Asia-pacific Journal of Convergent Research Interchange
Vol.9, No.5, May 31 (2023), pp.13-23
http://dx.doi.org/10.47116/apjcri.2023.05.02

The Prediction of Forward Reuse Distance Using a Classification Model for Write Buffer Management

쓰기 버퍼 관리를 위해 분류 모델을 활용한 미래 재참조거리 예측

Hyejin Cha[1], Taeseok Kim[2]

차혜진[1], 김태석[2]

[1] *Student, Computer Engineering, Kwangwoon University, Korea, hj.cha@kw.ac.kr*
[2] *Professor, Computer Engineering, Kwangwoon University, Korea, tskim@kw.ac.kr*

Corresponding author: Taeseok Kim

Abstract: The SSD write buffer is used to improve the latency of I/O requests and the lifetime of flash memory. Most algorithms for managing the SSD write buffer use past reference information such as LRU (Least Recently Used) or predict the future reference using deep learning with high computation cost. In this paper, we propose a technique for managing write buffers by efficiently predicting future reference patterns with a reasonable computation overhead using machine learning. If the future reuse distance, which means when the same data is re-referenced, is greater than the write buffer size, a strategy of directly flushing it in the flash memory without storing it in the buffer is used. However, since it is impossible to accurately predict the future reuse distance and is not essential for buffer management, we first group the future reuse distance into several levels considering the size of the write buffer. Then, a priority-based buffer management policy is proposed by taking the grouped future reuse distance as a priority for caching. As a result of the simulation, the proposed method showed a performance close to the optimal Belady's algorithm in terms of hit rate, while showing significant performance improvement compared to the existing LRU-based algorithm.

Keywords: SSD, Write Buffer, Forward Reuse Distance, Classification Model

요약: SSD의 쓰기 버퍼는 입출력 요청의 지연시간과 플래시 메모리의 수명을 개선하기 위해 사용된다. SSD 쓰기 버퍼를 관리하는 기존의 알고리즘들은 LRU와 같이 과거의 참조 정보를 사용하거나 계산 비용이 높은 딥러닝을 활용해 미래의 참조 패턴을 예측한다. 본 논문에서는 머신러닝을 활용해 합리적인 계산 오버헤드로 미래의 참조패턴을 예측해 쓰기 버퍼를 관리하는 기법을 제안한다. 동일한 데이터가 언제 다시 참조되는지를 의미하는 미래 재참조거리가 쓰기 버퍼보다 클 경우 버퍼에 저장하지 않고 바로 플래시 메모리에 저장하는 전략을 사용한다. 다만, 미래 재참조거리를 정확하게 예측하는 것은 불가능할 뿐만 아니라 버퍼관리에 필수적이지 않기 때문에 쓰기 버퍼의 크기를 고려해 몇 단계로 그룹핑한다. 그런 다음, 그룹으로 정의된 미래 재참조거리를 캐싱을 위한 우선순위 삼아 우선순위 기반의 버퍼관리 정책을 제안한다. 모의실험 결과, 제안하는 기법이 히트율 관점에서 최적인 Belady 알고리즘에 근접하는 성능을 보인 한편, LRU 기반의 기존 알고리즘에 비해 큰 성능 향상을

Received: January 05, 2023; 1st Review Result: February 19, 2023; 2nd Review Result: March 15, 2023
Accepted: April 30, 2023

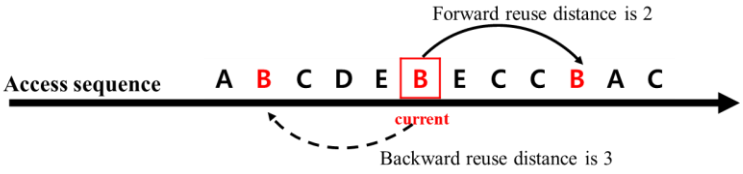
보였다.

핵심어: SSD, 쓰기 버퍼, 미래 재참조거리, 분류 모델

1. 서론

최근 저장장치로 널리 이용되는 SSD(Solid State Drives)는 하드디스크에 비해 짧은 입출력 대기 시간과 저전력 등의 장점을 가진다. 그러나 여전히 플래시 메모리의 물리적 특성으로 인해 짧은 수명과 상대적으로 낮은 쓰기 성능은 풀어야 할 숙제로 남아있다. SSD 내부에 있는 DRAM은 주소변환을 위한 매핑 테이블을 저장하기 위한 용도로도 사용되지만, 쓰기성능과 수명을 향상시키기 위한 쓰기 버퍼로도 활용된다[1]. 쓰기 버퍼는 다른 캐싱 기법과 유사하게 워크로드의 지역성을 활용해 적중률을 높이는 방법으로 성능을 개선할 수 있다. 지역성은 동일한 데이터가 재사용될 때까지 접근한 고유한 데이터 참조 횟수로 정의되는 재참조거리(reuse distance)로 설명될 수 있다[2].

[그림 1]은 과거 재 참조거리(backward reuse distance)와 미래 재 참조거리(forward reuse distance)를 설명하는데, 현재 데이터가 B일 경우 각각 그 거리는 3과 2가 된다. 적중률을 높이기 위해서는 미래 재참조거리가 가장 긴 데이터를 교체하는 Belady's 알고리즘을 사용하는 것이 이상적이나 미래의 참조 패턴을 미리 알 수 없기 때문에 기존 연구들은 과거 재 참조거리가 가장 긴 데이터를 교체하는 LRU(Least Recently Used)에 근거한 알고리즘들을 주로 사용하였다[3].



