

LearnedFTL: A Learning-based Page-level FTL for Reducing Double Reads in Flash-based SSDs

Shengzhe Wang, Zihang Lin, Suzhen Wu, Hong Jian, Jie Zhang, Bo Mao

Computer Science and Engineering Department, University of Texas at Arlington, Arlington, TX, USA *School of Computer Science, Peking University

2024 IEEE International Symposium on High-Performance Computer Architecture (HCPA)

2024.08.07

Presentation by Minseong Kim & Dayeon Wee
kms0509@dankook.ac.kr, wida10@dankook.ac.kr

Contents

1. Introduction
2. Background and Motivation
3. Design
4. Evaluation
5. Conclusion

1. Introduction

- 3D NAND와 NVMe기술로 SSD 성능 ↑
→ But, 기존의 FTL의 이중읽기 문제로 랜덤읽기 성능 ↓
- 랜덤 워크로드에 대한 성능을 개선하고자 LearnedFTL을 제안
 - ① $\text{LearnedFTL} = \text{TPFTL} + \text{Learned Index}$
 - ② In-place 업데이트 선형 모델
 - ③ 가상 PPN (=VPPN)
 - Group-based allocation

2. Background and Motivation

- DFTL

- 워크로드의 시간적 지역성 활용
- 성능 유지, 메모리 사용량 ↓
- 캐시 미스 시, 이중읽기 문제 발생
- 대표적 예시인 TPFTL: 시간적 및 공간적 지역성을 모두 활용

2. Background and Motivation

CMT = 캐시된 매핑 테이블
GTD = 글로벌 변환 디렉토리

DFTL

1

LPN1005의 요청

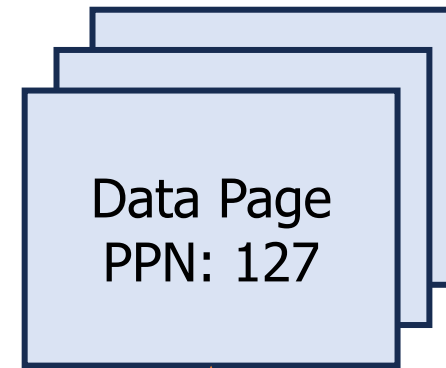
DRAM

CMT	
LPN	PPN
15	104
...	...
...	...

GTD	
LPN	TPPN
0-511	224
512-1023	225
...	..

2

FLASH



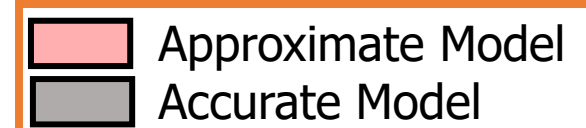
4

TransPage: 225	
LPN	PPN
1005	127
...	..

Double Read

3

2. Background and Motivation



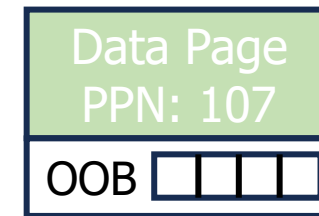
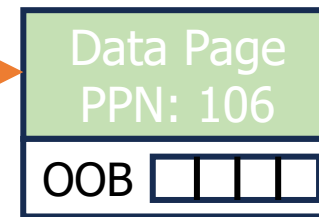
LeaFTL

DRAM

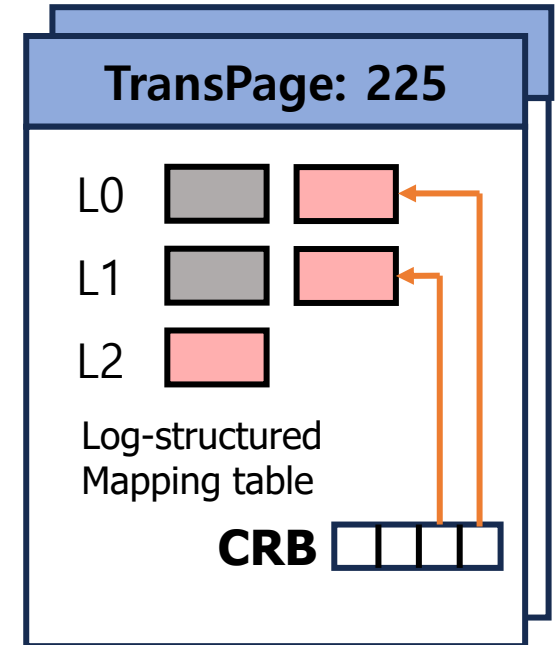
Model Cache	
LPN	Segment
0-255	Model
4096-4607	Model
...	..

GTD	
LPN	TPPN
0-255	223
256-511	225
...	...

FLASH



2 Double Read



1 Inaccurate PPN

2. Background and Motivation

- CMT 크기 증가
→ 성능이 기대만큼 좋진 않음
- 매핑 테이블 압축
→ SSD 쓰기 성능 ↓

500배 증가시켰지만
성능이 기대에 못미침

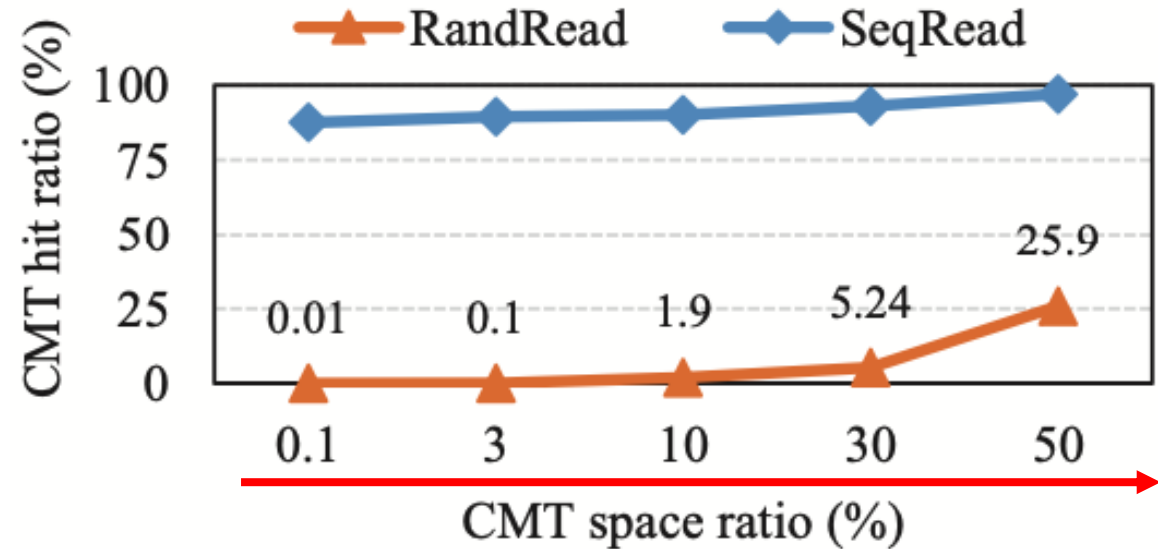
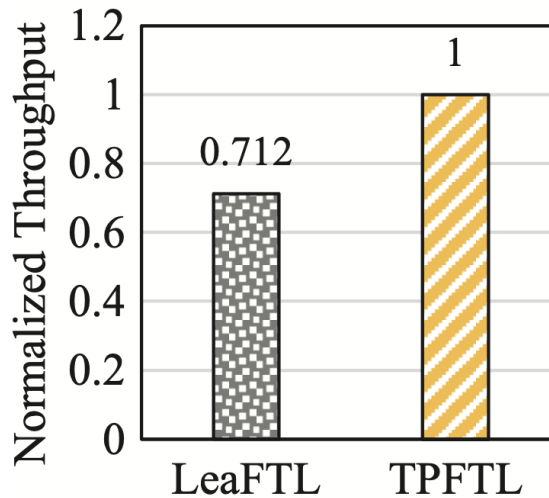


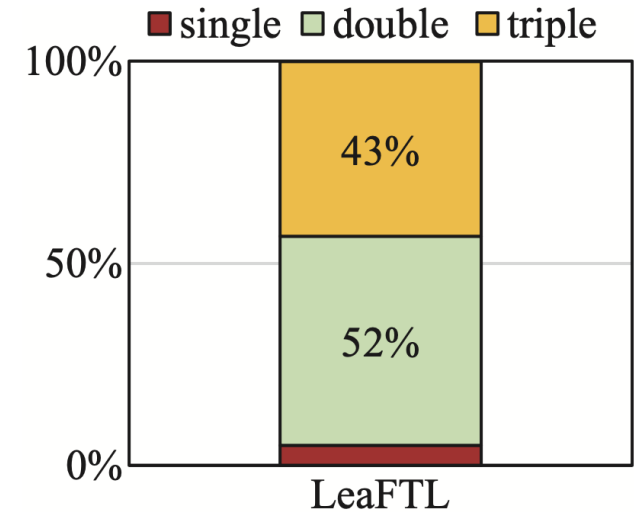
Fig. 3: The hit ratio of TPFTL under different CMT space.

