JDCS
디지털콘텐츠학회논문지
Journal of Digital Contents Society
Vol. 25, No. 10, pp. 2863-2870, Oct. 2024

Check for updates

SSD 내구성 강화를 위한 Window 기반 오프라인 Hot/Cold 데이터 분류 기법

백 승 수[1] · 차 재 혁[2*]
[1]한양대학교 컴퓨터소프트웨어학과 석박사통합과정
[2]한양대학교 컴퓨터소프트웨어학부 교수

Offline Data Placement using Window-based Hot/Cold Identification to Enhance Solid-state Drive Endurance

Seungsu Baik[1] · Jaehyuk Cha[2*]

[1]Ph.D. Candidate, Department of Computer Science, Hanyang University, Seoul 04565, Korea
[2]Professor, Department of Computer Science, Hanyang University, Seoul 04565, Korea

[요 약]

SSD와 같은 낸드 플래시 메모리 기반 저장 장치는 블록 저장 장치로 사용되기 위해 플래시 변환 계층 (FTL; flash translation layer)이라는 소프트웨어 계층을 가진다. 이 FTL은 데이터 배치 정책, 주소 사상, 마모도 평준화, 가비지 컬렉션과 같은 다양한 기능을 가지고, 이 기능들은 저장 장치의 성능에 많은 영향을 끼친다. 따라서 지금까지 다양한 연구는 FTL의 각 기능의 성능을 향상하는 기법을 제안했다. 본 논문은 낸드 플래시 메모리에서 발생하는 쓰기증폭(WA; write amplification)을 최소화하기 위한 오프라인 데이터 배치 정책을 제안한다. 제안하는 데이터 배치 정책은 트레이스 데이터를 분석하여 페이지의 미래 재 요청 여부에 따라 Hot과 Cold 페이지를 구분하고, 윈도우 크기를 조절해 효율적인 데이터 배치를 수행한다. 또한, 트레이스를 chunk로 분할하여 각 chunk별로 최적의 윈도우 크기를 찾고, 이를 통해 쓰기증폭계수(WAF; write amplification factor)를 1에 가깝게 유지하면서 제한된 자원과 시간 내에 데이터 배치 결과를 도출한다.

[Abstract]

NAND flash memory-based storage devices, such as solid-state drives (SSDs), use a software layer called the flash translation layer (FTL) to function as block storage devices. The FTL provides various functions, including data placement policies, address mapping, wear leveling, and garbage collection, all of which significantly impact the performance of the storage device. Accordingly, various studies have proposed techniques to improve the performance of each FTL function. This paper proposes an offline data placement policy to minimize write amplification in NAND flash memory. The proposed data placement policy analyzes trace data to distinguish between hot and cold pages based on the likelihood of future re-access and adjusts the window size to perform efficient data placement. Additionally, the trace is divided into chunks, and the optimal window size for each chunk is determined. Thus, the write amplification factor is kept close to 1 to produce optimal data placement results under the constraints of limited resources and time.

색인어 : 플래시메모리, 플래시 변환 계층, 데이터배치, 가비지콜렉션, 쓰기증폭
Keyword : Flash Memory, Flash Translation Layer, Data Placement, Garbage Collection, Write Amplification

http://dx.doi.org/10.9728/dcs.2024.25.10.2863

Received 25 September 2024; **Revised** 24 October 2024
Accepted 24 October 2024

***Corresponding Author; Jaehyuk Cha**

Tel: +82-2-2220-4458
E-mail: chajh@hanyang.ac.kr

Ⅰ. 서 론

SSD(sold state drive), eMMC(embedded multi media card), UFS(universal flash storage) 등 2차 기억 장치로 널리 사용되는 낸드 플래시 메모리는 기존 하드디스크와 다른 중요한 하드웨어적인 특성을 가진다. 플래시 메모리는 섹터 단위가 아닌 페이지 단위(일반적으로 4KB 이상)로 읽고 쓰고, 덮어쓰기를 허용하지 않기 때문에 업데이트시 데이터를 지우고 다시 써야 한다. 하지만, 플래시 메모리의 데이터를 읽고 쓰는 단위는 페이지 단위인 반면, 데이터를 지우는 단위는 페이지의 집합인 블록 단위여서 in-place 업데이트 시 쓰기-전-지우기를 해야 한다. 블록은 64, 128, 256개 혹은 그 이상의 페이지로 구성된다. 읽기/쓰기와 지우기의 단위가 달라서 쓰기-전-지우기 과정을 수행할 때, 같은 블록 내의 모든 페이지를 지우기 때문에 지우기 작업 뿐 만 아니라, 유효한 페이지들은 메모리에 복사한 후 다시 쓰는 과정을 동반한다. 플래시 메모리의 쓰기는 읽기보다 느리고, 지우기는 쓰기보다 더욱 느리다. 이러한 기본 작업(읽기/쓰기/지우기)의 비대칭적 특성과 불필요한 유효페이지 읽기, 쓰기 작업 때문에, 쓰기-전-지우기 과정은 매우 큰 지연시간을 유발한다[1],[2].