[그림 1] 재참조거리 예시

[Fig. 1] Example of Reuse Distances

본 논문에서는 SSD의 쓰기 버퍼 관리를 위해 머신러닝을 사용하여 미래 재참조거리를 예측하는 기법을 제안한다. 사실 미래 재참조거리를 정확하게 예측하는 것은 불가능할 뿐만 아니라 반드시 필요하지 않다. 그보다 미래 재참조거리가 버퍼보다 클 경우 해당 데이터는 다시 참조되기 전에 플래시메모리로 플러시 되므로 버퍼에 저장할 필요가 없다는 점에 착안한다. 따라서 재참조거리가 버퍼의 크기보다 큰지 여부를 파악하는 것이 중요한데, 다만 이렇게 간단하게 문제를 정의할 경우 충분한 데이터셋이 줘지지 않아 머신러닝 예측 성능이 좋지 않다. 따라서 본 논문에서는 버퍼의 크기에 기반하여 미래 재참조거리를 몇 단계의 그룹으로 분류하고 머신러닝 기법 중 분류(classification) 모델을 이용해 입출력 요청의 미래 재참조거리를 그룹 수준으로 예측한다. 그런 다음, 재참조거리가 짧은 그룹으로 예측된 데이터부터 버퍼에 저장하는 우선순위 기반의 관리 기법을 설계한다. 모의실험을 통해 여러 워크로드로 실험한 결과, 제안하는 기법이 Belady 알고리즘에 근접한 성능을 보이는 한편, 과거 재참조거리를 이용하는 LRU 계열의 알고리즘보다 월등한 성능을 보임을 알 수 있었다.

본 논문의 구성은 다음과 같다. 2장에서는 재참조거리를 활용한 버퍼 교체 알고리즘 과

최근 머신러닝을 사용한 SSD 입출력 최적화 연구에 대해 분석한다. 3장에서는 미래 재참조거리 예측 방법과 우선순위 기반의 쓰기 버퍼 관리 방법에 관해 설명한다. 4장에서는 제안한 알고리즘의 성능 실험 결과를 제시하고 마지막으로 5장에서는 결론 및 향후 과제에 대해 기술한다.

2. 관련 연구

적중률 관점에서 최적의 기법은 미래 재참조거리가 가장 긴 데이터를 교체하는 Belady's MIN 알고리즘이다[4]. 하지만, 미래의 데이터 참조패턴을 미리 알 수는 없기 때문에 지역성에 근거해 과거 재참조거리가 가장 긴 데이터부터 교체하는 LRU 알고리즘에 기반한 버퍼 관리 알고리즘이 그간 주를 이루었다. 대표적으로 CFLRU(Clean-First LRU)[5]가 있으며, 그 외에 LRU-WSR(LRU and Write Sequence Reporting)[6], AD-WSR(Adaptive Double LRU)[7], DPW-LRU(Dynamic Page Weight LRU)[8] 등이 있다.

최근 사회 각 분야에서 머신러닝을 통해 미래의 데이터를 예측하는 연구를 시도하고 있다. 머신러닝을 이용해 스토리지의 입출력 성능을 향상시키는 연구도 점차 활발해지고 있는데, 특히 버퍼 관리나 캐싱은 향후 워크로드의 패턴 예측 정확도에 따라 그 성능이 크게 달라지기 때문에 머신러닝을 활용해 효율성을 높이기 위한 다양한 시도가 이루어지고 있다.

MLCache[9]는 SSD의 입출력 성능 개선을 위해 딥러닝 모델을 사용하여 쓰기 버퍼 관리 방법을 제안하였다. 재참조거리를 학습에 사용하여 재참조거리의 분포를 지속해서 모니터링해 동적으로 쓰기 버퍼의 크기를 변경하였다. pOPT(Prediction-based OPT)[10]는 미래 재참조거리를 예측하기 위해 다중 LSTM(Multi-Long Short Term Memory)모델을 사용했다. 또한 [11]에서는 LSTM 기반 시퀀스투시퀀스(Seq2seq) 모델을 이용하여 입력 블록 참조 순서를 예측하였다. LRB(Learning Relaxed Belady)[12]는 부스팅 모델을 사용하여 낮은 오버헤드를 가질 수 있도록 Belady's 알고리즘을 근사화하는 시도를 하였다.

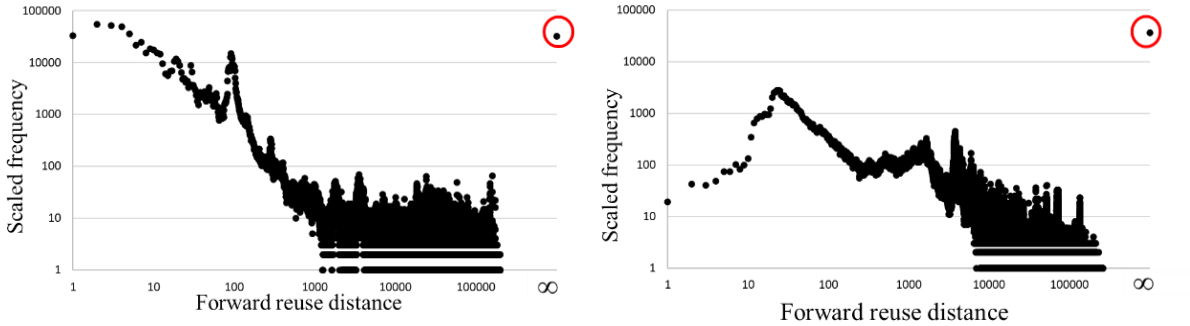
SSD의 쓰기 버퍼와 같이 제한된 환경에서 미래의 참조 패턴을 예측하기 위해 계산량이 많은 딥러닝 모델을 사용하는 건 아직 무리가 있다[9]. 본 연구에서는 머신러닝을 이용해 미래 재참조거리를 예측하되, 무엇보다 계산 오버헤드가 낮은 모델을 설계해 현실적으로 구현 가능한 기법을 제안한다.