2. Background and Motivation

- Learned Index의 정확성은 주소 변환의 효율성 결정
 - 잘못된 예측은 LeaFTL에서의 이중/삼중읽기 유발
 - 랜덤에서 성능 ↓



(a) Random read

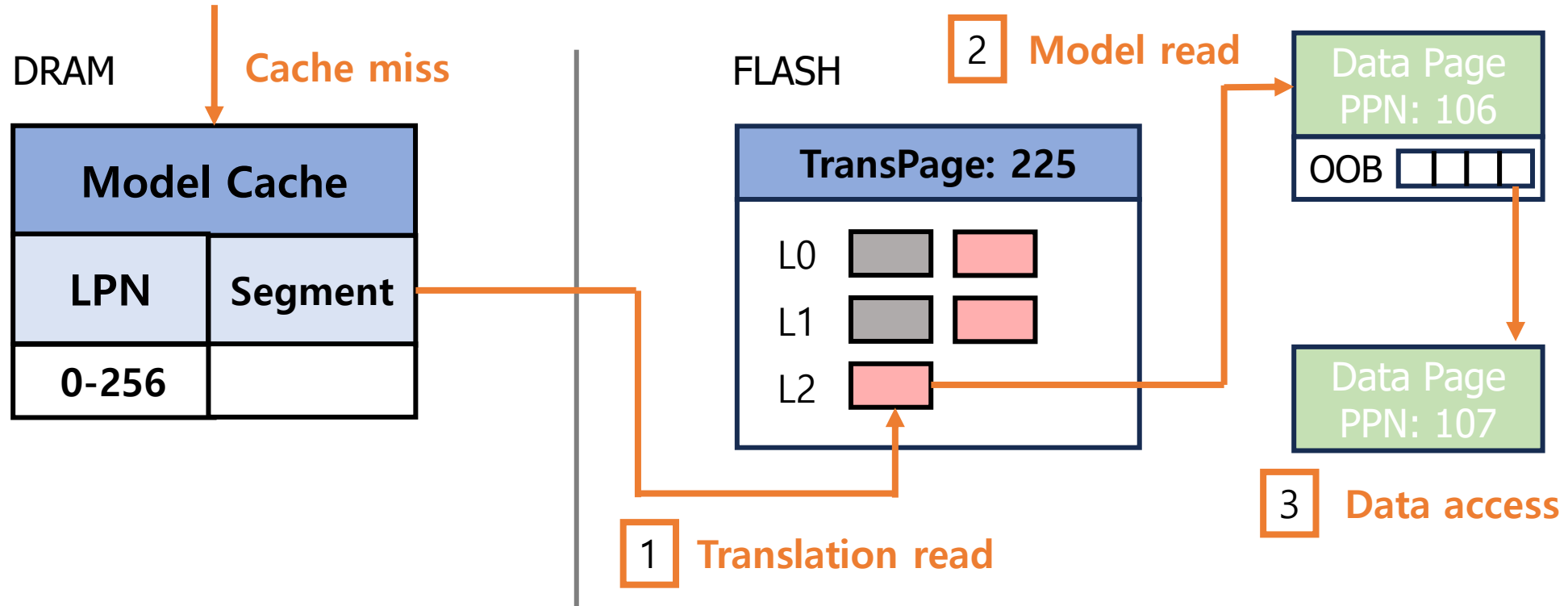


(b) Multi-read count statistics

Fig. 6: The performance results under random reads.

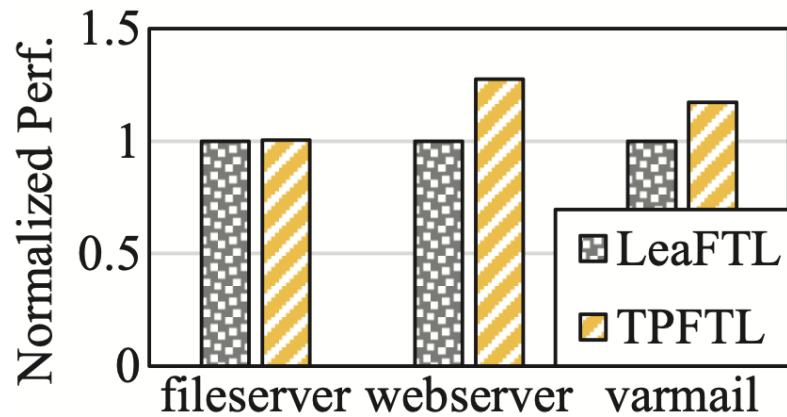
2. Background and Motivation

- LeaFTL에서의 삼중읽기

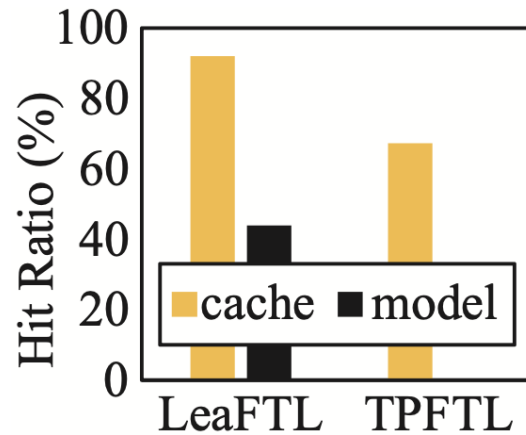


2. Background and Motivation

- TPFTL과 LeaFTL을 세 가지 filebench 워크로드(sequential)에서 비교



(a) Throughput



(b) Hit ratio in webserver

LeaFTL

- Cache는 거의 100% 히트
- 정확한 건 40%
- 나머지는 이중/삼중 읽기를 통해 탐색

Fig. 7: The performance of TPFTL and LeaFTL under workloads with high locality.