이를 해결하기 위해, 플래시 메모리는 OOP(out-of- place) 업데이트를 한다. OOP 업데이트로 기존에 쓰였던 페이지는 무효 페이지가 되고, 새로운 페이지를 확보하기 위해서는 이러한 무효 페이지들을 정리해야 한다. 결국, 플래시 메모리는 무효 페이지를 정리하기 위한 가비지 컬렉션(GC; garbage collection)을 필요로 하게 되며, 이는 많은 시간을 소모하며, 추가적인 쓰기를 발생시켜, 이로 인한 쓰기 증폭(WA; write amplification)이 발생한다. 이러한 쓰기증폭은 SSD의 성능과 수명을 저하시킨다[3],[4].

쓰기 증폭을 최소화하기 위해, 데이터를 효율적으로 배치하여 불필요한 유효페이지 복사를 최소화 하는 데이터 배치 전략에 대한 많은 연구가 진행되고 있다[5]-[7]. 이러한 데이터 배치의 최종 목표는 미래의 유사한 시점에 무효화될 페이지들을 동일한 블록에 저장하여 WA를 줄이는 것이다. 이는 다양한 워크로드 환경에서 SSD의 성능을 향상시키고, 수명을 연장하는 데 중요한 요소로 작용한다.

그러나 이러한 목표를 달성하는 과정에서 여러 도전 과제가 존재한다. 특히, 온라인 알고리즘에서 데이터의 과거 기록을 기반으로 미래를 예측하는 것은 매우 어렵다. 워크로드의 특성이 시간에 따라 변화하고 예측하기 어려운 경우가 많기 때문에, 최적의 데이터 배치를 실현하는 것은 복잡한 문제로 남아 있다. 더불어, 이러한 온라인 알고리즘들이 제안하는 데이터 배치가 실제로 올바른 선택인지, 즉 미래의 쓰기 증폭을 효과적으로 줄일 수 있는지 판단하는 것 또한 큰 난관이다.

WA를 최소화하기 위해서는 더 나은 수명 예측 모델과 알고리즘 개발이 필요하며, 이러한 알고리즘이 제안하는 데이터 배치 전략의 효과를 정확하게 평가할 수 있는 최적의 데이터 배치 방법에 대한 연구도 중요합니다. 이를 통해 워크로드에 최적화된 데이터 배치를 할 수 있는 알고리즘 개발이 가능해지고, SSD의 전반적인 효율성을 크게 향상시킬 수 있다.

그러나 트레이스 데이터를 통해 최적의 데이터 배치 방법을 찾기 위해서는 모든 페이지 할당의 경우를 전부 확인해야 한다. 이론적으로는 가능한 접근법이지만, 현실에서는 모든 페이지 할당을 일일이 확인하는 것이 매우 어렵다. 트레이스의 쓰기 요청 수가 매우 적은 경우에는 가능하겠지만, 대부분의 트레이스에서 페이지 할당의 모든 경우를 확인하는 방법은 현실적으로 제한된 시간 내에 풀 수 없는 문제이다[8].

따라서 이러한 한계를 극복하기 위해, 본 논문에서는 제한된 환경에서 1에 가까운 쓰기증폭계수(WAF; write amplification factor)를 달성할 수 있는 오프라인 데이터 배치 정책을 제안한다. 이 정책은 모든 가능한 페이지 할당을 고려하지 않으면서도, 최대한 효율적인 데이터 배치를 통해 SSD의 성능과 수명을 극대화하는 것을 목표로 한다. 이를 통해, 현실적인 제약(시간, 컴퓨팅 자원) 속에서도 WAF가 1에 근접한 데이터 배치 방법을 제안한다.

본 논문에서 제안하는 오프라인 데이터 배치 정책은 제한된 자원과 시간 내에서 효율적으로 데이터를 배치하여 쓰기 증폭을 최소화하려는 새로운 접근법을 제시한다. 제안하는 데이터 배치 정책은 트레이스 데이터를 미리 분석해, 각 쓰기 요청이 미래 특정 시점 이내에 재 요청 여부를 바탕으로 Hot과 Cold 페이지를 구분한다. 이 방식은 기존의 온라인 방식처럼 모든 요청을 실시간으로 처리하는 것이 아니라, 오프라인 환경에서 알 수 있는 미래의 요청을 고려하여 제한된 자원과 시간 내 최적의 데이터 배치를 수행한다는 점에서 차별점을 둔다.

특히, 이 정책은 윈도우 크기라는 개념을 도입하여, 일정한 시간 범위 내에서 페이지의 사용 빈도를 분석하고 이에 따라 Hot과 Cold 페이지를 동적으로 정의한다. 윈도우 크기가 작으면 Cold 페이지가 증가하고, 크면 Hot 페이지가 증가하는데, 이러한 특성을 바탕으로 다양한 윈도우 크기에 대한 실험을 통해 최적의 성능을 도출하였다. 실험 결과를 통해 트레이스마다 최적의 윈도우 크기가 다름을 확인하였으며, 이를 통해 데이터 배치를 최적화한다.

또한, 트레이스를 여러 chunk로 분할해 각 chunk별로 독립적인 최적의 윈도우 크기를 찾고, 이를 바탕으로 전체 트레이스에 대한 시뮬레이션을 실행하는 방식을 통해 . 이를 통해 WAF를 1에 근접하도록 쓰기 증폭을 최소화하면서도 제한된 자원과 시간 내에 결과 도출이 가능한 현실적인 데이터 배치 방법을 제시하였다.