3. 분류 모델 기반 쓰기 버퍼 관리 기법

3.1 미래 재참조거리 예측

본 연구는 미래 재참조거리를 예측하는 것으로 무엇보다 실제 워크로드에 재참조거리가 어떻게 분포되는지 확인하는 것이 선행되어야 한다. [그림 2]는 마이크로소프트에서 운영한 서버의 블록 계층에서 수집한 MSR 워크로드 중 높은 지역성을 가지는 워크로드와 낮은 지역성을 가지는 워크로드의 미래 재참조거리 분포를 나타낸다[13][14]. 가로축은 미래 재참조거리의 값을 의미하고 세로축은 그 빈도로 모두 로그 스케일로 표현하였다. 가장 오른쪽의 무한(∞)은 더 이상 참조 되지 않는 데이터를 의미하며 실제로 많은 데이터가 재참조되지 않는 것을 볼 수 있다. 왼쪽 부분의 빈도가 높을수록 미래 재참조거리 거리가 짧은 데이터가 많아 지역성이 높다고 판단할 수 있는데, mds_0은 미래 재참조거리가 짧은 데이터가 많고 더 이상 재참조되지 않는

데이터가 적어 지역성이 높으며, 반대로 stg_1은 미래 재참조거리가 짧은 데이터가 적고 더 이상 재참조되지 않는 데이터가 많아 지역성이 낮다고 볼 수 있다.



(a) mds_0 (b) stg_1

[그림 2] 미래 재참조거리 빈도 분포

[Fig. 2] Frequency Distribution of Forward Reuse Distance

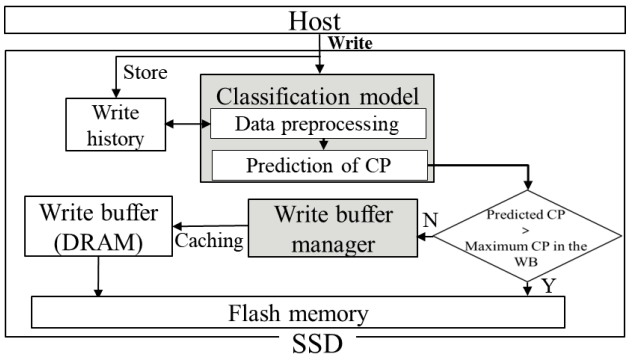
언급하였듯이 쓰기 버퍼 크기보다 미래 재참조거리가 큰 경우 쓰기 버퍼에 데이터를 저장하지 않고 바로 플래시 메모리로 저장하는 것이 효율적이다. 머신러닝을 사용하여 미래 재참조거리가 쓰기 버퍼 크기를 넘는지 판단하는 시도를 해볼 수 있으나, 단순히 이진(Binary)으로 분류되는 결과는 학습에 충분한 정보를 제공하지 못하였다. 따라서 여기서는 미래 재참조거리를 쓰기 버퍼의 크기를 고려해 n개의 그룹으로 나눈 후 분류 기법을 이용해 미래 재참조거리를 예측하는 전략을 이용한다. 즉, 쓰기 버퍼를 1이라 가정하고 그룹을 8단계로 구분하면, 1/8, 1/4, 1/2, 1, 2, 4, 8을 기준으로 나누어 미래 재참조거리를 그룹핑한다. 결론적으로 짧은 미래 재참조거리를 가진 데이터부터 버퍼에 저장하면 되므로 각 그룹은 캐싱을 위한 우선순위(caching priority, CP)라 할 수 있다.

여기서 가장 중요한 것은 계산량이 많은 딥러닝을 통해 각 입출력 요청들의 미래 재참조거리를 예측하는 것이 아니라, 비교적 계산 오버헤드가 적은 적절한 머신러닝 모델을 선택하는 것이다. 데이터 집합의 특징을 바탕으로 각 입출력 요청을 n개의 그룹으로 분류하는 데는 분류 모델이 적합하고 여기에는 몇 가지 후보가 있다. SVM(Support Vector Machine) 모델[15]은 학습 데이터가 큰 경우 딥러닝보다 학습 시간도 길고 예측 성능도 떨어지며 이진 분류에 적합하므로 여기서 사용하기에 적절하지 않다. 또한 나이브 베이즈(Naïve Bayes model) 모델 또한 이진 분류 기법이므로 적합하지 않다. 따라서 본 연구에서는 다중 분류를 지원하면서도 높은 정확도를 보여주는 앙상블 모델(Ensemble model)이 시도해볼 만하며, 그 중 랜덤 포레스트(Random forest), XGBoost(Extreme Gradient Boosting), AdaBoost(Adaptive Boosting) 등이 가능하다[16].

3.2 캐싱 우선순위를 이용한 쓰기 버퍼 관리

[그림 3]은 본 논문에서 제안하는 SSD 구조이다. 쓰기 요청이 들어오면 캐싱 우선순위를 예측하는 분류 모델(classification model)과 캐싱 우선순위에 따라 버퍼를 관리하는 쓰기 버퍼 관리자(Write buffer manager)가 구조의 핵심이다. 분류 모델에는 데이터를 수집 및 가공하는 데이터 전처리(Data preprocessing) 모듈과 캐싱 우선순위를