2. Background and Motivation

■ ① 선형 모델과 병렬성 간의 충돌

- 내부 병렬성 때문에 연속된 LPN에 연속된 PPN을 할당하기 어려움
- 선형 모델 훈련 시 SSD의 병렬성을 맞추는 데 문제가 될 수 있음

■ ② 학습에서의 오버헤드

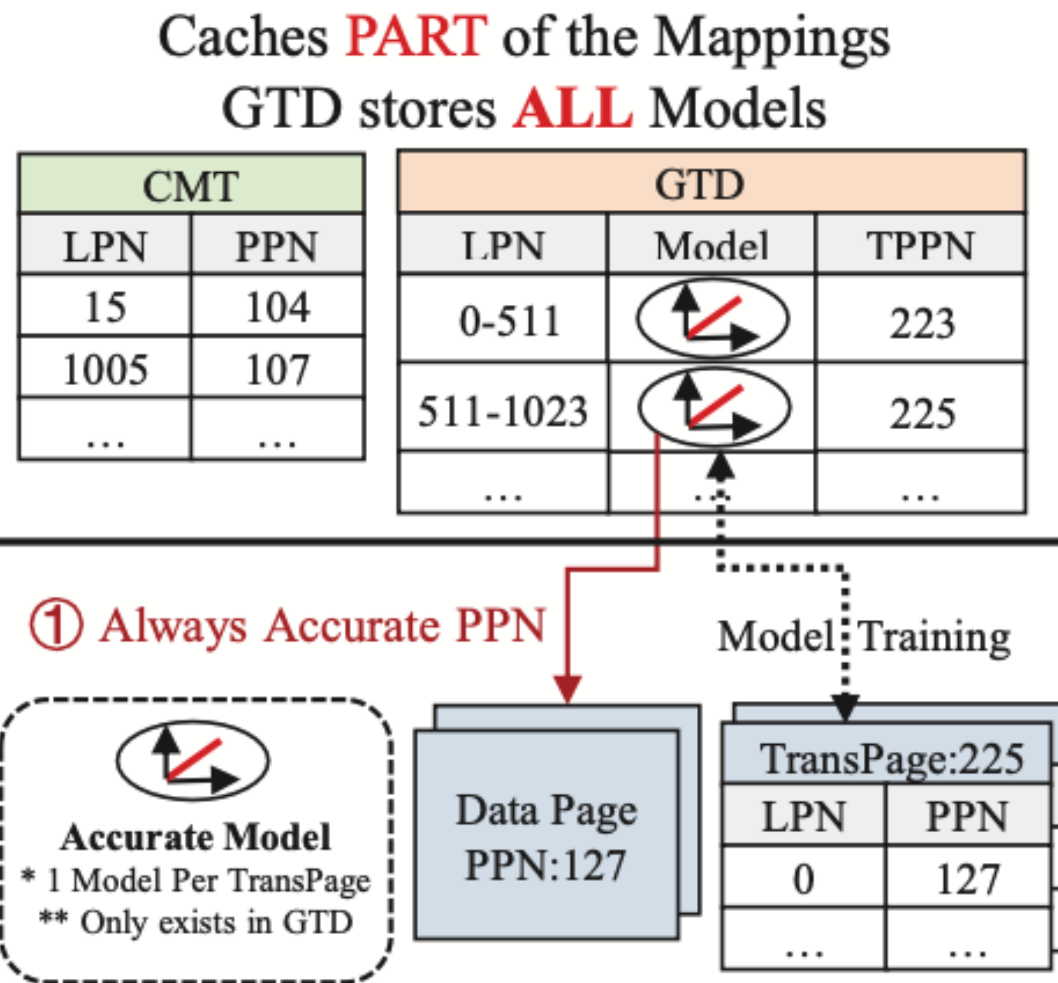
- 성능 오버헤드
 - 디스크에 쓰는 중에 모델 학습이 실행되면 쓰기 시간 ↑
- 공간 오버헤드
 - 랜덤 접근에서 수행하게 되면, 인접한 LPN이 물리적으로 분리
 - 모델 학습 시에도 개별 학습 세그먼트로 처리될 수 있음

2. Background and Motivation

- DFTL과 LeaFTL의 한계점
 - 모델 예측의 부정확성
 - 캐시 미스에 따른 성능 저하
 - 모델 업데이트의 어려움
 - 액세스 병렬성과의 충돌

3. Design

- 논문에서 제안하는 것
 - 랜덤 읽기로 인해 발생하는 이중읽기 문제
 - 순차는 **DFTL + 랜덤은 Learned Index**



Model Hits **Eliminate** Double Reads

(c) LearnedFTL

3. Design

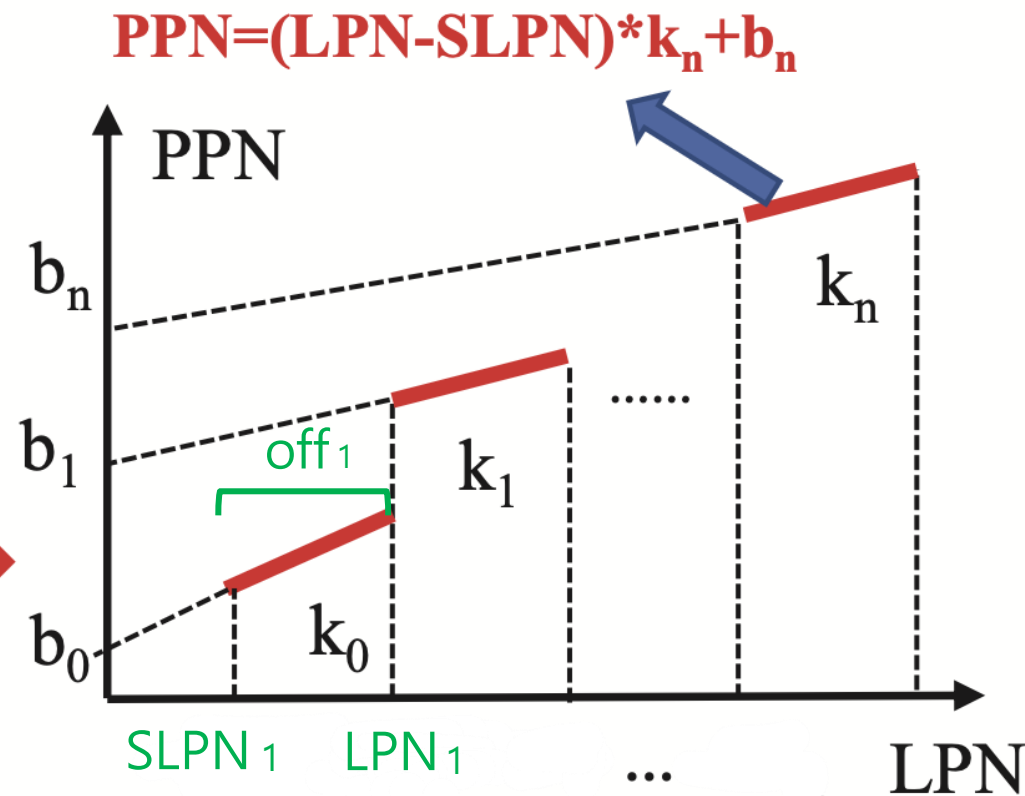
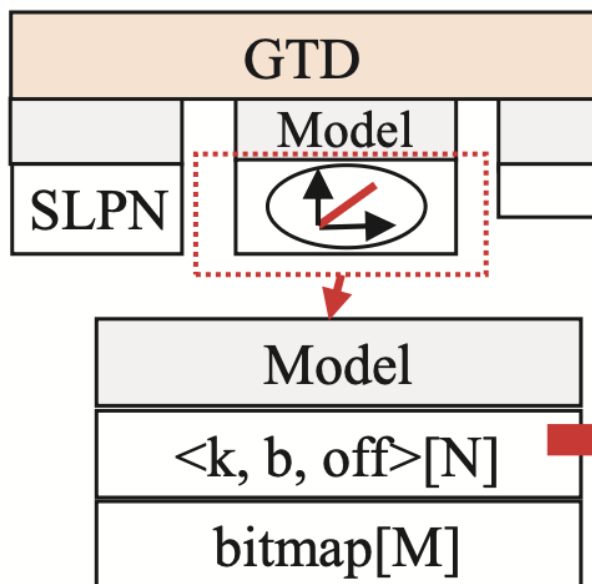
■ 제안하는 설계방법

- ① 모델 예측의 정확성 보장
 - 비트맵 사용
- ② 정렬된 LPN에 대해 연속적인 PPN 얻기
 - 가상 PPN 표현 사용
- ③ 공간 / 성능 오버헤드 줄이기
 - 그룹 기반 할당 방식
 - 순차 초기화 + GC /재작성을 통한 모델 학습

3. Design

■ In

- <
- k
- p



$$PPN = \underline{k_n} * (LPN - SLPN) + b_n$$

off 을 나타냄

bitmap filter

0	0	1	0	1	0	0	1	0	1
---	---	---	---	---	---	---	-------	---	---	---

Fig. 8: The structure of an in-place-update model in GTD.

