 fio의 실험 결과로 Fig. 9A는 zipf distribution의 값이 0.9인 경우에 세 가지 FTL에 대한 실험 결과이다. Fig. 9A에서 fio는 공간이 모두 사용 중인 상태로 실험이 진행되기 때문에 그리디의 경우 WAF가 초기부터 크게 증가하는 것을 확인할 수 있다. Fig. 9B는 zipf distribution의 값이 0.5, 0.9, 1.1인 경우에 대해 2R과 2R++에 대해 실험을 진행한 그래프이다. 벤치마크의 시작부터 종료까지 측정한 최종 누적 WAF와 2R과 비교한 2R++의 개선율을 Table 3에 나타내었다. 개선율에 대한 공식은 Equation (1)에 나타내었다. WAF의 값에서 1을 빼 WAF의 기본 값인 1을 개선율을 구하는 공식에서 제외하였다.

zipf 값 0.5에서 개선율은 6.126%, 0.9에서 12.255%, 1.1에서 60.383%로 skewness가 커질수록 2R++에 의한 WAF의 개선율이 높아짐을 확인할 수 있다. Skewness가 증가하면 접근 빈도가 높은 hot 페이지의 수는 줄어들고 접근

Table 3. Cumulative WAF of fio

	2R	2R++	Improvement Rate of 2R++(%)
fio(zipf = 0.5)	6.196	5.896	6.126
fio(zipf = 0.9)	4.223	3.871	12.255
fio(zipf = 1.1)	1.771	1.481	60.383

$$\text{개선율} = \frac{(WAFof2R - WAFof2R{+}{+})}{(WAFof2R{+}{+} - 1)} \quad (1)$$

빈도가 낮은 warm/cold 페이지의 수는 증가한다. Warm 페이지의 증가는 지역 오염 문제를 유발하기 때문에 skewness가 증가할수록 2R++의 개선율이 높아진다.