논문의 구성은 다음과 같다. 2장에서는 낸드 플래시 메모리의 특성과 WAF에 대한 연구 배경을 설명하며, 데이터 배치 전략과 관련된 기존 연구를 소개한다. 3장에서는 제안하는 오프라인 알고리즘을 자세히 설명한다. 이 장에서는 알고리즘의 설계 원리와 동작 메커니즘에 대해 다룬다. 4장에서는 제안한 알고리즘의 성능을 검증하기 위한 실험 평가를 수행하

며, 다양한 워크로드 환경에서의 성능 결과를 분석한다. 마지막으로, 5장에서는 본 연구의 결론을 제시하고, 연구의 한계와 향후 연구 방향에 대해 논의한다.

II. 연구 배경 및 관련 연구

2-1 낸드 플래시 메모리와 쓰기증폭계수

낸드 플래시 메모리는 기존 하드 디스크와 다른 몇 가지 특성을 가지고 있어, 이와 관련된 저장 장치의 데이터 관리 방식도 다르다[1],[2]. 이러한 특성들은 저장 장치의 성능과 수명에 중요한 영향을 미치며, 효율적인 관리 방법이 필요하다.

첫째, 읽기/쓰기 단위의 차이가 있다. 기존 하드 디스크는 섹터 단위로 데이터를 읽고 쓰지만, 낸드 플래시 메모리는 페이지 단위(보통 4KB 또는 8KB)로 데이터를 처리한다. 이는 데이터를 기록하고 관리하는 방식에 큰 차이를 만든다.

둘째, 덮어쓰기가 불가능하다는 점이다. 하드 디스크는 동일한 위치에 데이터를 덮어쓸 수 있지만, 낸드 플래시 메모리는 기존 데이터를 지운 후에만 새로운 데이터를 다시 쓸 수 있다. 또한, 지우기 동작은 페이지 단위가 아닌 블록 단위(여러 페이지로 구성)로 이루어지며, 이로 인해 불필요한 쓰기 동작이 발생할 수 있다.

셋째, 수명이 제한적이다. 낸드 플래시 메모리는 각 셀이 특정 횟수 이상 지워지면 에러율이 증가하여 더 이상 사용할 수 없게 된다. 예를 들어, 최근 널리 사용되는 TLC (triple-level cell) 낸드 플래시의 경우 약 3,000회 정도의 지우기 횟수를 견딜 수 있다.

이런 특성들 때문에, 낸드 플래시 메모리 기반 저장 장치에는 데이터를 효율적으로 관리하기 위한 소프트웨어인 플래시 변환 계층(FTL; flash translation layer)가 필요하다. FTL은 데이터의 효율적인 관리와 성능 최적화를 위해 OOP 업데이트 방식을 사용하며, 이는 기존 데이터를 덮어쓰지 않고 새로운 위치에 데이터를 기록하는 방식이다[3].

이 과정에서 발생하는 주요 문제 중 하나가 WA이다. WA는 호스트의 쓰기 요청 외에도, 가비지 컬렉션 등으로 인해 내부적으로 발생하는 추가 쓰기 작업을 의미한다. 이로 인해 실제로 저장 장치에 기록되는 데이터 양이 호스트가 요청한 양보다 많아지며, 이러한 쓰기 증폭을 측정하기 위해 WAF라는 지표가 사용된다. WAF는 실제로 기록된 데이터 양을 호스트 요청 데이터 양으로 나눈 비율로, WAF 값이 클수록 쓰기 증폭이 크다는 것을 의미한다.

WAF를 줄이기 위한 대표적인 방법으로는 버퍼를 활용한 중복 제거, 가비지 컬렉션 시 희생 블록 선정 전략, 효율적인 데이터 배치 전략 등이 있다. 특히, 본 연구에서는 데이터 배치 전략에 중점을 두어, 데이터를 효율적으로 분산시킴으로써 WAF를 최소화하고 저장 장치의 성능을 향상시키는 방법을 제안한다.

이를 바탕으로, 낸드 플래시 메모리의 특성과 WAF의 중요성을 고려한 최적의 데이터 배치 전략을 개발하는 것이 본 논문의 주요 목표이다.

2-2 관련 연구

본 절에서는 페이지 클러스터링과 가비지 컬렉션의 성능을 향상시키기 위한 다양한 기법들을 다룬다.페이지 클러스터링은 유사한 I/O(input/output) 요청을 동일한 블록에 저장하여, 무효화된 데이터가 집중되도록 함으로써 가비지 컬렉션의 효율성을 높이는 기법이다. 이 과정에서 write pointer를 활용하여 데이터를 여러 블록에 분산 저장하는 방법이 연구되고 있으며, 이를 통해 쓰기 빈도에 따른 데이터 배치 정책을 개선하고자 한다.

DAC(dynamic data clustering)은 페이지를 여러 Region으로 나누어 클러스터링하는 기법으로, 요청 페이지의 빈도에 따라 저장 공간을 분리한다. 그러나 빈 공간의 단편화를 최소화하기 위해 Region의 수를 제한하였다[9].