3. Design

■ 비트맵 필터

- PPN이 정확한지
- 모델 매개변수
- 데이터 일관성
- 모델 저장 및 재

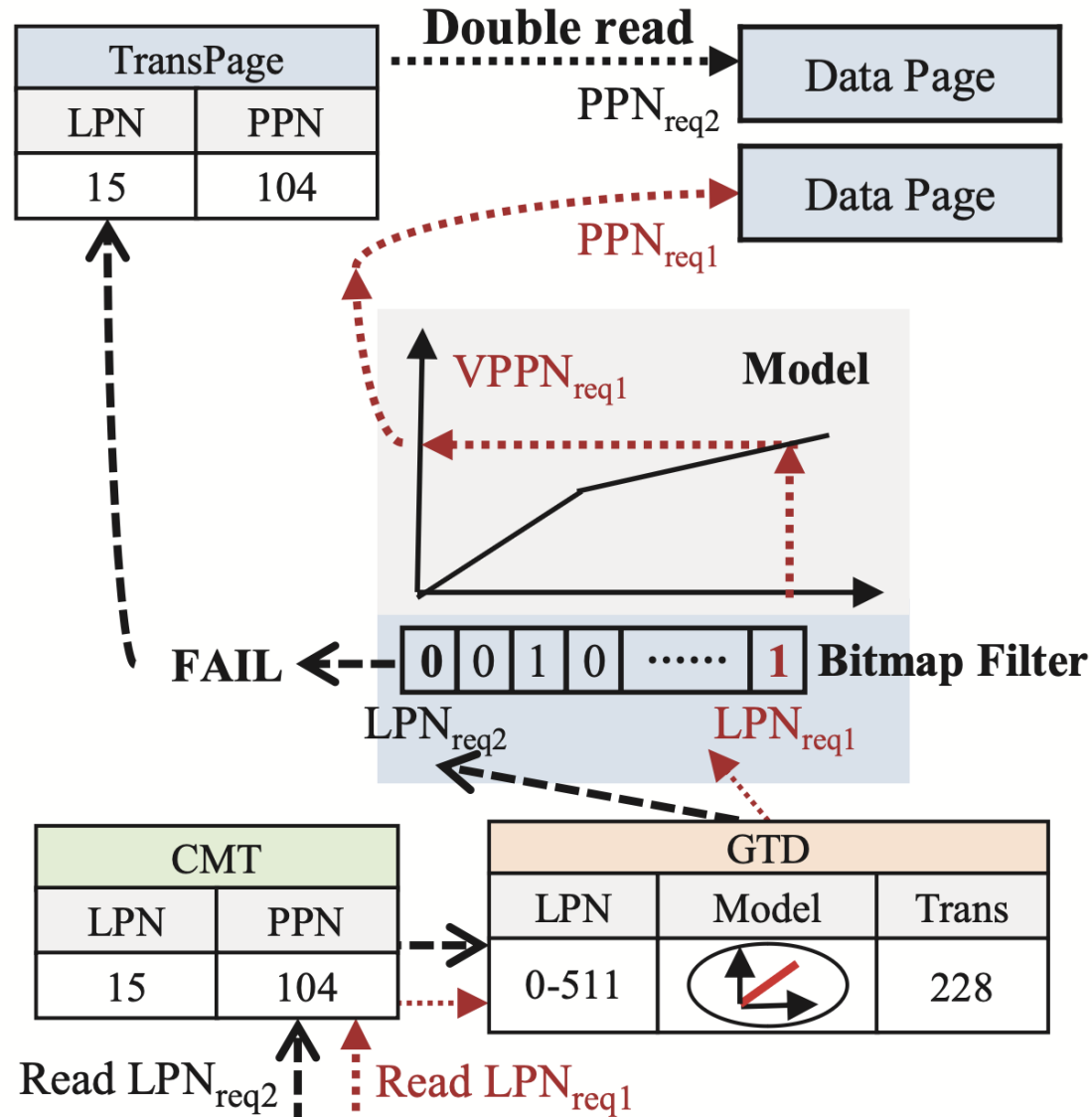


Fig. 9: The workflow of bitmap filter.

3. Design

- In-place 업데이트의 전체 흐름

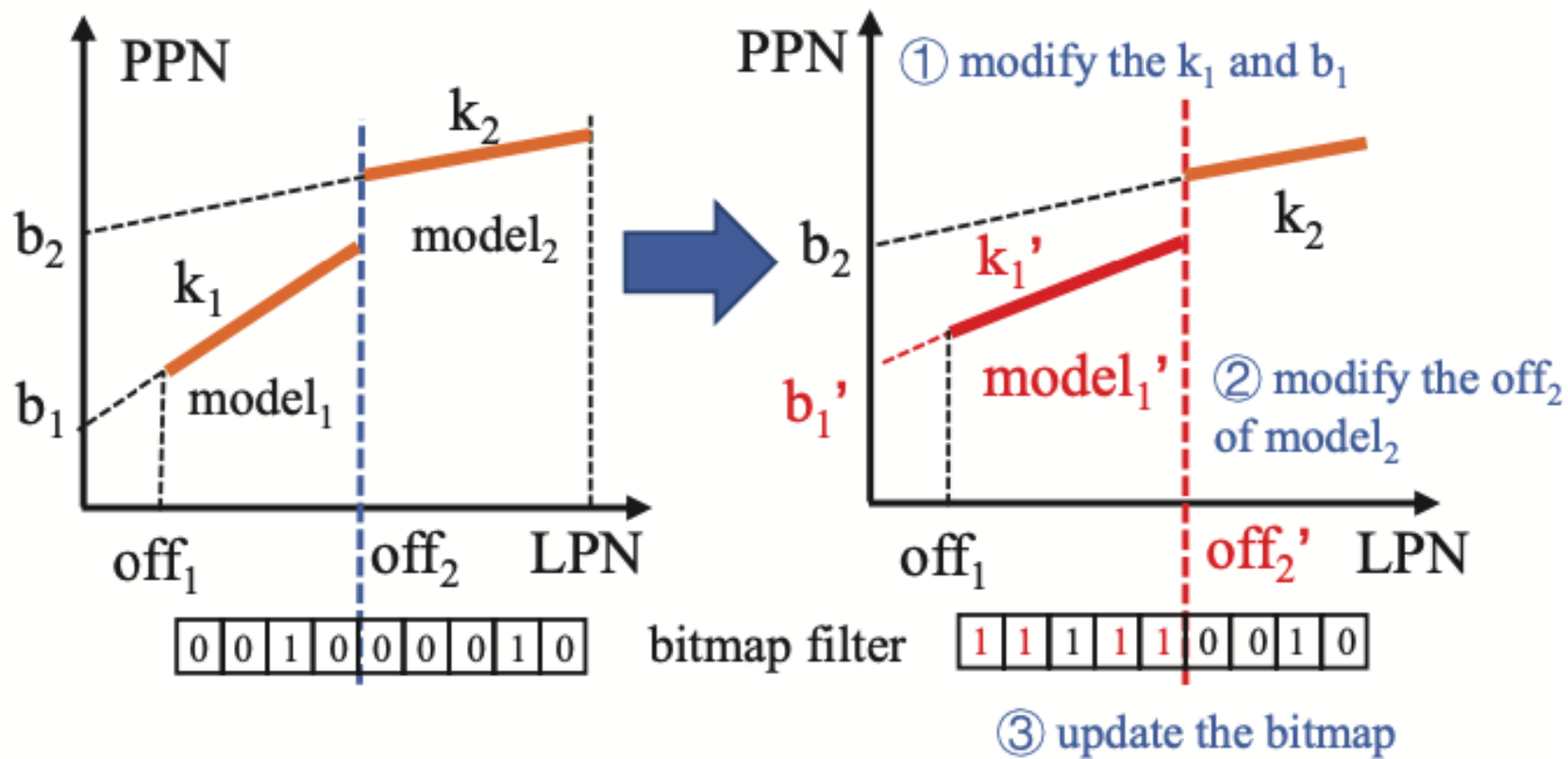
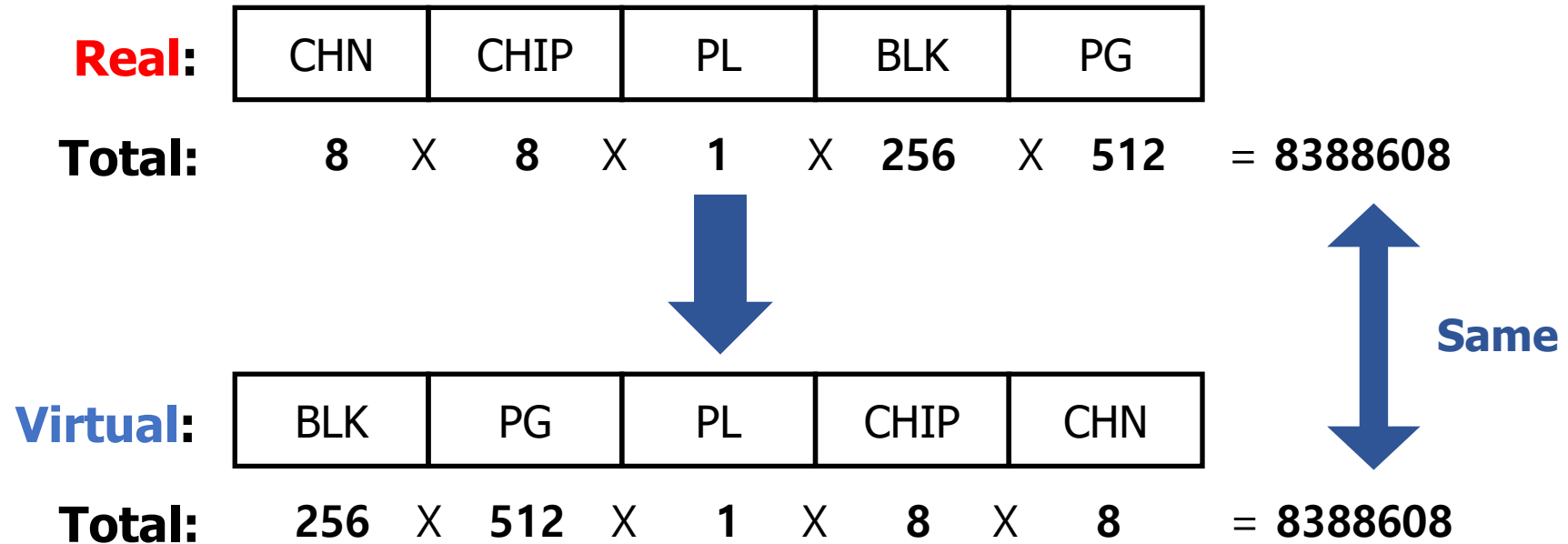