5.2 TPC-C 실험결과 분석

Fig. 10A와 Fig. 10B는 TPC-C에 대한 그래프로 Fig. 10B는 Fig. 10A의 결과에서 2R과 2R++에 대한 결과만을

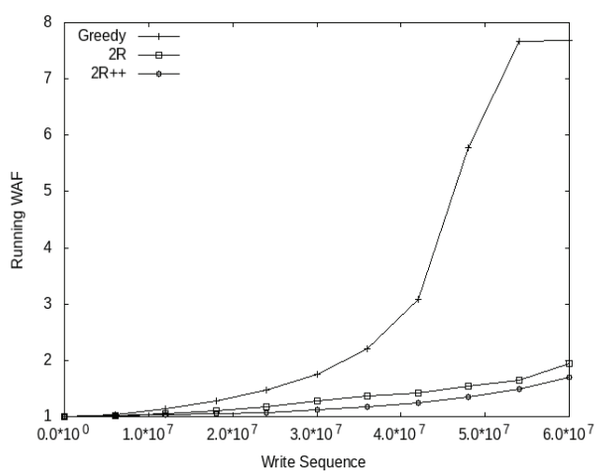

Fig. 10A. Running WAF of TPC-C

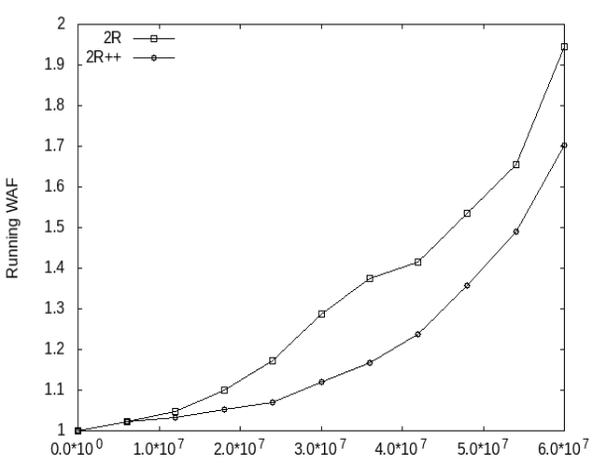

Fig. 10B. Running WAF of TPC-C in Detail

Table 4. Cumulative WAF of OLTP

	Greedy	2R	2R++
TPC-C	3.286	1.356	1.225
Linkbench	3.379	1.217	1.19

확대한 그래프이다. TPC-C는 데이터베이스의 사이즈가 점차 증가하는 벤치마크로 초기에는 세 FTL 모두 원만한 WAF의 증가를 보여주지만 데이터베이스가 저장 용량의 과반을 사용하는 시점부터 그리디의 WAF가 급격하게 증가하는 것을 볼 수 있다. 2R의 지역 오염 문제를 해결하는 2R++에 의한 성능 개선 효과가 가장 잘 드러난다. 벤치마크의 시작부터 종료 시점까지 측정한 최종 누적 WAF는 Table 4에 나타내었다. 2R++의 WAF 값은 그리디와 2R의 WAF와 비교해서 923.513%, 57.815%의 개선율을 보여준다.

5.3 Linkbench 실험결과 분석

Fig. 11A와 Fig. 1B는 Linkbench에 대한 실험 결과이며 Fig. 11B는 Fig. 11A의 결과에서 2R과 2R++에 대한 결과만을 확대한 그래프이다. Linkbench는 소셜 그래프 모델링에 관한 벤치마크로 데이터의 확장보다는 업데이트가 주 동작이기 때문에 데이터베이스가 10GB까지 성장하는데 TPC-C의 10배에 가까운 쓰기가 발생했다. 벤치마크의 초기에는 2R++의 running WAF가 낮지만 결국 2R과 비슷해진다. 수많은 업데이트는 normal region과 cold region의 경계를 모호하게 만들고 한 번의 카피백으로는 제대로 된 cold region을 분리하기 어렵게 만들기 때문이다. 벤치마크의 시작부터 종료 시점까지 측정한 최종 누적 WAF는 Table 4에 나타내었다. 2R++의 WAF 값은 그리디와 2R의 WAF와 비교해서 1149.385%, 13.836%의 개선율을 가진다.

5.4 전체 분석

Fig. 12는 카피백을 통해 cold region으로 배치된 페이지 중 쓰기 요청을 통해 다시 normal region으로 돌아온 페이지의 비율로 TPC-C, Linkbench에서 2R과 2R++에 대한 결과를 나타내었다. TPC-C에서 2R은 cold region의 67.71%의 페이지가 normal region으로 돌아갔으나 2R++에서는 41.6%로 줄어든 것을 확인할 수 있다. Linkbench의 경우 2R의 97.75%에서 2R++의 90.83%로 줄어들었으나, 여전히 90%로 추가적인 개선의 여지가 있음을 확인했다.

2R에서 TPC-C를 수행하는 동안 총 20,411,851번의 카피백이 발생했으며, 2R++에서 발생한 카피백의 수는 12,273,717로 2R 대비 66.3%로 감소되었다. Linkbench에서는 131,538,957에서 116,457,462로 13% 감소되었다.