2R(two regions)은 WAF를 줄이기 위해 Cold 페이지를 분리하여 저장하는 정책을 제안한다. 이때 Cold 페이지는 가비지 컬렉션 시 선택된 블록 내의 유효 페이지로 정의되며, 다양한 희생 블록 선정 정책을 통해 보다 정확한 Cold 페이지 분류를 목표로 한다[10].

FADaC(fading average data classifier)는 Workload의 시간적 지역성(temporal locality)에 기반하여 데이터의 Hotness를 정의하고, 이를 바탕으로 효과적인 데이터 배치 정책을 제안한다[11].

이러한 온라인 데이터배치 연구들은 WAF를 줄이고, 저장 장치의 성능과 수명을 향상시키는 데 중점을 두며, 주로 Greedy 정책을 사용하여 유효 페이지가 가장 적은 블록을 희생 블록으로 선정한다.

하지만, 이는 항상 최적의 성능을 보장하지 않는다. 예를 들어, 선택된 블록 내의 유효 페이지가 곧 재 요청될 경우 불필요한 복사가 발생하여 WAF가 증가할 수 있다. 따라서 유효 페이지가 적은 블록을 희생 블록으로 선정하는 것은 최적의 희생 블록 선정 방법이 아니다.

최적의 희생 블록 선정을 위해, Shafaei et al.은 몬테카를로 트리 탐색(MCTS; monte carlo tree search)을 활용하여 단일 Write pointer 환경에서 효율적으로 희생블록을 선정하는 Near-Optimal 희생 블록 선정 방법을 제안하였다[8]. MCTS는 모든 경우의 수를 테스트하는 대신, 여러 가능성을 무작위로 탐색하고 그 u결과를 트리 구조로 평가하여 최적에 가까운 결과를 도출한다. 이 과정은 선택, 확장, 시뮬레이션, 역전파의 네 단계로 이루어지며, 반복적인 시뮬레이션을 통해 성능을 평가하고 최적화된 희생 블록을 선택한다. 이 연구는 데이터 배치를 전혀 고려하지 않고, 가비지 컬렉션 중 희생 블록 선정 과정에만 집중하였다.

2-3 연구 동기

앞서 살펴본 온라인 환경에서의 연구들과 달리, 오프라인 환경에서는 모든 I/O 요청을 미리 알고 있지만, 그럼에도 불구하고 모든 가능한 경우를 실험하는 것은 시간적 제약으로 인해 현실적으로 어려운 문제이다.

이를 해결하기 위해, 전체 트레이스를 분석하여 모든 요청에 대하여 페이지 무효화 순서를 파악한 후, 무효화 순서를 기반으로 비슷한 시점에 무효화될 페이지들을 그룹화하여 동일한 물리 블록에 할당하는 방식으로 시뮬레이션하였다. 그 결과, 무효화 시점이 비슷한 페이지들이 같은 블록으로 할당되어 유효페이지 복사를 완전히 제거한 최적의 데이터 배치를 찾을 수 있었으나, 많은 활성 블록을 생성하여 블록사용량이 급증하는 문제가 있었다. 이는 물리 블록 수가 한정적인 실제 환경에서 무효화 시점이 다른 페이지들이 여러 너무 많은 활성 블록을 생성하면서 물리 블록이 부족한 상황이 발생하게 된다. 이는 여유 공간이 존재하더라도 새로운 블록을 할당할 수 없게 된다.

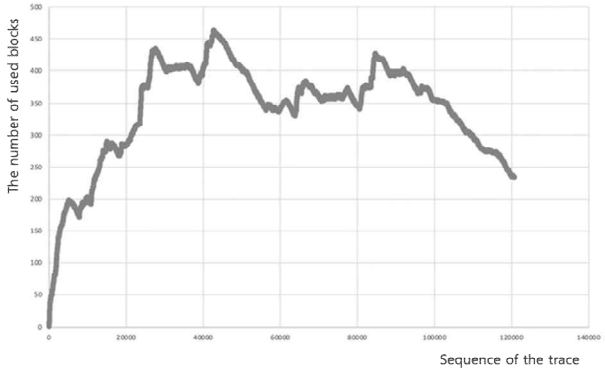

그림 1. 트레이스 순번에 따른 무효화 시점 기반 페이지 할당 시 블록 사용량 변화

Fig. 1. Changes in block usage based on invalidation timing in page allocation according to the trace sequence numbers

그림 1은 무효화 시점을 기반으로 페이지를 할당하여 실제 블록 사용량을 트레이스 순번에 따라 나타낸 것이다. 블록의 모든 페이지가 무효화 되면, 이를 바로 지워, 블록 사용량을 감소시킨다. X축은 트레이스의 순번을 의미하고 Y축은 사용된 블록 수를 나타낸다. 시작 지점에서 확인할 수 있듯이 사용 블록 수가 급속도로 증가한다. 이는 초반에는 페이지들의 재사용 빈도가 많지 않다는 것을 의미한다. 즉, 재 요청 간의 간격이 길다는 의미이다. 따라서 초반에 여러 블록에 나뉘어 쓰인 후, 43000번째 쓰기에 최대 블록 수를 달성한 후 블록 수가 감소하게 된다.

본 연구에서는 현실적인 물리 블록 수와 제한된 활성 블록 환경에서 WAF를 1에 가깝게 달성할 수 있는 최적의 오프라인 데이터 배치 정책을 연구하고자 한다.