Fig. 10: The workflow of model in-place update.

3. Design

Virtual PPN Representation

- 연속적인 LPN에 연속적인 PPN을 할당하는 것은 모델 학습의 정확성 향상
- SSD의 병렬성 때문에 LPN에 연관된 page는 다른 flash chip에 저장
- PPN 주소 표현 방식이 채널부터 페이지 순으로 계산하여 불연속적인 PPN이 할당
- 불연속적인 PPN을 연속적인 VPPN으로 변환



3. Design

기존 방식

LPN	CHN	CHIP	PL	BLK	PG	True PPN			
	8	X	8	X	1	X	256	X	512
1001	4	5	1	64	127	=	5013631		

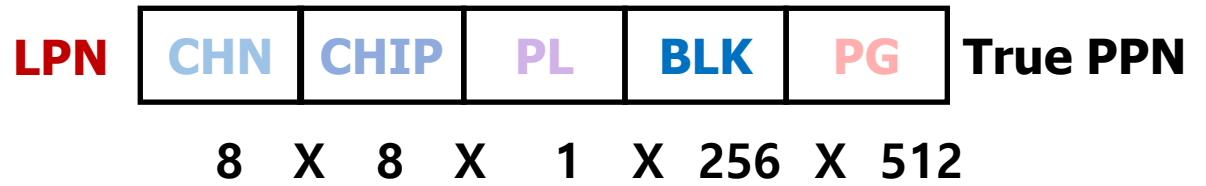
$$4 \times (8 \times 1 \times 256 \times 512) + 5 \times (1 \times 256 \times 512) + 1 \times (256 \times 512) + 64 \times 512 + 127 = 5013631$$

1002 5 5 1 64 127 = 6062207

1003 6 5 1 64 127 = 7110783

3. Design

기존 방식

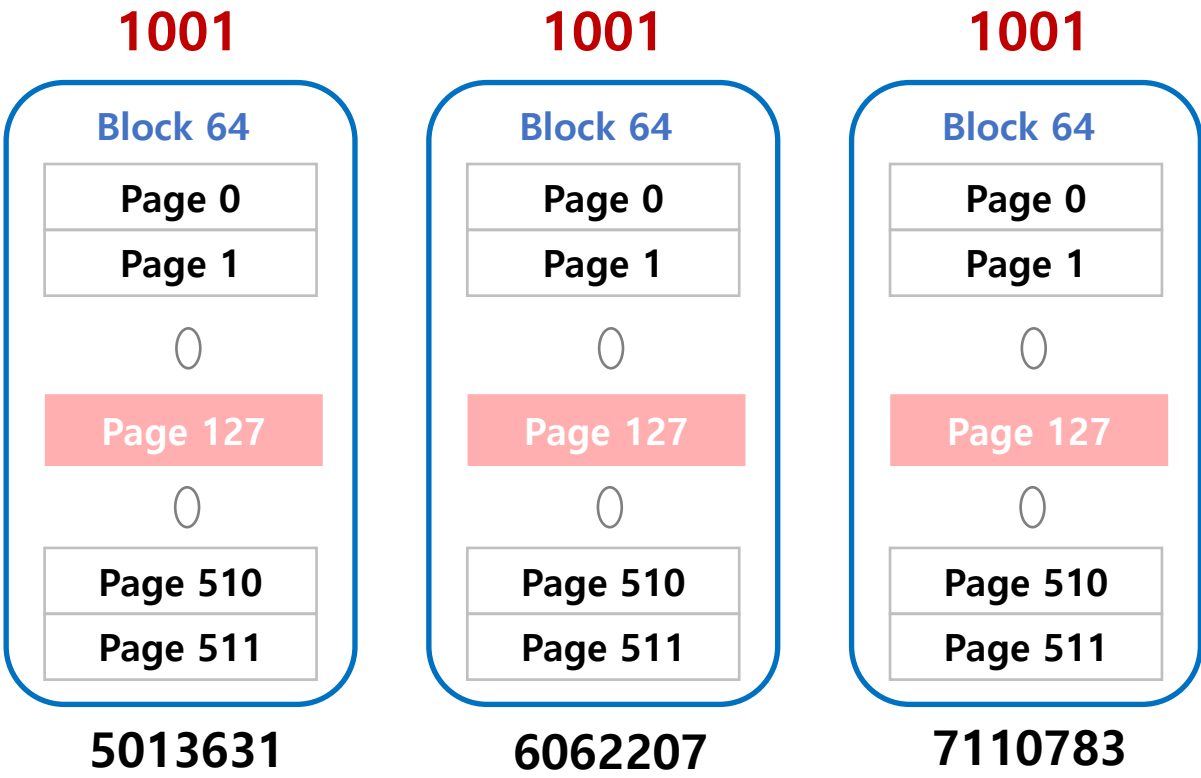


1001 4 5 1 64 127 = 5013631

$$4 \times (8 \times 1 \times 256 \times 512) + 5 \times (1 \times 256 \times 512) + 1 \times (256 \times 512) + 64 \times 512 + 127 = 5013631$$

1002 5 5 1 64 127 = 6062207

1003 6 5 1 64 127 = 7110783



3. Design

기존 방식

LPN	CHN	CHIP	PL	BLK	PG	True PPN
	8	X	8	X	1	X
				256		512
1001	4	5	1	64	127	= 5013631

$$4 \times (8 \times 1 \times 256 \times 512) + 5 \times (1 \times 256 \times 512) + 1 \times (256 \times 512) + 64 \times 512 + 127 = 5013631$$