Fig. 13은 각 지역에서 발생한 카피백이 전체 카피백에서 차지하고 있는 비율을 2R과 2R++에 대해 나타낸 그래프이

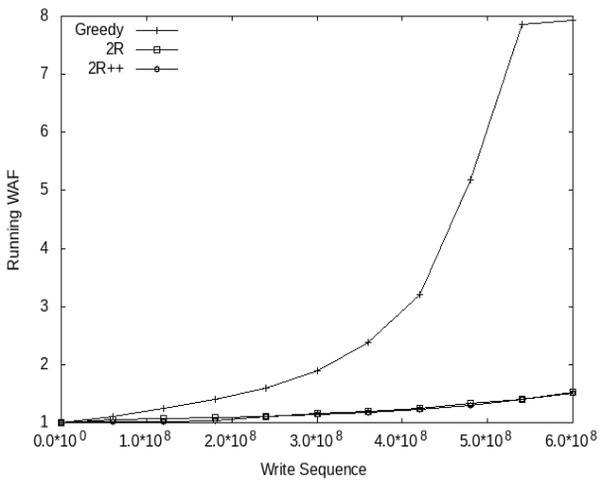

Fig. 11A. Running WAF of Linkbench

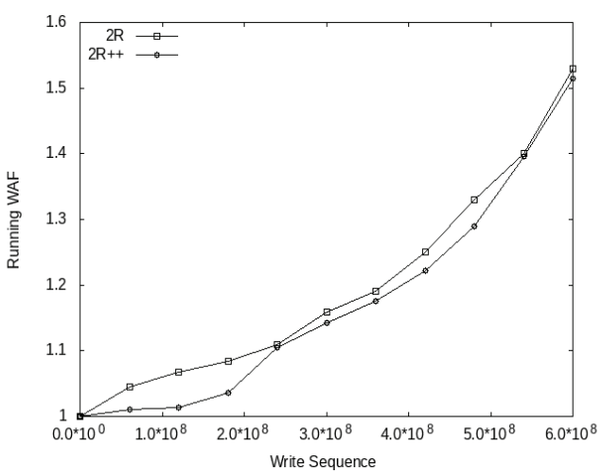

Fig. 11B. Running WAF of Linkbench in Detail

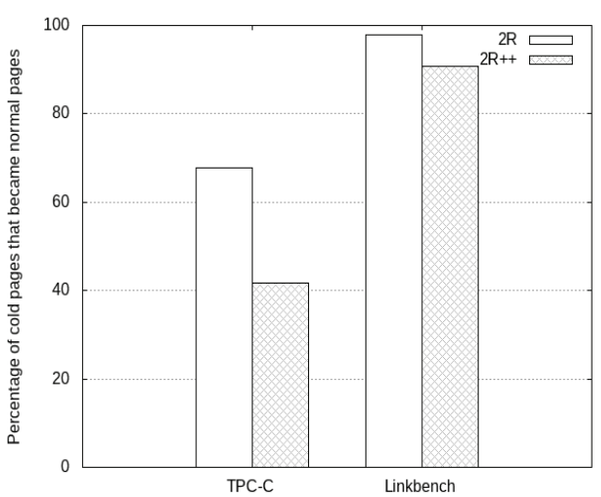

Fig. 12. Ratio of Pages Returned from Cold Region to Normal Region

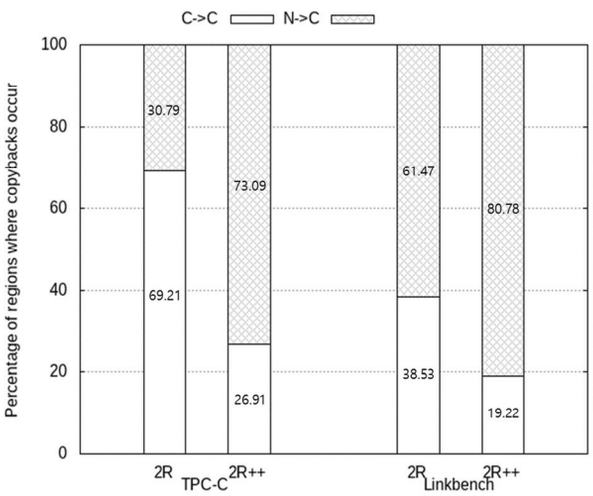

Fig. 13. Copybacks in the Normal Region and the Cold Region Respectively

다. 그래프를 통해 두 벤치마크 모두에서 2R++의 cold region에서 발생하는 카피백이 차지하는 비율이 2R보다 낮아졌음을 확인할 수 있다. 2R과 2R++가 보여주는 비율 변화는 cold 페이지의 제대로 된 분리를 통해 2R++의 설계 목표인 희생 블록으로 선정되면 안 되는 블록의 생성을 만족했음을 보여준다.

6. 관련 연구

SSD에 대한 성능 향상을 목표로 많은 연구가 진행되어왔다. 이 중 페이지의 접근 빈도에 따라 페이지를 나누어 저장하고 관리하는 논문에 대해 논의하고 본 논문의 기법과 비교해본다.

Wei Xie, Yong Chen, Philip C. Roth [9]는 K-means 클러스터링을 사용해 hot, warm, cold 페이지를 분류하는 ASA-FTL을 제안했다. ASA-FTL은 저장 공간을 hot, warm, cold의 세 가지 영역으로 구분하였다. 각 영역의 기준 값은 IRR이라 부르는 페이지의 마지막 접근 간격을 이용해 K-means 클러스터링을 이용해 설정하였다. 램에서 희생 페이지가 결정되었을 때, IRR에 따라 페이지가 저장될 영역을 정하고, K-means 클러스터링을 통해 그 영역의 기준 값을 변경한다.