Ⅲ. 제안 방법

이 장에서는 제한된 전체 블록 개수와 활성블록 개수 내에서 1에 가까운 WAF를 달성하면서 현실적인 오프라인 데이터 배치 정책을 제안한다. 앞서 2-3절에서 설명한 것과 같이 지난 요청들의 쓰기패턴 분석을 통한 온라인 데이터 배치 정책과 달리, 오프라인은 모든 요청을 미리 알 수 있다. 주어진 트레이스에서 각 페이지 요청을 분석하여 사용빈도가 높은 핫(Hot) 페이지와 사용빈도가 적은 콜드(Cold) 페이지로 명확하게 구분하고, 효율적인 데이터 배치가 가능하다. 하지만, 데이터 배치에 대한 모든 경우의 수를 시뮬레이션하는 방법은 현실적으로 제한된 시간 내에 실행이 어렵다. 미리 계산된 무효화 시점을 기반한 데이터 배치 방법은 매우 높은 물리 블록의 사용량으로 비현실적이다. 따라서, 본 논문에서는 하나의 쓰기 요청에 대하여 이후 발생할 쓰기요청 윈도우 내에 재요청 여부에 따라 Hot과 Cold를 새롭게 정의하고, 이를 기반으로 새로운 오프라인 데이터 할당 정책을 제안한다.

제안하는 데이터 배치 정책은 미래의 요청에 대한 윈도우를 기반으로 Hot과 Cold를 구분하기 때문에, 윈도우 크기에 따라 Hot과 Cold의 비율이 달라진다. 이 비율은 WAF에 직접적인 영향을 미친다. 윈도우 크기가 너무 작으면 대부분의 요청이 Cold로 정의되어 유효페이지가 많은 Cold 블록의 증가로 WAF가 높아진다. 반면에, 윈도우 크기가 너무 크면 대부분의 요청이 Hot으로 정의되어 블록의 무효화 시점이 높아져 WAF가 높아진다. 따라서, 오프라인 환경에서 요청의 Hot과 Cold 정의는 윈도우 크기에 의해 결정되며, 모든 경우의 수를 시뮬레이션하는 대신 몇 가지 고정된 윈도우 크기를 적용하여 실험을 수행한다.

표 1. 윈도우 크기에 따른 추가 쓰기 수 변화

Table 1. Changes in the number of additional writes according to the window size

Window Size	# of valid page copies
16,384	13,046
32,768	11,415
65,536	8,551
131,072	2,308
262,144	8,412
524,288	6,081
1,048,576	6,329
2,097,152	6,329
No Clustering	565,927

표 1에서는 SNIA(solid state storage initiative)에서 제공하는 서버 트레이스 중 proj_3 트레이스를 사용하여 실험한 결과를 보여준다. 데이터 할당 정책을 개선한 경우, 그렇지

않은 경우에 비해 성능이 대폭 향상됨을 확인할 수 있다. 또한, 윈도우 크기에 따라 성능이 크게 차이 난다는 점도 주목할 만하다. 실험 결과, 최적의 윈도우 크기는 131,072로 나타났으며, 이는 트레이스에 의존적인 값임을 알 수 있다. 다른 트레이스에서는 최적의 윈도우 크기가 다를 수 있으며, 이는 단일 트레이스 내에서도 각 요청마다 최적의 윈도우 크기가 다를 수 있다는 의미이다. 트레이스는 상호 독립적인 데이터의 집합이기 때문에, 일부 트레이스를 추출하여 새로운 트레이스로 사용할 수 있다.

제안하는 정책은 총 3단계로 이루어진다. 첫 번째 단계는 정책에 사용되는 변수와 낸드 플래시 메모리 및 트레이스에 따른 초기 설정을 구성하는 단계이며, 두 번째 단계는 최적의 윈도우 크기를 찾는 단계이다. 마지막으로, 세 번째 단계는 최적의 윈도우 크기를 활용하여 전체 트레이스를 시뮬레이션하는 단계이다.

- **초기 설정 구성**

첫 번째 단계에서는 트레이스로부터 쓰기 요청만을 추출하고, 쓰기 요청의 최대 주솟값을 확인하여 저장 장치의 용량을 설정한다. 트레이스마다 사용하는 최대 주솟값이 다르기 때문에, 이 과정을 통해 트레이스에 맞는 낸드 플래시 메모리 저장 장치가 설정된다. 예를 들어, 최대 주솟값이 100,000일 경우, 저장장치의 총 페이지 수를 100,000보다 큰 2의 거듭제곱 수인 131,072개로 설정한다.

- **최적의 윈도우 크기 탐색**

다음 단계는 트레이스를 N개의 chunk로 나누고, 각 chunk 별로 최적의 윈도우 크기를 탐색한다. 윈도우 크기가 너무 작으면 모든 트레이스에서 성능이 나쁘기 때문에, 최소 윈도우 크기를 블록 내 페이지 수의 2배로 설정하고, 최소 윈도우 크기부터 chunk 크기의 절반까지 2의 곱연산으로 윈도우 크기를 증가시켜가며 모든 윈도우 크기에 대해 각 chunk 별로 시뮬레이션한다. 모든 chunk에 대하여 각각의 최소의 WAF를 갖는 최적의 윈도우 크기를 찾는다. 이 때, 실제로는 앞선 chunk의 물리 블록 할당 결과에 따라 다음 chunk의 시뮬레이션 결과가 달라질 수 있다. 하지만, 계산 병렬성을 높이기 위해서 각 chunk 마다 독립적으로 평가한다.