Virtual PPN 적용한 방식

LPN	BLK	PG	PL	CHIP	CHN	VPPN
	8	X	8	X	1	X
				256		512
1001	64	127	1	5	4	= 2105388

$$64 \times (512 \times 1 \times 8 \times 8) + 127 \times (1 \times 8 \times 8) + 1 \times (8 \times 8) + 5 \times 8 + 4 = 2105388$$

$$1002 \quad 5 \quad 5 \quad 1 \quad 64 \quad 127 = 6062207$$

$$1001 \quad 64 \quad 127 \quad 1 \quad 5 \quad 5 = 2105389$$

$$1003 \quad 6 \quad 5 \quad 1 \quad 64 \quad 127 = 7110783$$

$$1001 \quad 64 \quad 127 \quad 1 \quad 5 \quad 6 = 2105390$$

3. Design

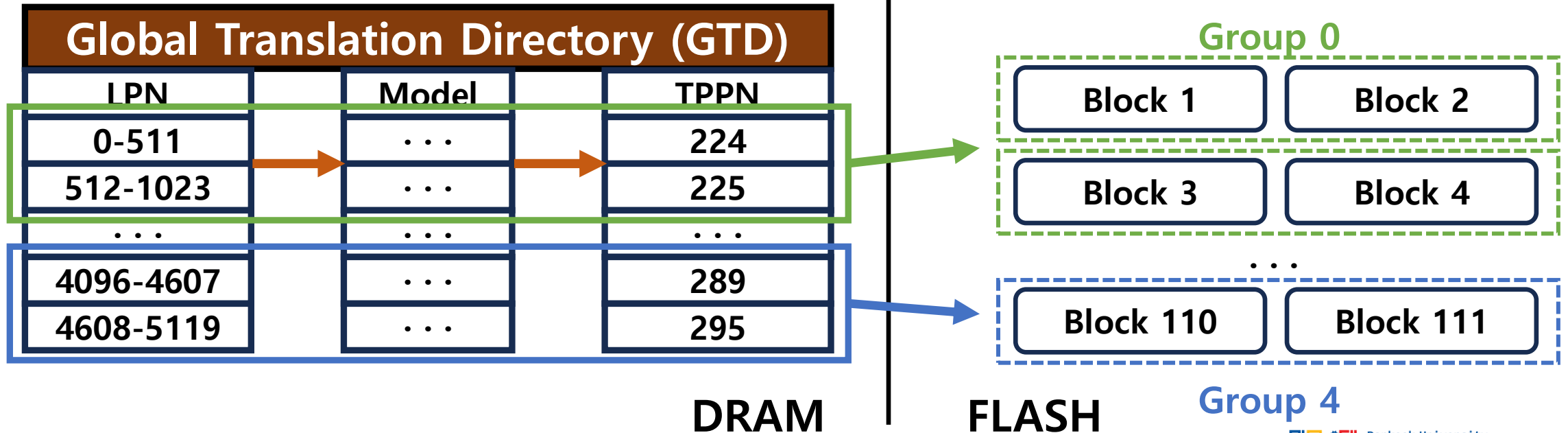
▪ Group-based Allocation Strategy

- Random write에서 많은 불연속적인 LPN 요청이 발생
- GC를 하며 GTD 항목의 PPN을 재정렬하여 모델 학습이 가능
 - But GTD 항목의 PPN이 다양한 위치에 할당되어 GC시 빈번한 데이터 이동이 발생

3. Design

▪ Group-based Allocation Strategy

- Random write에서 많은 불연속적인 LPN 요청이 발생
- GC를 하며 GTD 항목의 PPN을 재정렬하여 모델 학습이 가능
 - But GTD 항목의 PPN이 다양한 위치에 할당되어 GC시 빈번한 데이터 이동이 발생



3. Design

- Hot group은 자주 수정되므로 **WAF**가 발생
 - 각 GTD entry group에 global count를 두어 free page 수를 계산해 hot group을 구분
 - Hot group에 할당할 page가 없는 상황을 해결하기 위해 *Opportunistic cross-group allocation strategy* 사용
- *Opportunistic cross-group allocation strategy*
 - Cold group에 속하는 플래시 블록의 free page를 활용
 - Cold group의 free page를 hot group에 임계값 이상 할당하면 두 그룹이 GC를 작동
 - GC의 performance 향상, WAF 감소, learned index training 보장

3. Design

■ Model Training

- Sequential Initialization

- Sequential write을 기반으로 learned index model을 in-place 업데이트
- 각 I/O 요청에서 연속적인 LPN-PPN 매핑을 $y=x$ 같은 선형 모델로 업데이트
- 새로 생성된 모델의 길이(L_{new})가 기존 모델의 길이(L_{old})보다 길면 기존 모델을 업데이트

- Model Training via GC

- 더 포괄적이고 정확한 모델을 위해 GC를 통한 training 제안
- GTD entry group의 valid한 page를 LPN순으로 정렬 후 연속적인 PPN을 flash block에 다시 쓰고 이를 기반으로 in-place 업데이트 선형 모델을 학습한다.
- 그 후 모델을 평가한 후 bitmap filter를 업데이트한다.

4. Evaluation

- 실험 환경

Host setting

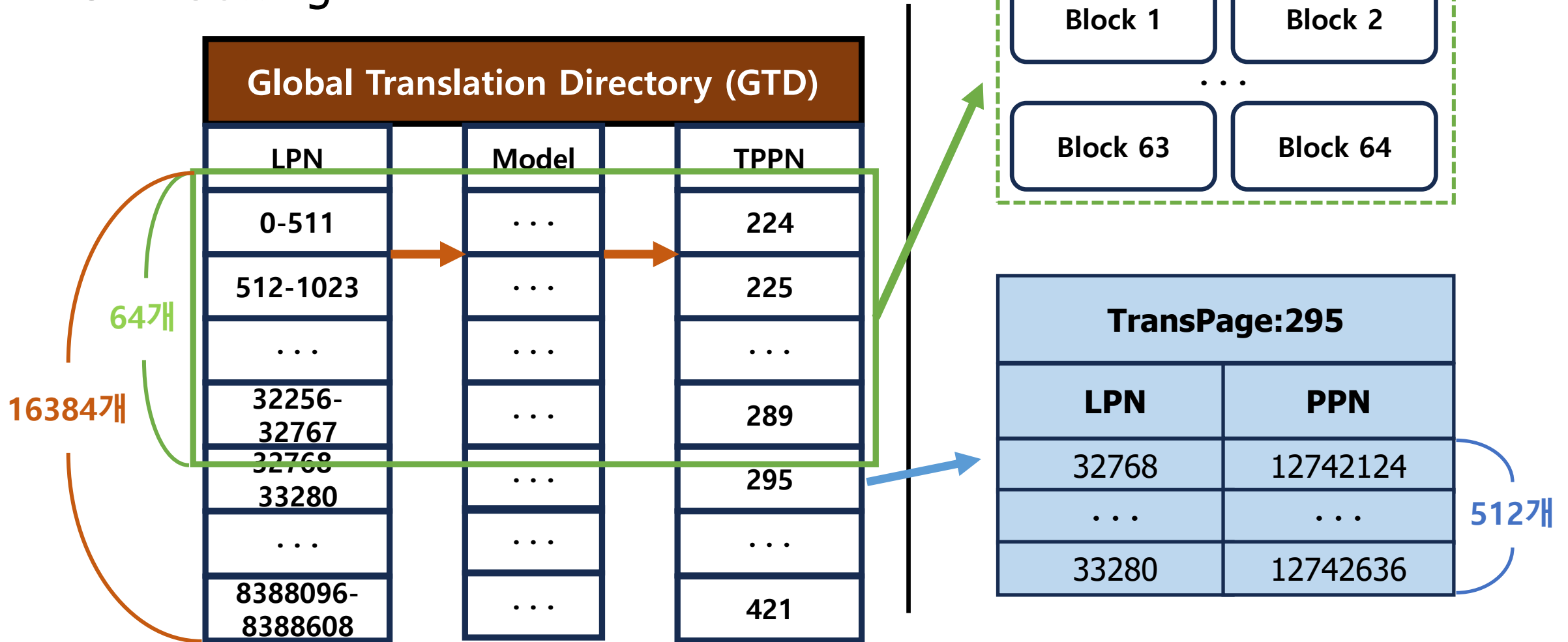
CPU	Intel(R) Xeon(R) Gold 5318Y 2.10GHz CPU
DRAM	128GB
OS	Linux kernel 5.4.0