이 과정을 위해 페이지별로 램과 SSD에 IRR 값과 마지막 접근 시기를 기록하는 추가 공간이 필요하고 주기적으로 K-means 클러스터링을 통해 영역의 기준 값을 변경하는 과정이 필요하다.

2R++는 블록 단위로 데이터의 hotness를 결정하기 때문에 블록 당 bit 단위의 추가 용량을 가진다. 또한, 페이지를 저장할 영역은 기존의 쓰기, 지우기 과정에서 결정되기 때문에 시간적으로도 더 적은 오버헤드를 가지고 있다는 장점을 지니고 있다.

7. 결 론

OLTP의 워크로드를 반영하여 가비지 컬렉션 기법을 개선한 2R에서 지역 오염에 의한 문제를 발견했다. 본 논문에서는 cold region으로 배치될 때까지 추가적인 기회를 부여함으로써 지역 오염 문제를 해결하는 2R++를 제안하였다. 실험 결과는 2R++가 충분히 2R의 지역 오염 문제를 해결할 수 있음을 보여주었으며, 결과적으로 WAF 또한 개선되었음을 확인할 수 있었다. 향후 연구에서는 수많은 업데이트로 인해 지역의 경계가 모호해지는 경우에도 cold region을 정확하게 분리할 수 있는 방법에 대해 연구할 예정이다. 또한 FEMU 외에도 Open channel SSD와 같은 실제 하드웨어 SSD에서의 구현도 향후 연구 주제이다.

References

[1] M. Kang, S. Choi, G. Oh, and S. W. Lee, "2R: Efficiently isolating cold pages in flash storages," *Proceedings of the VLDB Endowment*, Vol.13, No.12, pp.2004-2017, 2020.

[2] S. T. Leutenegger and D. Dias, "A modeling study of the TPC-C benchmark," *ACM Sigmod Record*, Vol.22, No.2, pp.22-31, 1993.

[3] A. Jaleel, K. B. Theobald, S. C. Steely Jr, and J. Emer, "High performance cache replacement using re-reference interval prediction (RRIP)," *ACM SIGARCH Computer Architecture News*, Vol.38, No.3, pp.60-71, 2010.

[4] H. Li, M. Hao, M. H. Tong, S. Sundararaman, M. Bjørling, and H. S. Gunawi, "The {CASE} of {FEMU}: Cheap, accurate, scalable and extensible flash emulator," In *16th USENIX Conference on File and Storage Technologies (FAST 18)* (pp. 83-90), 2018.

[5] Li Huaicheng. FEMU: Accurate, Scalable and Extensible NVMe SSD Emulator (FAST'18) [Internet], https://github.com/vtess/FEMU, 2018.

[6] J. Axboe. FIO (Flexible IO Tester). [Internet], http://git.kernel.dk/?p=fio.git;a=summary, 2006.

[7] T. G. Armstrong. Facebook Graph Benchmark. [Internet], https://github.com/facebookarchive/linkbench, 2013.

[8] T. G. Armstrong, V. Ponnekanti, D. Borthakur, and M. "Callaghan, Linkbench: A database benchmark based on the facebook social graph," In *Proceedings of the 2013 ACM SIGMOD International Conference on Management of Data*, pp.1185-1196, 2013.

[9] W. Xie, Y. Chen, and P. C. Roth, "ASA-FTL: An adaptive separation aware flash translation layer for solid state drives," *Parallel Computing*, Vol.61, pp.3-17, 2017.

안 효 준

https://orcid.org/0000-0003-4627-9054
e-mail : 97aahhjj@naver.com
2021년 성균관대학교 소프트웨어학과(학사)
2021년 ~ 현 재 성균관대학교
소프트웨어학과 석사과정
관심분야 : 플래시 메모리 내 FTL

이 상 원

https://orcid.org/0000-0002-4206-3718
e-mail : swlee@skku.edu
1991년 서울대학교 컴퓨터공학과(학사)
1994년 서울대학교 컴퓨터공학과(석사)
1999년 서울대학교 컴퓨터공학과(박사)
1999년 ~ 2001년 한국오라클 기술과장
2001년 ~ 2002년 이화여자대학교 BK21 계약교수
2002년 ~ 2015년 성균관대학교 정보통신대학 교수
2015년 ~ 현 재 성균관대학교 소프트웨어융합대학 교수
관심분야 : 플래시 메모리 기반 DBMS