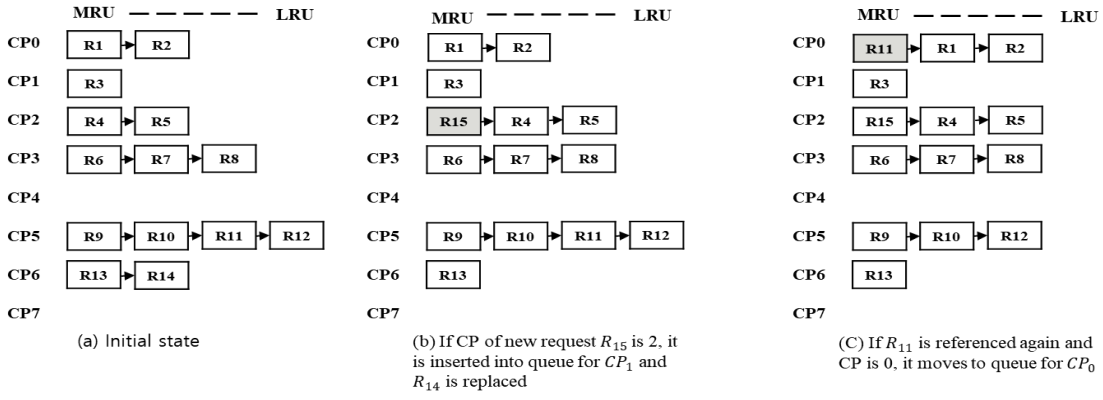
예측하는 모듈(Prediction of CP)로 다시 나뉜다. 전체 동작은 다음과 같은데, 먼저 호스트에서 쓰기 요청이 들어오면 분류 모델로 이동된다.



[그림 3] 분류 모델을 이용한 쓰기 버퍼 관리

[Fig. 3] Write Buffer Management using a Classification Model

분류 모델에 사용되는 정보는 크게 두 가지로 나뉘는데, 하나는 입출력 요청을 통해 바로 알 수 있는 입출력 요청 시간, LBA(Logical Block Address), 입출력 크기, 입출력 종류이다. 나머지는 이러한 정보들을 수집에서 가공하거나 통계적인 정보들인데, 워크로드의 특성은 지속적으로 변하므로 윈도우를 정의하고 그만큼 기록하기 위한 모듈(write history)을 사용한다. 이를 통해 예측하고자 하는 요청의 과거 재참조 거리, 윈도우내 동일한 요청의 빈도, 재참조 거리까지의 요청 개수, 인터벌을 구한다. 여기에 인터벌과 입출력 크기의 평균과 표준편차 정보를 각각 수집해 분류 모델의 데이터셋으로 입력한다. 수집한 데이터 정보를 통해 예측한 캐싱 우선순위는 쓰기 버퍼에 공간이 없고, 쓰기 버퍼에 저장된 캐싱 우선순위들과 비교해 가장 낮을 경우 바로 플래시 메모리로 저장한다.



[그림 4] 캐싱 우선순위가 8 단계일 때의 쓰기 버퍼 관리 시나리오

[Fig. 4] Write Buffer Management Scenario when Caching Priorities is Divided into 8 Levels

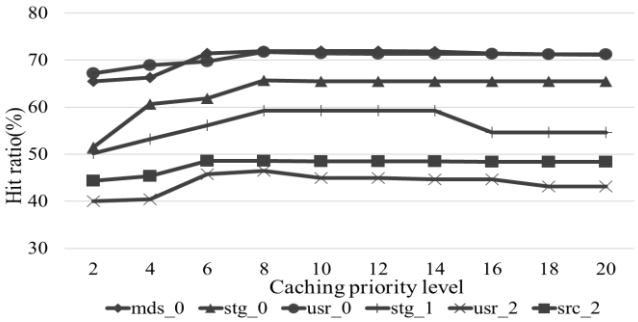
쓰기 버퍼 관리자는 캐싱 우선순위의 개수만큼 구성된 다중 큐를 관리하며, 각 큐는 LRU 방식으로 데이터가 교체된다. [그림 4]는 캐싱 우선순위를 8단계로 정의했을 때 쓰기 버퍼 관리 시나리오를 보여준다. 그림 4(b)처럼 쓰기 버퍼의 빈 공간이 없는 상태에서

새로운 요청이 들어오면 해당 큐에 삽입이 되고, 가장 낮은 우선순위 큐의 LRU위치에 있는 데이터가 교체된다. 만일 새로운 요청이 큐에 있는 요청들의 우선순위보다 가장 낮으면 큐에 아무런 변화없이 플래시 메모리에 바로 기록되고, 큐에 있는 데이터가 재참조될 경우 큐 사이에 이동이 발생한다(그림 4(c).

4. 성능 평가

4.1 실험 환경 및 방법

제안한 알고리즘은 캐싱 우선순위를 몇 단계로 구분하느냐에 따라 예측 정확도와 쓰기 버퍼의 성능이 결정된다. [그림 5]는 MSR 워크로드 중 6개의 워크로드를 사용해 캐싱 우선순위의 단계에 따른 적중률을 나타낸다. 쓰기 버퍼의 크기는 32MB 고정하여 캐싱 우선순위를 2부터 20까지 변경하여 실험 한 결과, 대부분의 경우 캐싱 우선순위를 8단계부터 14단계까지 나눌 때 가장 좋은 성능을 보였다. 따라서 머신러닝 학습 오버헤드를 고려하여 이후 실험에서는 캐싱 우선순위를 8로 고정하였다.



[그림 5] 캐싱 우선순위 단계에 따른 적중률

[Fig. 5] Hit rate by Caching Priority Levels

캐싱 우선순위 예측 시 사용해야 할 머신러닝 모델 선정이 무엇보다 중요한데, 앞에서 언급한 바와 같이 다중 분류가 필요해 앙상블 모델을 사용하였다. 쓰기 버퍼 크기를 32MB, 캐싱 우선순위를 8로 고정하여 MSR 워크로드 중 mds_0로 실험한 결과, 정확도, 정밀도, 재현율 모두에서 랜덤 포레스트가 가장 우수한 성능을 보이는 것을 확인하였다[표 1]. 오버헤드 측면에서도 랜덤 포레스트가 좋은 후보인데, 과거 가비지 컬렉션 실행 여부를 랜덤 포레스트 모델을 사용한 결과 평균 예측 시간이 33μs였고, 또 다른 성능 최적화 연구에서도 랜덤 포레스트 모델 사용 시 예측 시간이 20.23μs이었다[17][18]. 이러한 근거를 통해 본 연구에서는 캐싱 우선순위를 예측하기 위한 분류 모델로 랜덤 포레스트를 사용하였다.