FEMU Setting (SSD emulator)

Parameter	Value	Parameter	Value
Logical SSD	32GB + 2GB(provisioning)	#Page Page Size	512 4KB
#Channel	8	Read latency	40μs
#Chip	8	Write latency	200μs
#Block	256	Erase latency	2ms

4. Evaluation

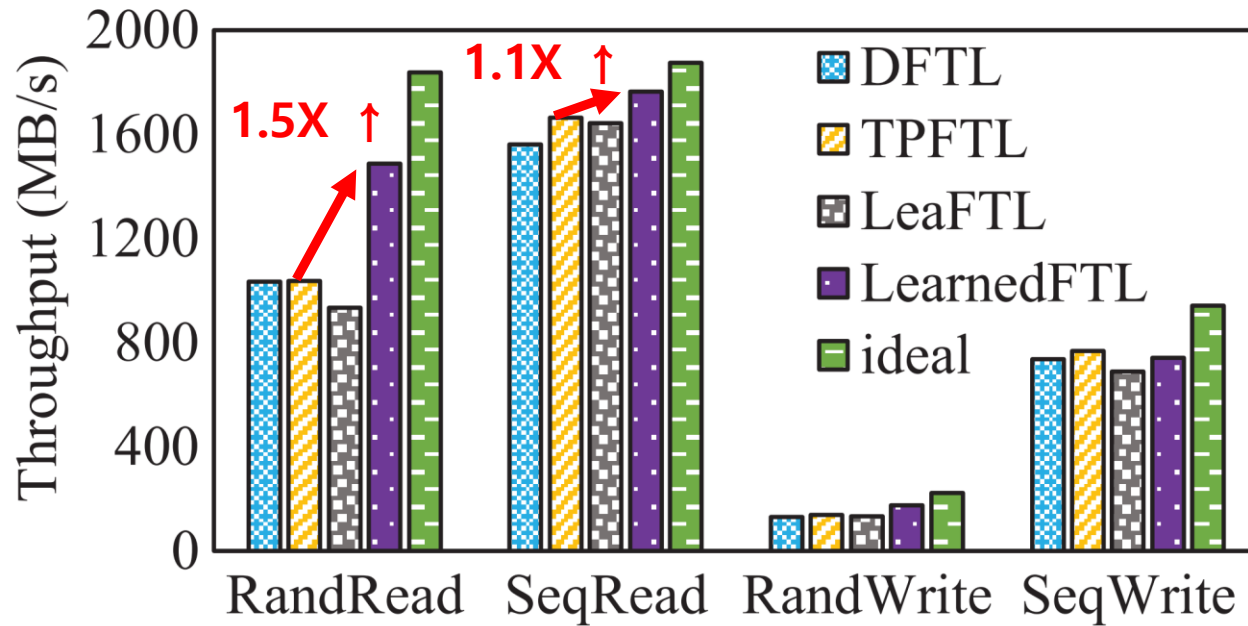
■ GTD Setting



4. Evaluation

▪ Read(FIO)

- Read 전 sequential/random write으로 워밍업
- 4KB I/O size, psync I/O engine을 사용해 64개의 threads에서 읽기 실험 진행

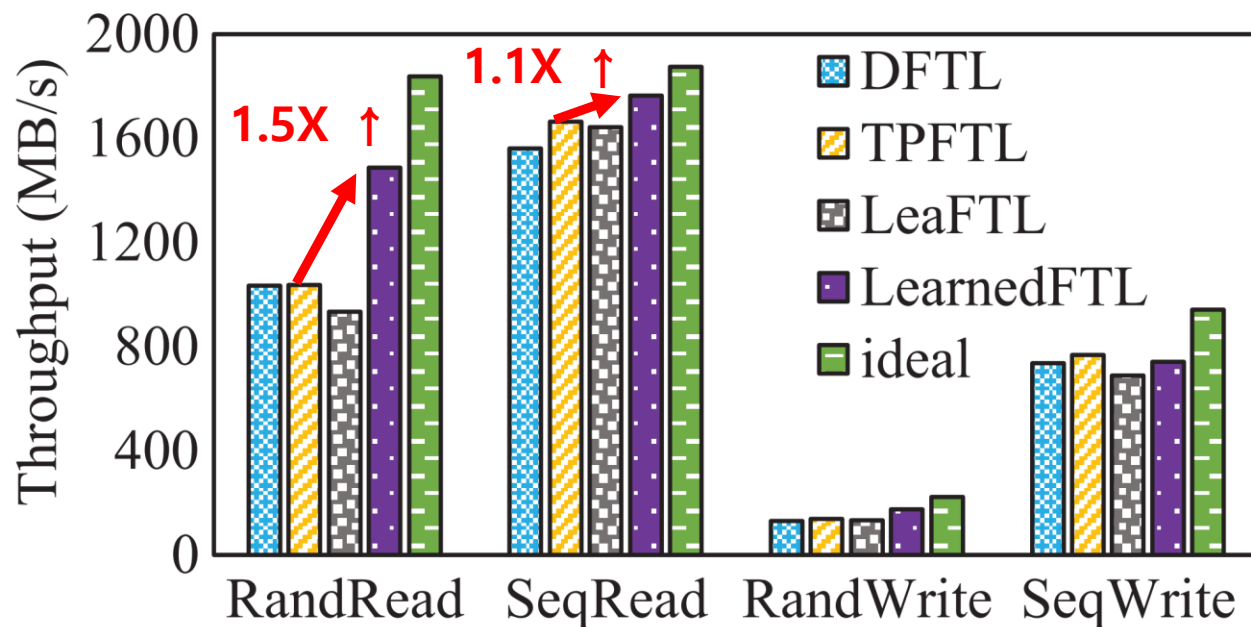


(a) Throughput

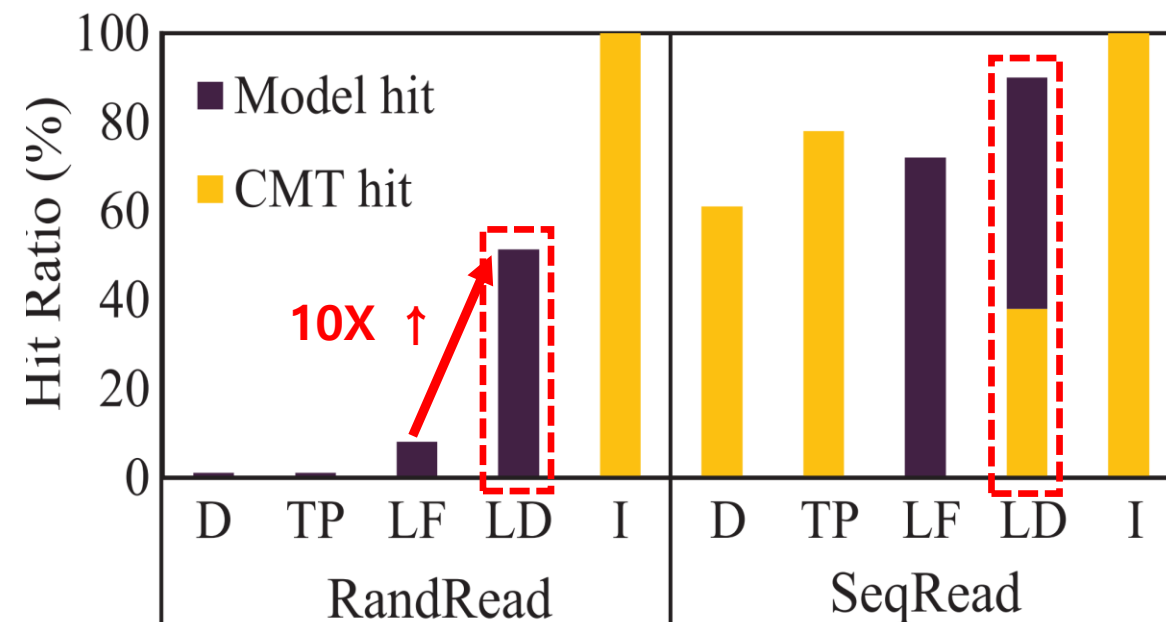
4. Evaluation

Read(FIO)

