- 2025 전기 졸업과제 착수보고서 -

딥러닝을 이용한 SSD 성능 개선 연구

지도교수 : 안성용

팀명 : SSDeep Learning

201924544 이준형

201914502 강인석

202155537 김지수

1. 과제 개요 3

1.1 연구 배경 및 필요성 3

1.2 과제 주제 및 목적 4

1.3 기대 효과 5

2. 기술 배경 및 선행 연구 6

2.1 SSD 및 FTL 기본 구조와 한계 6

2.2 입출력 워크로드 특성 7

2.3 선행연구 요약 8

3. 연구 내용 및 방법론 9

3.1 데이터 수집 및 전처리 계획 9

3.2 딥러닝 모델 설계 및 적용 방안 9

3.3 FTL 성능 개선 전략 10

3.4 성능 평가 및 검증 계획 11

4. 연구 일정 및 추진 계획 12

4.1 연구 단계별 일정 12

4.2 팀원별 역할 분담 13

5. 연구 리스크 및 대응 방안 14

5.1 예상되는 문제점 및 한계 14

5.2 대응 전략 및 대안 15

6. 참고문헌 16

1. 과제 개요

1.1 연구 배경 및 필요성

현대 컴퓨팅 환경에서 저장장치의 성능은 전체 시스템의 처리 속도와 사용자 경험에 직접적인 영향을 미친다. 특히 SSD(Solid State Drive)는 기계적 구조가 없는 반도체 기반 저장장치로서, 빠른 접근 속도와 낮은 지연 시간 덕분에 하드디스크를 대체하며 빠르게 보급되고 있다. 하지만 SSD는 NAND 플래시 메모리의 구조적 특성상 덮어쓰기가 불가능하고 삭제가 블록 단위로만 가능하다는 제약이 존재하며, 이로 인해 쓰기 지연, 쓰기 증폭(write amplification), 수명 저하 등의 문제가 발생할 수 있다.

이러한 한계를 해결하기 위해 SSD 내에는 FTL(Flash Translation Layer)이라는 펌웨어 기반의 주소 변환 계층이 존재하며, 논리 주소를 물리 주소로 매핑하고, Garbage Collection, Wear Leveling 등 저장 장치 최적화를 수행한다. 그러나 FTL은 사전에 정의된 정책에 따라 동작하며, 실제 사용자 환경에서 발생하는 다양한 입출력 패턴을 충분히 반영하지 못해 성능 저하로 이어지는 경우가 많다.

최근에는 입출력 워크로드의 시계열적·공간적 패턴을 보다 정교하게 분석하고 예측하기 위한 방법으로 딥러닝 기반 분석 기법이 주목받고 있다. 워크로드 데이터에는 명확한 규칙성과 반복성이 존재하는 경우가 많기 때문에, 이를 학습한 모델은 향후 발생할 입출력 요청을 예측하거나, FTL의 운영 방식을 사전에 조정하는 데 활용될 수 있다.

따라서 본 연구는 딥러닝을 이용하여 SSD 입출력 워크로드 특성을 분석하고, 이를 기반으로 FTL의 동작을 최적화함으로써 SSD의 전체적인 성능과 수명을 향상시키는 방안을 제시하고자 한다. 이는 기존의 정적이고 고정적인 FTL 정책을, 학습 기반으로 점진적 개선이 가능한 지능형 저장장치 제어 방식으로 발전시킬 수 있는 기초가 될 것이다.

1.2 과제 주제 및 목적

본 과제의 주제는 「딥러닝을 이용한 SSD 성능 개선 연구」이다.

SSD 내부의 Flash Translation Layer(FTL)는 논리 주소와 물리 주소 간의 매핑을 수행하며, Garbage Collection, Wear Leveling, Bad Block Management 등의 중요한 기능을 담당한다. 하지만 대부분의 FTL 알고리즘은 미리 정의된 정책에 따라 정적인 방식으로 동작하고, 실제 워크로드의 변화에 능동적으로 대응하지 못한다는 한계가 존재한다.