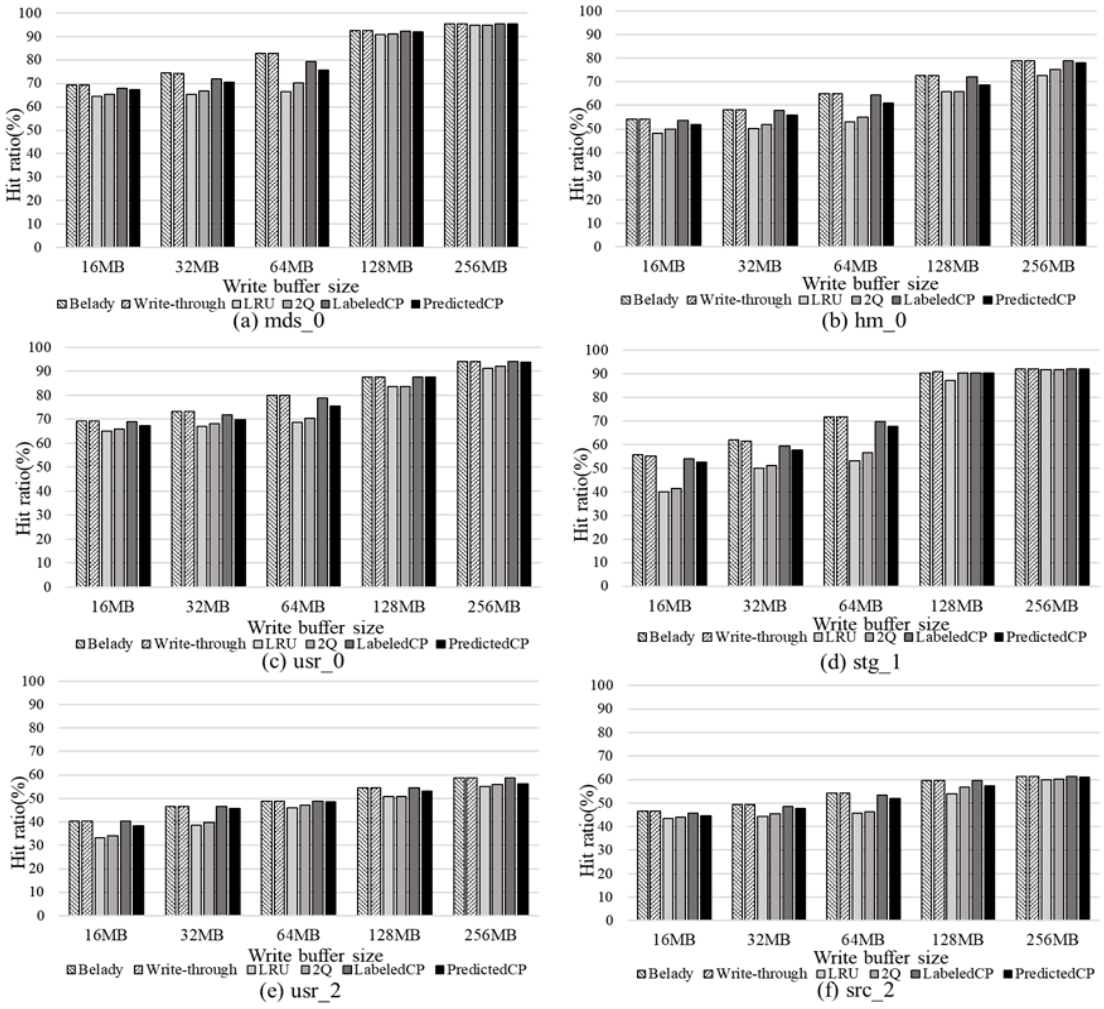
[표 1] 분류 모델별 성능

[Table 1] Performance by Classification Model

Model	Accuracy (%)	Precision (%)	Recall(%)
Random forest	96	96	96
XGBoost	93	94	93
AdaBoost	86	86	86