- **최적의 윈도우 크기를 활용한 전체 시뮬레이션**

마지막으로는 이전 단계에서 찾은 chunk별 최적의 윈도우 크기를 적용하여 전체 트레이스에 대하여 시뮬레이션을 실행한다.

이와 같은 정책을 통해, 트레이스 특성에 맞는 최적의 데이터 배치 방법을 도출하였고, 이로써 WAF를 최소화하여 저장장치의 성능과 수명을 향상시키는 데 기여하였다.

Ⅳ. 실험 결과

4-1 실험 환경

본 논문에서는 제안된 데이터 배치 정책을 검증하기 위해 Flashsim[12]을 사용한다. Flashsim은 낸드 플래시 메모리 기반의 저장 장치를 시뮬레이션할 수 있는 오픈 소스 도구로, 유저 공간에서 트레이스 기반으로 동작하는 시뮬레이터로, 오프라인 환경에서 데이터 배치 정책을 실험할 수 있는 적합한 도구이다. 특히, 오프라인 데이터 배치 정책의 경우 전체 트레이스를 여러 번 순회하여 최적의 클러스터링을 적용해야 하므로, 이러한 실험을 지원하는 Flashsim을 선택하였다.

본 논문에서는 SNIA에서 제공하는 MSR 트레이스를 사용하여 알고리즘을 검증하였다[13]. 각 트레이스는 읽기와 쓰기 요청을 모두 포함하고 있지만, 본 연구에서는 WAF를 검증하기 위해 쓰기 요청만을 추출하여 사용하였다. 또한, 이 트레이스는 Block I/O 저장 장치에서 추출된 섹터 단위의 데이터로, 이를 페이지 단위로 변환하여 실험을 수행하였다.

표 2. MSR Traces 쓰기 요청 수와 요청 주소에 적절한 SSD 크기

Table 2. The write request counts and appropriate SSD size based on requested addresses from MSR Traces

Trace	Write I/Os	Required SSD Size (GB)
hm_0	2,575,568	16
hm_1	28,415	32
mds_0	1,067,061	64
prn_0	4,983,406	128
rsrch_0	1,300,030	32
stg_0	1,722,478	16
ts_0	1,485,042	32
wdev_2	181,077	64
web_0	1,423,458	64

표 3. SSD 설정

Table 3. SSD configuration

Pages per Block	256
Page size	4,096
Blocks per Plane	Dependent on SSD capacity
Planes per Die	1
Dies per Chip	8
Num Chips	8
Over-provisioning(OP)	10%
GC threshold	When 2 blocks left
# of CHUNKS	10

실험에 사용된 트레이스는 가비지 컬렉션을 유도하기 위해 각 트레이스의 논리 블록 주소(LBA; logical block address)를 분석하여 적절한 SSD 크기를 설정하였다. 실험의 공정성을 유지하기 위해 over-provisioning 영역 등 SSD 설정값은 모두 동일하게 적용하였다.

제안된 데이터 배치 정책의 성능을 평가하기 위해서는 충분한 가비지 컬렉션이 발생해야 한다. 이는 WAF가 성능 지표로 사용되기 때문이다. SSD 시뮬레이터는 초기 실행 시 데이터가 없는 상태에서 시작하기 때문에, 대부분의 트레이스에서는 충분한 가비지 컬렉션이 발생하지 않는다. 따라서, 실험 전에 pre-conditioning 과정이 필요하다. 이 과정은 가비지 컬렉션이 원활히 발생하도록 무작위 데이터를 사전에 쓰는 작업으로, 정규 분포를 따르는 무작위 함수를 사용하여 논리 주소를 생성하고, 이에 따라 데이터를 기록한다. 이 과정에서 전체 물리 공간의 50%에 해당하는 논리 주소 범위를 사용하며, 해당 범위에 대해 SSD 크기의 2배에 달하는 쓰기를 요청한다. 이렇게 하여 물리 공간에 충분한 데이터가 기록되고, 유효하지 않은 페이지가 약 50%에 달하도록 구성하였다. 이와 같은 실험 환경에서 본 논문이 제안한 데이터 배치 정책의 효과를 검증하였다.

4-2 실험 결과

그림 2는 MSR trace 환경에서 오프라인 데이터 배치 정책과 다양한 온라인 데이터 배치 알고리즘의 성능을 비교하고 있다. 온라인 데이터 배치 정책과 비교했을 때, 제안하는 데이터 배치 정책이 가장 작은 WAF를 가진다.