이에 따라 본 과제는 SSD의 입출력 워크로드를 딥러닝 기반으로 학습 및 분석함으로써, 반복적이거나 특징적인 I/O 패턴을 식별하고 예측하는 모델을 구축하는 것을 핵심 목적으로 한다. 예측된 워크로드 정보를 FTL의 주소 매핑 정책, 쓰기 위치 선택, 캐싱 전략 등과 연계하여 적용함으로써, SSD의 지연 시간 감소, 쓰기 증폭 완화, 수명 향상 등의 성능 개선을 유도하고자 한다.

구체적인 과제 수행 목적은 다음과 같다:

1. SSD 입출력 워크로드의 시계열적 패턴 분석
   SNIA IOTTA Repository[1]에서 수집된 I/O trace 데이터를 기반으로, 연속된 읽기/쓰기 요청 간의 상관성과 반복성을 분석함
2. 딥러닝 모델을 활용한 패턴 예측 및 분류
   LSTM, Transformer 등 시퀀스 예측에 특화된 딥러닝 모델을 활용하여 미래의 워크로드 요청을 예측하거나 입력되는 데이터의 hotness (hot/cold) 사전 인식
3. FTL 동작의 지능화 및 최적화 전략 설계
   예측된 정보에 기반하여 블록 할당, GC 우선순위, 쓰기 회피 전략 등을 조정하는 동적 FTL 제어 방식 제안
4. 시뮬레이터 기반의 성능 비교 실험 수행
   기존 정적 FTL과 본 과제에서 제안한 동적 제어 방식 간의 성능 차이를 SimpleSSD-SA[2] SSD 시뮬레이터 환경에서 검증

이를 통해 본 과제는 딥러닝 기술과 저장장치 아키텍처를 융합한 응용 사례를 제시하며,

향후 지능형 SSD 또는 자율적 스토리지 운영 기술로의 확장을 위한 기반 기술을 확보하는 것을 목표로 한다.

1.3 기대 효과

본 연구는 딥러닝 기반 워크로드 예측 기법을 저장장치 아키텍처에 접목함으로써, 기존 SSD의 정적 제어 방식이 갖는 한계를 보완하고, 보다 지능적이고 상황 적응적인 FTL 제어 체계로 발전시킬 수 있는 가능성을 제시한다.

아래와 같이 기술적, 실용적 측면에서의 기대 효과를 도출할 수 있다.

1) 저장장치 성능 향상

- 입출력 워크로드를 사전 분석 및 예측함으로써, FTL이 보다 선제적으로 최적화 동작(Garbage Collection, Block Allocation 등)을 수행 가능
- 입출력 요청이 집중되는 영역을 사전에 감지하여 병목 현상을 최소화하고, 쓰기 지연(Latency)을 줄임
- 전체적인 처리량(IOPS) 및 응답 속도 개선

2) SSD 수명 연장

- 반복 접근이 많은 Hot 영역을 효율적으로 분산시키고, Cold 영역을 안정적으로 보존함으로써 Wear Leveling 전략의 효과 극대화
- 불필요한 내부 복사 및 쓰기 연산을 줄여 쓰기 증폭(Write Amplification) 완화
- 결과적으로 셀 소모량 감소 → SSD 수명 증가

3) 딥러닝 기반 FTL 제어 프레임워크의 기반 마련

- 전통적인 FTL 정책이 반영하지 못했던 동적 환경 적응성을 모델 기반 제어를 통해 실현
- 향후 실제 SSD 펌웨어에 적용 가능한 정책 강화형 FTL 설계 기반 제공

- 관련 연구 분야에서 AI + Storage 융합 기술의 파일럿 사례로 활용 가능

4) 학술적 및 실용적 확장성 확보

- 입출력 로그 데이터 기반 학습-예측 방법론은, SSD 외에도 RAID, 하이브리드 스토리지, 분산 파일 시스템 등 다양한 저장 매체에 확장 가능
- 시뮬레이터 기반 실험을 통해 논문 발표 및 실험적 검증이 용이
- 저장장치, 임베디드, 머신러닝 분야의 융합적 연구로의 발전 가능성 확보