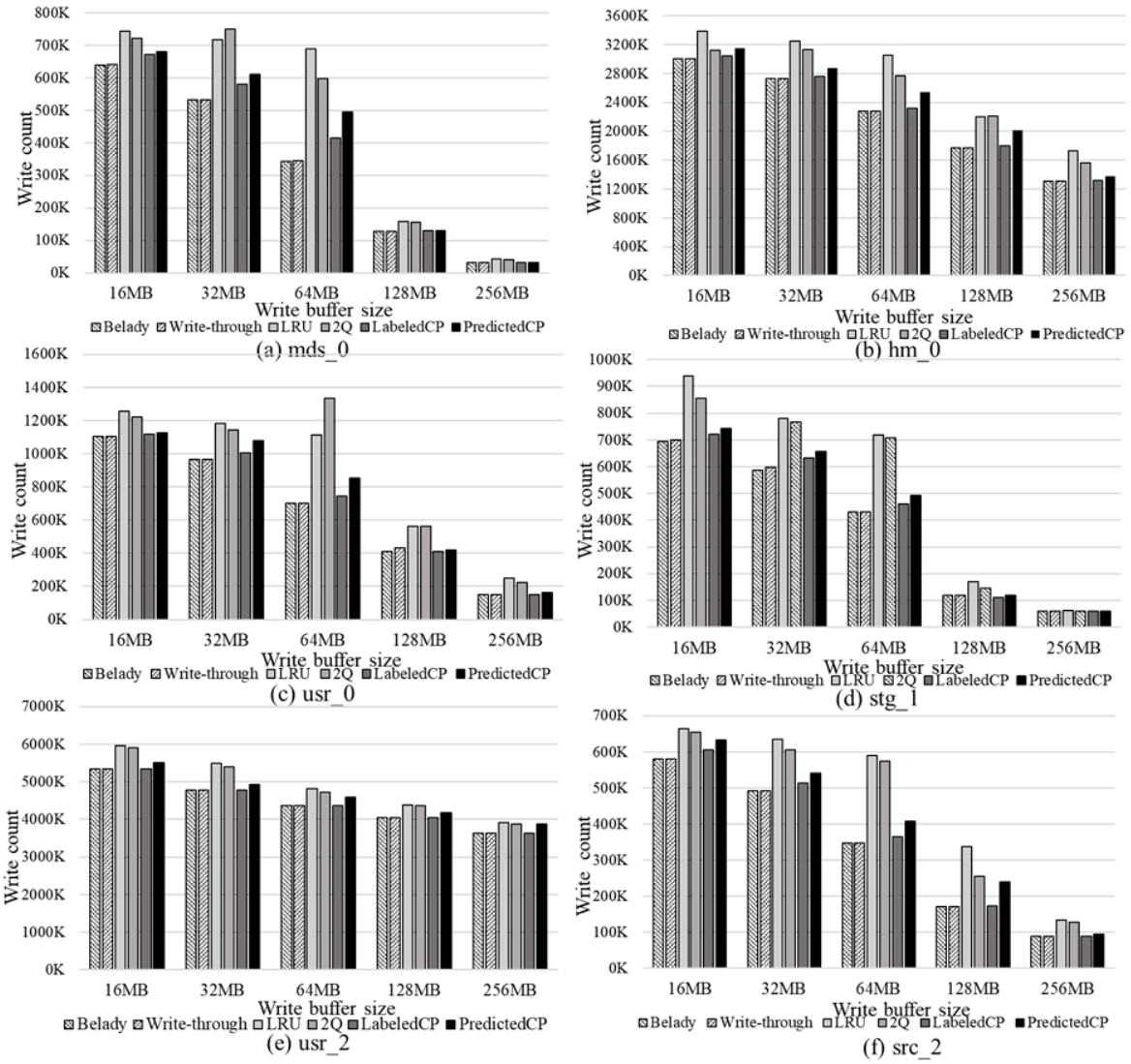
본 논문에서 제안하는 알고리즘을 검증하기 위해 파이썬으로 자체 제작한 SSD 시뮬레이터를 사용하였다. 해당 시뮬레이터는 데이터가 들어오면 판다스(Pandas)를 통해 데이터를 전처리하고, 랜덤 포레스트를 통해 캐싱 우선순위를 예측한 후 제안한 방법으로 쓰기 버퍼 관리하도록 설계하였다. 쓰기 버퍼에서 다루는 기본 단위는 페이지 크기인 4KB로 설정하였고 쓰기 버퍼 크기는 16MB부터 256MB까지 변경하면서 실험을 진행하였다. 또한 MSR 워크로드 중 6개의 워크로드 외에 비교적 최신 워크로드인 YCSB RocksDB 워크로드를 추가로 사용하였다[14][19]. 제안한 알고리즘(PredictedCP)의 성능을 평가하기 위해 과거 재참조거리에 기반한 LRU 알고리즘과 2Q 알고리즘[20]을 구현하였다. 또한 Belady 알고리즘과 미래 재참조거리가 버퍼 크기보다 클 경우 바로 플래시 메모리에 기록하는 Write-through 알고리즘을 추가로 구현해 제안하는 기법이 이들 최적 알고리즘과 얼마나 차이가 나는지 살펴보았다. 마지막으로 미래 재참조거리를 100% 예측하는 것을 가정하는 알고리즘(labeledCP)을 구현해 예측 정확도에 따른 성능 평가를 시도하였다.

4.2 실험 결과



[그림 6] 쓰기 버퍼 크기에 따른 적중률

[Fig. 6] Hit Ratio by Write Buffer Size



[그림 7] 쓰기 버퍼 크기에 따른 쓰기 연산 횟수

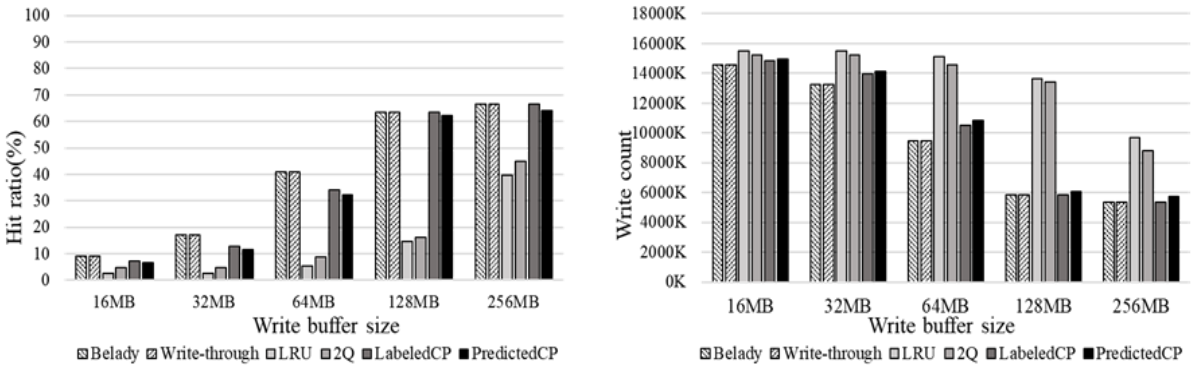
[Fig. 7] Write Count by Write Buffer Size