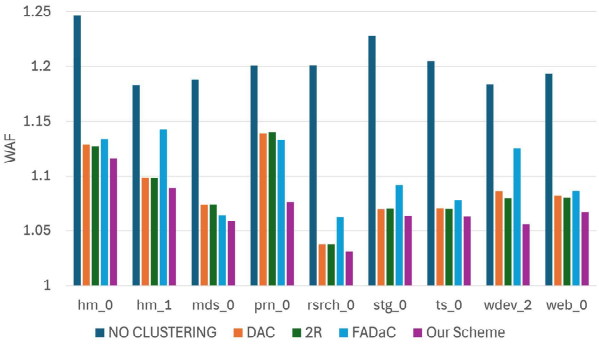

그림 2. 트레이스별 WAF 비교
Fig. 2. Comparison of WAF in the traces

NO CLUSTERING은 클러스터링을 적용하지 않았을 때 데이터 배치가 비효율적임을 나타낸다. 제안하는 데이터 배치 정책은 모든 경우에서 가장 낮은 WAF를 달성했다. 이는 제안된 정책이 다양한 환경에서 매우 효율적으로 작동함을 의미한다. 반면, DAC, 2R, FADaC은 NO CLUSTERING에 비해서는 전반적으로 낮은 WAF를 보이지만, 모든 워크로드에서 일관되게 성능이 좋은 결과를 보이지 않고, 워크로드에 의존적인 결과를 보인다.

결과적으로, 제안된 데이터 배치 정책은 일관된 성능으로 다양한 워크로드에서 SSD의 성능과 수명을 향상시킬 수 있음을 보여준다.

또한, 제안하는 데이터 배치 정책의 결과는 사전에 정의한 chunk에 의해 성능 영향을 받는다. 표 3에 정의된 것처럼 chunk 값을 10으로 고정하였으나, 트레이스 길이 또는 시뮬레이션하는 SSD 크기에 따라 더 많은 chunk 수를 사용하여 더욱 세분화된 실험이 가능하다.

하지만, 제안하는 데이터 배치 정책은 모든 트레이스의 가장 작은 WAF를 가지지만, 가장 이상적인 값인 1이 되지는 않는다. 이는 pre-conditioning 과정에서 생긴 유효페이지 중에서 트레이스에서 전혀 쓰이지 않는 주소를 GC를 통해 Cold로 정리하는 과정에서 발생하는 쓰기인 Non-trace Data copies가 대부분을 차지했다.

표 4. 트레이스의 유효 페이지 복사 분류
Table 4. Classification of valid page copy in traces

Trace	Valid page copies # (A)	Non-trace Data copies # (B)	A - B
hm_0	908,554	905,977	2,577
hm_1	15,120	15,120	0
mds_0	173,570	173,570	0
prn_0	2,166,796	2,166,734	62
rsrch_0	127,634	127,544	90
stg_0	358,038	357,276	762
ts_0	276,319	276,310	9
wdev_2	30,686	30,686	0
web_0	305,776	305,772	4

표 4에 보인 것 같이, 전체 유효 페이지 복사 수에서 Non-trace Data copies 수를 뺀 A-B의 결과 값이 트레이스 데이터에 의한 유효페이지 복사 수이다. 이 값은 트레이스의 쓰기 수에 비하면 현저히 작은 값이다. 이미 쓰여진 데이터 중에서 앞으로 쓰이지 않는 데이터를 정리하여 여유페이지를 확보하는 것은 꼭 필요한 작업이다. 해당 복사를 제외한 복사 수 (A-B)로 WAF를 다시 계산하면, 1에 매우 근접한다. 이는 여유공간 확보를 위한 유효페이지 복사 외의 추가적인 쓰기가 거의 발생하지 않음을 확인할 수 있다.

V. 결 론

SSD와 같은 낸드 플래시 메모리 기반 저장 장치는 성능 향상과 수명 연장을 위한 연구가 매우 활발히 진행되고 있다. 이러한 저장 장치는 페이지 단위로 읽고 쓰지만, 블록 단위로 데이터를 지워야 하는 비대칭적 특성으로 인해 쓰기 증폭 문제가 발생한다. 특히, 쓰기-전-지우기 작업과 가비지 컬렉션

으로 인한 불필요한 유효 페이지 복사 등이 SSD의 성능과 수명에 부정적인 영향을 미친다.

이를 해결하기 위해, 본 논문에서는 오프라인 환경에서 데이터를 효율적으로 배치하여 WA를 최소화하는 정책을 제안하고, 이를 다양한 온라인 알고리즘과 비교하였다. 실험 결과, 제안된 오프라인 데이터 배치 정책은 모든 워크로드에서 가장 낮은 WAF를 기록하였으며, 이는 SSD의 성능과 수명에 긍정적인 영향을 미쳤다. 특히, 데이터를 chunk 단위로 세분화하여 배치함으로써, 쓰기 증폭을 효과적으로 줄일 수 있었고, 다양한 환경에서도 일관된 성능을 유지할 수 있었다.

비록 WAF가 완벽한 1에는 도달하지 못했으나, Non-trace 데이터 복사를 제외한 WAF는 1에 매우 근접한 결과를 보였다. 이는 향후 데이터 배치 알고리즘 최적화 방향에 중요한 시사점을 제공한다. 또한, 제안된 정책은 Write Pointer가 2개 이상인 환경에서 클러스터링을 활용하여 효율적으로 데이터 배치의 경우의 수를 줄임으로써, 제한된 시간과 자원 내에서도 1에 가까운 WAF를 달성하는 성과를 보였다.