2. 기술 배경 및 선행 연구

2.1 SSD 및 FTL 기본 구조와 한계

- SSD 내부 구조

  SSD는 플래시 메모리 칩(NAND Flash)과 이를 제어하는 컨트롤러, 그리고 DRAM 캐시로 구성된다. 플래시 메모리는 페이지(page) 단위로 쓰기/읽기하고, 블록(block) 단위로 지우기(erase)를 한다. 페이지는 플래시에서 가장 작은 쓰기·읽기 단위로, 한 번 프로그래밍된 페이지는 같은 블록이 지워지기 전까지는 다시 덮어쓸 수 없다. 따라서 페이지 단위로 데이터를 쓰려면 내부적으로 빈 페이지를 찾아 1→0 프로그램을 수행한다. 블록은 여러 페이지를 묶은 단위로, 가장 작은 지우기(erase) 단위이다. 블록 전체를 한꺼번에 지워야만(모든 비트를 1로 복원) 그 안의 페이지들을 다시 사용할 수 있다. 각 블록은 지울 수 있는 횟수가 정해져 있어, 지우기 단위가 클수록(블록 내 페이지 수 많을수록) 동일 데이터 갱신 시 더 빠르게 수명 한계에 도달하게 된다. 컨트롤러 내 FTL(Flash Translation Layer)은 호스트가 요청한 논리 블록 주소(LBA)를 물리 페이지 주소(PBA)로 매핑하며, 이 과정에서 가비지 컬렉션(GC), 웨어 레벨링(WL) 등을 관리한다. [3]
- 쓰기 증폭(Write Amplification)

  호스트는 페이지 단위로 쓰기 요청을 주지만, FTL은 블록 단위로만 지울 수 있어

“블록 내 일부 유효 페이지를 보존 → 나머지 페이지 지우기 → 보존할 데이터 재프로그래밍” 과정을 거친다. 이 읽기→지우기→다시쓰기(read–modify–write) 사이클이 추가되며, 결과적으로 실제 플래시에 기록되는 데이터량이 호스트 요청량보다 커지는 쓰기 증폭(write amplification)이 발생한다.

- 가비지 컬렉션(GC) 오버헤드

  블록 지우기를 위해 유효 데이터가 섞인 블록을 선택하면, 해당 블록의 모든 유효 페이지를 다른 블록으로 복사한 뒤 원블록을 지운다. 이 복사 작업이 I/O 대기 시간을 증가시키고, SSD의 지연(latency) 편차(long-tail latency)를 심화시킨다.

  이러한 GC나 WL이 임의 시점에 동작하면서 호스트 I/O 응답 시간에 큰 변동을 초래한다. 특히 QoS가 중요한 실시간 시스템에서 문제로 작용할 수 있다.

2.2 입출력 워크로드 특성

- 시간적 지역성(Temporal Locality)

  동일한 논리 주소에 대한 반복 접근이 빈번하다. 이 특성은 워킹셋 크기, 재참조 간격(inter-reference gap) 분석을 통해 정량화할 수 있으며, SSD 캐시 정책 설계에 활용된다.

- 공간적 지역성(Spatial Locality)

  인접한 주소 범위를 순차적으로 읽거나 쓰는 패턴으로, 순차 I/O 비율과 연속 구간 길이(run-length)로 측정된다. 순차 쓰기는 쓰기 증폭을 최소화하는 반면, 랜덤 쓰기는 증폭을 심화시킨다.

- 순차 vs 랜덤 I/O 비율

  순차 I/O가 많을수록 FTL이 대규모 병합(write coalescing)에 유리하여 성능이 향상되는 반면, 랜덤 I/O 비율이 높으면 GC 오버헤드가 증가하여 지연 및 쓰기 증폭이 악화된다.

2.3 선행연구 요약

Agrawal 등(2008)의 "Design Tradeoffs for SSD Performance" 논문은 SSD 내부 구조와 성능 최적화를 위한 설계 선택지에 대한 체계적인 분류와 분석을 제시한다. 이 논문은 SSD 제조사들이 직면하는 다양한 설계 트레이드오프를 정리하고, 실제 시스템에서 수집한 워크로드 트레이스와 트레이스 기반 시뮬레이터를 활용해 여러 SSD 구성의 성능을 평가하였다.