[그림 6]은 MSR 워크로드의 알고리즘별 적중률 결과이다. 가로축은 쓰기 버퍼 크기이며 세로축은 각 알고리즘의 적중률을 보여준다. Belady와 Write-through는 거의 동일한 성능을 보이며 제안한 알고리즘인 PredictedCP가 LRU, 2Q와 비교 시 모든 워크로드에서 압도적으로 향상된 성능을 보였다. 버퍼 크기가 64MB일 때, 제안하는 기법이 LRI, 2Q와 비교해 mds_0에서는 각각 9.2%, 5.5%, hm_0에서는 7.9%, 5.9%, usr_0에서는 6.8%, 5.2%, stg_1에서는 14.4%, 11.2%, usr_2에서는 6.9%. 5.9%, src_2에서는 6.3%, 5.8%의 성능 향상이 있었다. PredictedCP가 Belady와 비교 시 최대 9%의 차이를 보여주며 LabeledCP과는 최대 5% 차이를 보여준다. LabeledCP는 대부분의 경우에 Belady와 거의 유사한 성능을 보이는데, 이는 머신러닝을 통해 예측을 잘하는 것이 상당히 중요하다는 것을 알려준다. 또한 쓰기 버퍼 크기가 128MB보다 클 경우 적중률 측면에서 거의 비슷한 성능을 보이는데, 이는 워크로드의 데이터셋 크기 때문이다.

[그림 7]은 MSR 워크로드의 알고리즘별 플래시 메모리에 대한 쓰기 연산 횟수 그래프이다. 가로축은 쓰기 버퍼 크기이며 세로축은 각 알고리즘의 쓰기 연산 횟수를

보여준다. 당연하게도 적중률이 높을수록 플래시 메모리에 대한 쓰기 횟수는 낮아져 결국 수명이 향상된다고 볼 수 있다. 적중률 실험 결과와 유사하게 대부분의 결과 LabeledCP가 Belady와 유사한 성능을 보이고, PredictedCP 또한 큰 차이를 보이지 않는다. 한편 과거 재참조거리에 기반한 LRU나 2Q는 현저하게 성능이 떨어짐을 알 수 있다.

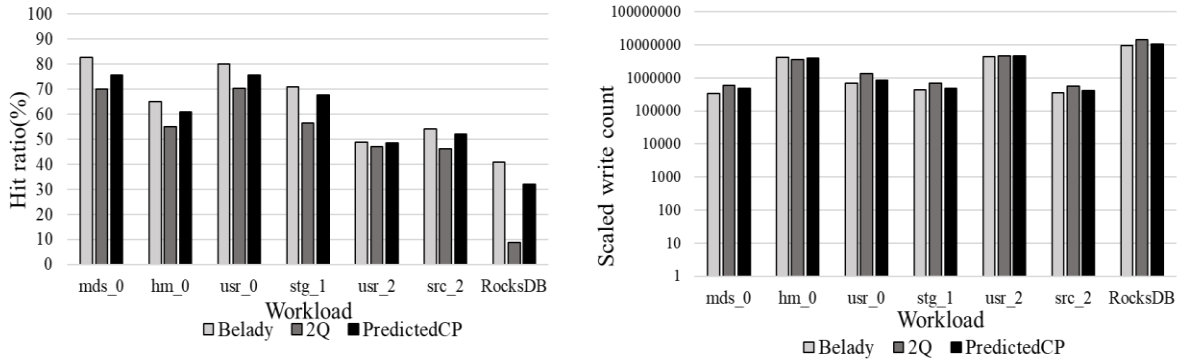
[그림 8]은 RocksDB 워크로드를 사용했을 때 적중률 및 쓰기 연산 횟수 그래프이다. RocksDB 워크로드는 MSR 워크로드보다 지역성이 낮아 쓰기 버퍼 크기에 따른 적중률이 크게 달라진다. 많은 경우에 있어서 PredictedCP는 Belady에 꽤 근접한 성능을 보이지만 LRU와 2Q는 매우 좋지 않은 성능을 보인다. 이러한 점은 과거 재참조거리를 이용하는 기존 알고리즘이 특정 조건에서는 성능이 크게 떨어져 머신러닝을 통한 미래의 참조패턴 연구가 중요함을 보여준다. [그림 9]는 앞서 성능 평가 결과를 종합한 것으로 버퍼 크기가 64MB일 때 Belady 알고리즘, 기존 알고리즘인 2Q, 그리고 제안한 PredictedCP의 성능 평가 결과이다.



(a) hit ratio (b) write count

[그림 8] RocksDB 워크로드에서의 성능 평가

[Fig. 8] Performance Evaluation with RocksDB Workload



(a) hit ratio (b) write count

[그림 9] 워크로드별 성능 결과

[Fig. 9] Performance Results by Workload

5. 결론

본 논문에서는 머신러닝을 사용하여 미래 재참조거리를 예측하는 쓰기 버퍼 관리 기법을 제안하였다. 반드시 미래 재참조거리를 정확하게 예측할 필요가 없다는 점에 착안해 계산량이 많은 딥러닝 대신 비교적 오버헤드가 적은 분류 기법을 이용해 미래 재참조거리를 예측하였다. 이를 바탕으로 버퍼 교체 알고리즘을 설계한 결과, 제안한 알고리즘이 Belady 알고리즘과 적중률 측면에서 큰 차이를 보이지 않았으며, 과거 재참조거리에 근거한 LRU나 2Q보다 월등히 높은 성능을 보였다. 무엇보다 미래 참조패턴을 예측하는데 있어서 작은 오버헤드로 높은 성능을 보이는 머신러닝 활용사례를 제안하였다는데 의의가 있으며, 향후 보다 정교한 머신러닝 모델 개발 및 버퍼 관리 알고리즘 설계, 그리고 다양한 워크로드에서의 추가 실험이 필요할 것으로 보인다.