본 연구는 오프라인 Hot/Cold 데이터 분류를 활용한 데이터 배치 전략이 SSD의 성능을 개선하고 수명을 연장할 수 있음을 입증하였다. 또한, 향후 연구 방향으로 Hot/Cold 분류 기법을 보다 심층적으로 연구하여, 더욱 다양한 워크로드 환경에서 최적의 데이터 배치를 실현할 수 있는 가능성을 제시하고 있다. 이를 통해, 기존 온라인 데이터 배치 기법의 성능 개선을 위한 가이드라인으로 활용함으로써 SSD의 성능 향상 뿐만 아니라 수명 연장에도 중요한 기여를 할 것으로 기대된다.

감사의 글

본 연구는 2018년도 한국연구재단의 지원에 의하여 이루어진 연구로서, 관계부처에 감사드립니다(2018R1A5A7059549).

참고문헌

[1] N. Agrawal, W. J. Bolosky, J. R. Douceur, and J. R. Lorch, “A Five-Year Study of File-System Metadata,” *ACM Transactions on Storage,* Vol. 3, No. 3, pp. 9-es, October 2007. https://doi.org/10.1145/1288783.1288788

[2] N. Agrawal, V. Prabhakaran, T. Wobber, J. D. Davis, M. Manasse, and R. Panigrahy, “Design Tradeoffs for SSD Performance,” in *Proceedings of USENIX 2008 Annual Technical Conference (ATC ’08)*, Boston: MA, pp. 57-70, June 2008.

[3] K. S. Yim, “A Novel Memory Hierarchy for Flash Memory Based Storage Systems,” *Journal of Semiconductor Technology and Science*, Vol. 5, No. 4, pp. 262-269, December 2005.

[4] W. Bux and I. Iliadis. “Performance of Greedy Garbage Collection in Flash-Based Solid-State Drives,” *Performance Evaluation*, Vol. 67, No. 11, pp.1172-1186, November 2010. https://doi.org/10.1016/j.peva.2010.07.003

[5] J.-U. Kang, J. Hyun, H. Maeng, and S. Cho, “The Multi-Streamed Solid-State Drive,” in *Proceedings of the 6th USENIX Conference on Hot Topics in Storage and File Systems (HotStorage ’14)*, Philadelphia: PA, 13, June 2014.

[6] J. Yang, R. Pandurangan, C. Choi, and V. Balakrishnan, “AutoStream: Automatic Stream Management for Multi-Streamed SSDs,” in *Proceedings of the 10th ACM International Systems and Storage Conference (SYSTOR ’17)*, Haifa, Israel, 3, May 2017. https://doi.org/10.1145/3078468.3078469

[7] M. Bjørling, A. Aghayev, H. Holmberg, A. Ramesh, D. Le Moal, G. R. Ganger, and G. Amvrosiadis, “ZNS: Avoiding the Block Interface Tax for Flash-based SSDs,” in *Proceedings of the 2021 USENIX Annual Technical Conference (USENIX ATC ’21),* Online, pp. 689-703, July 2021.

[8] M. Shafaei and P. Desnoyers, “Near-Optimal Offline Cleaning for Flash-Based SSDs,” in *Proceedings of the 33rd International Conference on Massive Storage Systems and Technology (MSST ’17)*, Santa Clara: CA, May 2017.

[9] M.-L. Chiang, P. C. H. Lee, and R.-C. Chang, “Using Data Clustering to Improve Cleaning Performance for Flash Memory,” *Journal of Software: Practice and Experience,* Vol. 29, No. 3, pp. 267-290, March 1999. https://doi.org/10.1002/(SICI)1097-024X(199903)29:3<267::AID-SPE233>3.0.CO;2-T

[10] 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, August 2020. https://doi.org/10.14778/3407790.3407805

[11] K. Kremer and A. Brinkmann, “FADaC: A Self-Adapting Data Classifier for Flash Memory,” in *Proceedings of the 12th ACM International Conference on Systems and Storage (SYSTOR ’19)*, Haifa, Israel, pp. 167-178, June 2019. https://doi.org/10.1145/3319647.3325829

[12] Y. Kim, B. Tauras, A. Gupta, and B. Urgaonkar, “FlashSim: A Simulator for NAND Flash-Based Solid-State Drives,” in *Proceedings of the 1st International Conference on Advances in System Simulation*, Porto, Portugal, pp. 125-131, September 2009. https://doi.org/10.1109/SIMUL.2009.17

[13] SNIA (Storage Networking Industry Association). Block I/O Traces [Internet]. Available: http://iotta.snia.org/traces/block-io.

백승수(Seungsu Baik)

2015년 : 세종대학교 (공학사-정보통신공학)

2016년~현 재: 한양대학교 컴퓨터소프트웨어학과 석박사통합과정
※관심분야 : 스토리지 시스템, 플래시메모리(Flash memory) 등

차재혁(Jaehyuk Cha)

1987년 : 서울대학교 (이학사-계산통계학)
1991년 : 서울대학교 대학원 (공학석사-컴퓨터공학)
1997년 : 서울대학교 대학원 (공학박사-컴퓨터공학)

1998년~현 재: 한양대학교 컴퓨터소프트웨어학부 교수
※관심분야 : 데이터베이스, e-러닝, 스토리지 시스템, 플래시스토리지