핵심적으로, SSD의 성능과 수명은 워크로드 특성에 매우 민감하며, 기존 저장장치 계층(예: 파일시스템, 분산시스템)에서 발생했던 복잡한 시스템 문제가 SSD 펌웨어 수준에서도 동일하게 나타남을 보였다. 논문에서는 데이터 배치, 칩 간 병렬성, NAND 플래시의 쓰기 순서 제약, 워크로드 관리 등 네 가지 주요 설계 요소가 SSD 성능에 결정적인 영향을 미친다고 분석했다. 예를 들어, 데이터 배치는 부하 분산과 wear-leveling(마모 균등화)을 위해 중요하며, 칩 간 병렬 처리는 단일 플래시 칩의 한계를 극복하는 데 필수적이다. 또한, 작은 랜덤 쓰기와 같이 NAND 플래시의 특성에서 비롯된 문제는 SSD 설계에서 특히 까다로운 이슈로 지적된다.

논문은 DiskSim 기반 SSD 시뮬레이터를 활용해, TPC-C, Exchange 서버, 다양한 파일시스템 벤치마크 등 실제 워크로드에서 설계 선택이 성능과 수명에 미치는 영향을 분석했다. 그 결과, SSD의 하드웨어 및 소프트웨어 구성과 워크로드 특성이 복합적으로 작용해 최적의 설계가 달라질 수 있음을 밝혔다. 또한, plane interleaving, 오버프로비저닝, wear-leveling 알고리즘 등 다양한 기법의 효과를 실험적으로 제시하며, SSD 성능 최적화를 위해서는 워크로드 특성을 고려한 설계가 필수적임을 강조했다. [3]

3. 연구 내용 및 방법론

3.1 데이터 수집 및 전처리 계획

- 블록 I/O 트레이스 수집

  현실적이고 대표성 있는 블록 I/O 패턴 확보를 위해 SNIA IOTTA Repository에서 다양한 저장장치·운영체제 워크로드를 다운로드 후 원시 트레이스를 확보한다.
- 시뮬레이터 로그 생성

  SimpleSSD-SA 시뮬레이터를 구축하여, 실측 트레이스의 보완 및 FTL 정책 실험에 사용할 입출력 로그를 생성한다. 순차, 랜덤, 혼합(70/30) 접근 패턴, 블록 크기(4 KiB - 1 MiB) 등 다양한 시나리오 기반으로 실험 데이터를 수집할 계획이다.
- 전처리

  모델 입력에 적합한 피처 세트를 구축하기 위해, 타임스탬프 정규화와 이상치 제거를 수행한다. 노이즈 제거 및 처리 시간 단축을 위해 비정상적으로 지연된 I/O 레코드를 필터링하고, 대용량 트레이스는 시간 창 단위 또는 요청 수 기준으로 샘플링하여 처리한다.
- 피처 엔지니어링 및 hot/cold 분류

  주요 입력 변수로서 timestamp, operation type, sector number, I/O size 등의 피처를 추출하고, 접근 빈도 기반으로 Hot/Cold 데이터를 분류한다.
- 데이터 분할

  과적합 방지 및 일반화 성능 확보를 위해 시계열 순서 보존한 split-by-time 방식으로 훈련(70%)·검증(15%)·테스트(15%) 세트를 분리하여 검사한다.

3.2 딥러닝 모델 설계 및 적용 방안

- 모델 선택

  시계열 I/O 패턴 예측에는 LSTM(Long Short-Term Memory)이나 Transformer

기반 모델이 적합하다. LSTM은 과거 I/O 시퀀스를 학습해 미래 워크로드를 예측하며, Transformer는 더 긴 의존성 및 병렬 처리가 가능하다.