6. 감사의 글

이 논문은 2021년도 광운대학교 우수연구자 지원 사업에 의해 연구되었음. 또한 정부(과학기술정보통신부)의 재원으로 한국연구재단의 지원을 받아 수행된 연구임(No. 2020R1F1A1074676)

References

[1] I. Shin, J. Kim, Performance analysis of buffer management policy considering internal parallelism of solid-state drives, IEICE Electronics Express, (2015), Vol.15, No.15, pp.1-8.
DOI: http://dx.doi.org/10.1587/elex.15.20180419

[2] R. L. Mattson, J. Gecsei, D. R. Slutz, I. L. Traiger, Evaluation techniques for storage hierarchies, IBM System Journal, (1970), Vol.9, No.2, pp.78-117.
DOI: http://dx.doi.org/10.1147/sj.92.0078

[3] S. Jiang, X. Zhang, LISR : An efficient low inter-reference recency set replacement policy to improve buffer cache performance, ACM SIGMETRICS Performance Evaluation Review, (2002), Vol.30, No.1, pp.32-42.
DOI: http://dx.doi.org/10.1145/511399.511340

[4] L. A. Belady, A study of replacement algorithms for a virtual-storage computer, IBM System Journal, (1966), Vol.5, No.2, pp.78-101.
DOI: http://dx.doi.org/10.1147/sj.52.0078

[5] S. Park, D. Jung, J. Kang, J. Kim, J. Lee, CFLRU : A replacement algorithm for flash memory, Proceedings of the 2006 international conference on Compilers, architecture and synthesis for embedded systems, ACM DL, pp.234-241, (2006)
DOI: http://dx.doi.org/10.1145/1176760.1176789

[6] H. Jung, H. Shim, S. Park, S. Kang, J. Cha, LRU-WSR : Integration of LRU and writes sequence reordering for flash memory, IEEE Transactions on Consumer Electronics, (2008), Vol.54, No.3, pp.1215-1223.
DOI: http://dx.doi.org/10.1109/TCE.2008.4637609

[7] P. Jin, Y. Ou, T. Härder, Z. Li, AD-LRU : An efficient buffer replacement algorithm for flash-based databases, Data & Knowledge Engineering, (2012), Vol.73, pp.83-102.
DOI: http://dx.doi.org/10.1016/j.datak.2011.09.007

[8] Y. Yuan, J. Zhang, G. Han, G. Jia, L. Yan, W. Li, DPW-LRU : An efficient buffer management policy based on dynamic page weight for flash memory in cyber-physical systems, IEEE Access, (2019), Vol.7, pp.58810-58821.

DOI: http://dx.doi.org/10.1109/ACCESS.2019.2914231

[9] W. Liu, J. Cui, J. Liu, L. T. Yang, MLCache: A space-efficient cache scheme based on reuse distance and machine learning for NVMe SSDs, Proceedings of the 39th International Conference on Computer-Aided Design, ACM DL, pp.1-9, (2020)
DOI: http://dx.doi.org/10.1145/3400302.3415652

[10] P. Li, Y. Gu, Learning forward reuse distance, arXivpreprint zrXic:2007.15859, (2020)
DOI: http://dx.doi.org/10.48550/arXiv.2007.15859

[11] H. Choi, S. Park, Learning future reference patterns for efficient cache replacement decisions, IEEE Access, (2022), Vol.10, pp.25922-25934.
DOI: http://dx.doi.org/10.1109/ACCESS.2022.3156692

[12] Z. Song, D. S. Berger, K. Li, W. Lloyd, Learning relaxed Belady for content distribution network caching, Proceedings of the 17th Usenix Conference on Networked Systems Design and Implementation, ACM DL, (2020)
Available from: https://www.usenix.org/conference/nsdi20/presentation/song

[13] D. Narayanan, A. Donnelly, A. Rowstron, Write off-loading: Practical power management for enterprise storage, ACM Transactions on Storage, (2008), Vol.4, No.3, pp.1-23.
DOI: http://dx.doi.org/10.1145/1416944.1416949

[14] http://iotta.snia.org/traces/block-io, Dec 29 (2022)

[15] C. Cortes, V. Vapnik, Support-vector networks, Machine Learning, (1995), Vol.20, pp.273-297.
DOI: http://dx.doi.org/10.1007/bf00994018

[16] K. P. Murphy, Machine learning: a probabilistic perspective, MIT Press, (2012)

[17] C. Wu, I. Li, J. Chen, A supervised-learning-based garbage collection in solid-state drives (SSDs), IT Professional, (2021), Vol.23, No.6, pp.33-45.
DOI: https://doi.org/10.1109/mitp.2021.3106173

[18] Y. Zhang, K. Zhou, P. Huang, H. Wang, J. Hu, Y. Wang, Y. Ji, B. Cheng, A machine learning based write policy for SSD cache in cloud block storage, 2020 Design, Automation & Test in Europe Conference & Exhibition(DATE), IEEE, (2020)
DOI: https://doi.org/10.23919/date48585.2020.9116539

[19] G. Yadgar, M. Gabel, S. Jaffer, B. Schroeder, SSD-based workload characteristics and their performance implications, ACM Transactions on Storage, (2008), Vol.17, No.1, pp.1-26.
DOI: http://dx.doi.org/10.1145/3423137

[20] T. Johnson, D. Shasha, 2Q: A low overhead high performance buffer management replacement algorithm, Proceedings of the 20th International Conference on Very Large Data Bases, ACM DL, pp.439-450, (1994)
DOI: https://dl.acm.org/doi/10.5555/645920.672996