- 입력 피처 및 라벨링

  입력 피처로는 타임스탬프, 연산 종류(Read/Write), 섹터 번호, I/O 크기 등을 사용한다. Hot/Cold 분류, 다음 I/O 타입/위치 예측 등 다양한 태스크에 맞는 라벨을 구성한다.

- 학습 및 튜닝

  훈련 데이터로 모델을 학습시키고, 검증 세트로 하이퍼파라미터(레이어 수, hidden size, learning rate 등)를 조정한다. 과적합 방지를 위해 드롭아웃, 조기 종료(Early Stopping) 등 기법을 적용한다.

- 워크로드 패턴 예측 및 분류

  학습된 모델을 통해 미래 I/O 요청의 Hot 영역, 랜덤/순차 패턴, 쓰기 집중 구간 등을 예측한다. 예측 결과는 FTL 정책에 실시간으로 피드백된다.

3.3 FTL 성능 개선 전략

- 정책 연동

  예측된 워크로드 정보를 바탕으로 FTL의 블록 할당, 가비지 컬렉션(GC) 시점, Wear Leveling 우선순위, 캐싱 전략 등을 동적으로 조정한다. 예를 들어, Hot 데이터는 별도의 블록에 집중 배치해 GC 오버헤드를 줄이고, Cold 데이터는 장기 보존 블록에 할당한다.

- 지능형 매핑

  LSTM 등 딥러닝 기반 온도(Hot/Cold) 예측 결과를 활용해 데이터 온도별로 물리 블록을 분리 배치하거나, Learned Index 기반 주소 매핑 최적화 기법(LearnedFTL 등)을 적용해 읽기/쓰기 지연을 최소화한다.

- 정책 적용 구조

  기존 정적 FTL 정책과 비교해, 예측 기반 동적 FTL 정책을 시뮬레이터에 구현해 실험한다.

3.4 성능 평가 및 검증 계획

- 실험 환경

  SimpleSSD-SA SSD 시뮬레이터에 실제 트레이스와 시뮬레이터 생성 로그를 입력해, 기존 FTL과 딥러닝 기반 FTL의 성능을 비교한다.

- 평가 지표
  - 쓰기 증폭(Write Amplification Factor)
  - 처리량(IOPS, MBps)
- 성능 분석

  각 실험 결과를 바탕으로 딥러닝 기반 정책의 효과(지연 감소, 쓰기 증폭 완화 등)를 정량적으로 분석한다. 필요시 다양한 워크로드(순차, 랜덤, 혼합)별로 세부 성능을 비교한다.

4. 연구 일정 및 추진 계획

4.1 연구 단계별 일정

업무	4				5				6				7				8				9			
	1	2	3	4	1	2	3	4	1	2	3	4	1	2	3	4	1	2	3	4	1	2	3	4
기획 및 설계																								
착수보고서 작성																								
필요 지식 습득																								
데이터 전처리																								
모델 개발																								
입출력 패턴 코드 분석																								
FTL 개선																								
시뮬레이션																								

4.2 팀원별 역할 분담

이준형	문서작성 모델 개발 입출력 패턴 코드 분석 FTL 개선 시뮬레이션
강인석	데이터 전처리 모델 개발 입출력 패턴 코드 분석 FTL 개선 시뮬레이션
김지수	자료조사 모델 개발 입출력 패턴 코드 분석 FTL 개선 시뮬레이션

5. 연구 리스크 및 대응 방안

5.1 예상되는 문제점 및 한계

본 연구에서 제안하는 딥러닝 기반 SSD 성능 개선 기법은 시뮬레이터 환경에서 검증할 수 있으나, 실제 SSD 하드웨어에 바로 적용하기에는 다음과 같은 한계점이 존재한다.

- 실제 SSD 하드웨어 적용의 어려움
  연구에서 개발한 모델 및 제어 정책은 소프트웨어 시뮬레이터(SimpleSSD-SA 등)에서만 실험·평가가 가능하며, 상용 SSD 펌웨어나 하드웨어에 직접 적용할 수 있는 인터페이스가 제공되지 않는다. SSD 펌웨어는 제조사별로 폐쇄적으로 관리되고, 실시간/임베디드 환경에서 동작해야 하므로, 연구 결과의 실질적 상용화 및 적용에는 추가적인 기술적·제도적 장벽이 있다.
- 모델의 경량화 및 효율성 검증 한계
  제안된 딥러닝 모델이 실제 SSD 컨트롤러(임베디드 환경)에서 효율적으로 동작할 수 있을지, 즉 연산 자원(메모리, CPU 등) 소모가 허용 범위 내에 있는지 시뮬레이터만으로는 확인이 어렵다. 실제 하드웨어 환경에서는 모델의 경량화(모델 크기 축소, 연산 최적화 등)가 필수적이지만, 현재 연구 범위에서는 이 부분의 실효성을 충분히 검증하기 어렵다.
- 입출력 패턴 변화에 대한 일반화 한계
  연구에 사용된 데이터셋(워크로드 트레이스)과 실험 환경은 특정 시나리오에 한정되어 있어, 실제 운영 환경에서 워크로드 패턴이 급격히 변화할 경우(예: 새로운 애플리케이션 도입, 사용자 행동 변화 등) 모델이 동일한 수준의 성능을 보장할 수 있을지 확신하기 어렵다. 즉, 학습된 모델의 일반화 성능과 적응성에 대한 추가 검증이 필요하다.

이와 같은 한계점들은 향후 실제 SSD 펌웨어 적용을 위한 후속 연구, 임베디드 환경에서의 모델 경량화, 다양한 실환경 워크로드에 대한 추가 실험 등을 통해 보완될 필요가 있다.

5.2 대응 전략 및 대안

실제 SSD 하드웨어 적용의 한계, 모델 경량화 검증의 어려움, 입출력 패턴 변화에 대한 불확실성 등 앞서 제기된 문제점에 대응하기 위한 전략과 대안은 다음과 같다.

- 실제 하드웨어 적용 한계에 대한 대응
    - 시뮬레이터 기반의 반복적 실험을 통해 다양한 워크로드와 환경에서 모델의 성능과 안정성을 충분히 검증한다.
    - 연구 결과를 실제 펌웨어 개발사와 협력하거나, 오픈소스 FTL 구현체에 적용해 실증 사례를 확보한다.
- 모델 경량화 및 효율성 확보
    - 임베디드 환경에서 동작 가능한 경량 딥러닝 모델(Lightweight LSTM, TinyML 등)로 구조를 단순화하고, 연산량 및 메모리 사용량을 최소화한다.
    - 모델 경량화 기법(양자화, 프루닝 등)과 캐시 기반의 워크로드 예측 기법을 병행 적용해 실제 SSD 컨트롤러의 자원 제약 내에서 동작 가능성을 높인다.
    - 시뮬레이터에서 모델 연산 시간과 메모리 사용량을 측정해, 하드웨어 적용 전 단계에서 실질적 오버헤드를 예측·평가한다.
- 입출력 패턴 변화에 대한 적응성 강화
    - 다양한 유형의 입출력 패턴(순차, 랜덤, 혼합 등)에 대해 사전 학습된 모델을 준비하거나, 패턴 변화 감지 시 동적으로 모델을 전환하는 메커니즘을 설계한다.
    - 실제 운영 환경에서 수집되는 새로운 트레이스를 활용해, 모델의 일반화 성능을 지속적으로 검증하고 필요시 추가 학습을 수행한다.

이러한 전략을 통해, 연구의 실효성과 확장성을 높이고 실제 SSD 시스템에 적용 가능한 기반을 마련할 수 있다.

6. 참고문헌

[1]"IOTTA Repository: Input/Output Traces, Tools, and Analysis," SNIA. [Online]. Available: https://iotta.snia.org/. [Accessed: May 9, 2025].

[2] "SimpleSSD: Open-Source Licensed Full-System SSD Simulator," SimpleSSD, https://docs.simplessd.org/en/v2.0.12/ (accessed May 9, 2025).

[3] AGRAWAL, Nitin, et al. Design tradeoffs for {SSD} performance. In: *2008 USENIX Annual Technical Conference (USENIX ATC 08)*. 2008.