- Read 전 sequential/random write으로 워밍업
- 4KB I/O size, psync I/O engine을 사용해 64개의 threads에서 읽기 실험 진행



(a) Throughput

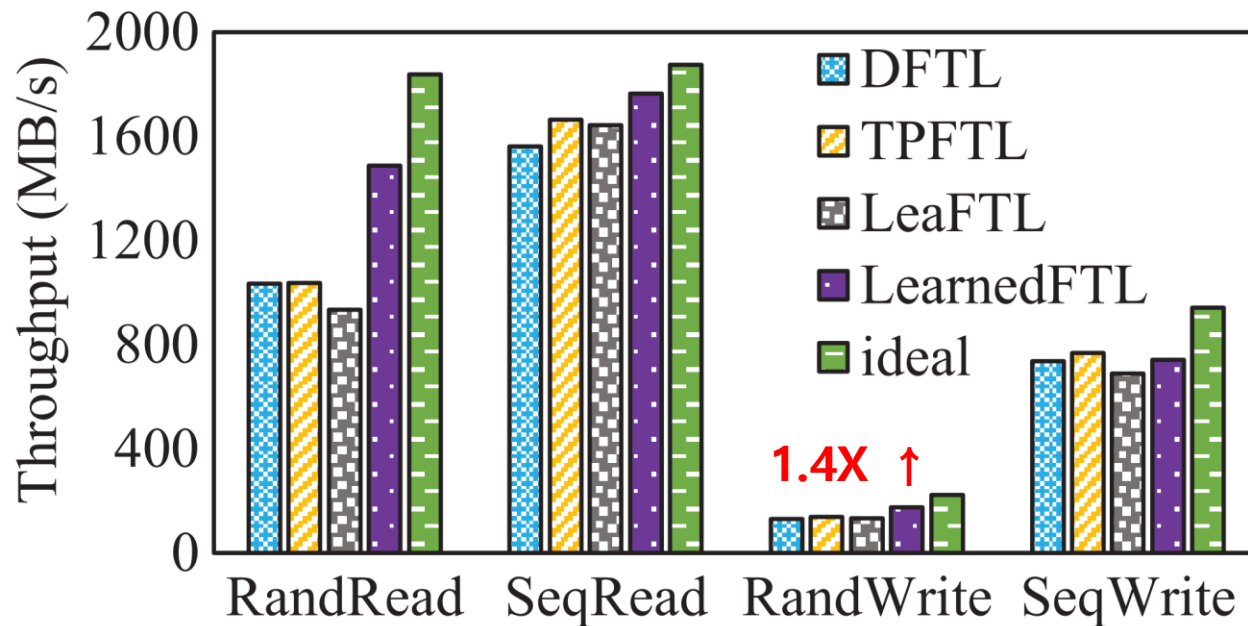


(b) Model and CMT hit ratio

4. Evaluation

▪ Write(FIO)

- 4KB I/O size, psync I/O engine을 사용해 쓰기 실험 진행
- Group-based allocation 방식을 사용해 다른 FTL보다 뛰어난 성능

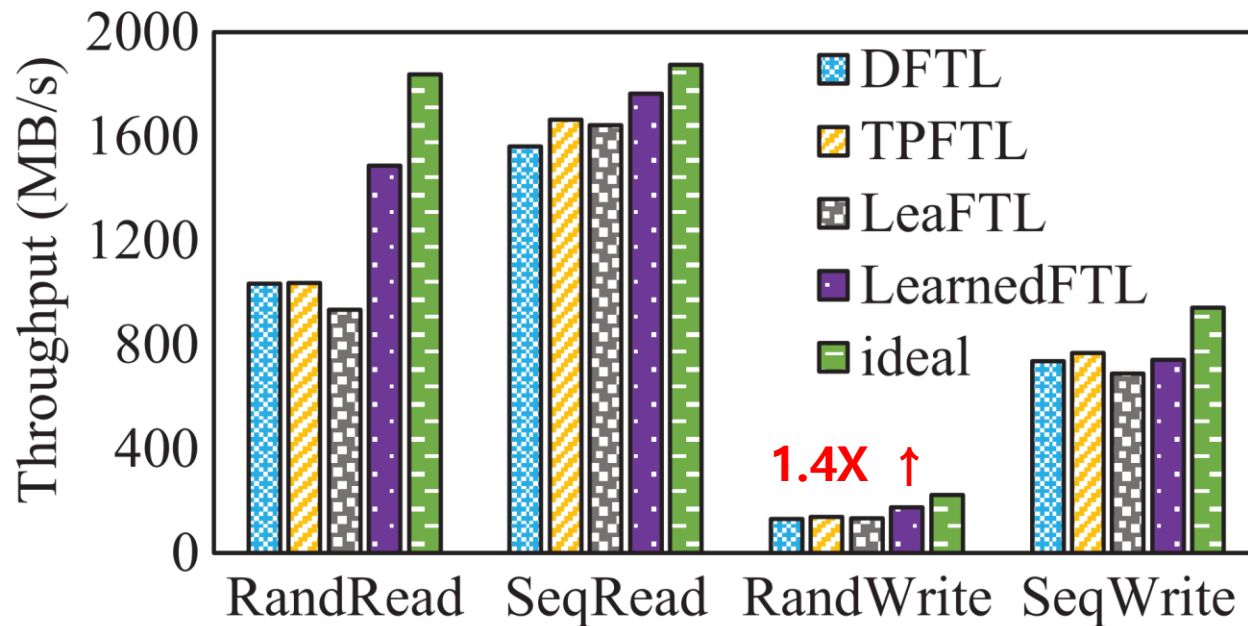


(a) Throughput

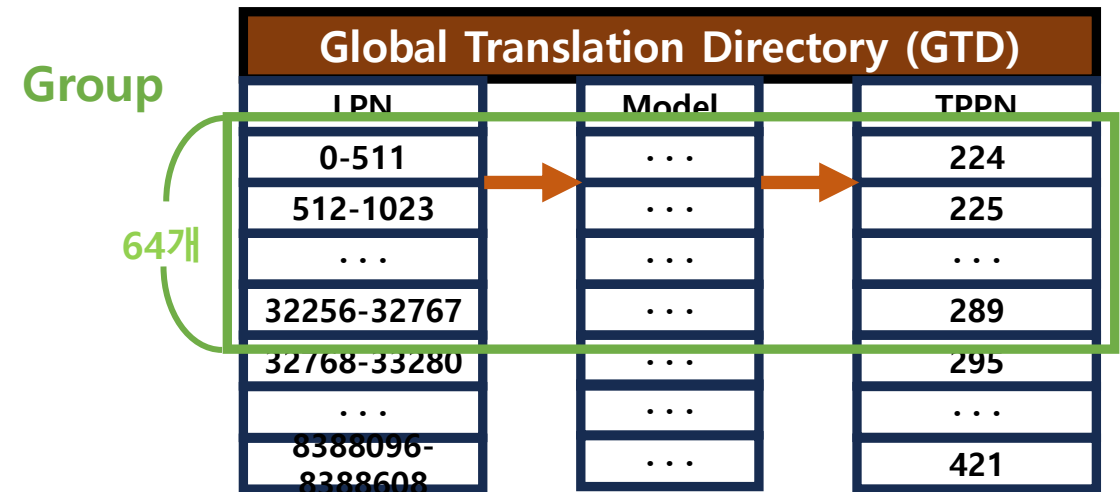
4. Evaluation

Write(FIO)

- 4KB I/O size, psync I/O engine을 사용해 쓰기 실험 진행
- Group-based allocation 방식을 사용해 다른 FTL보다 뛰어난 성능



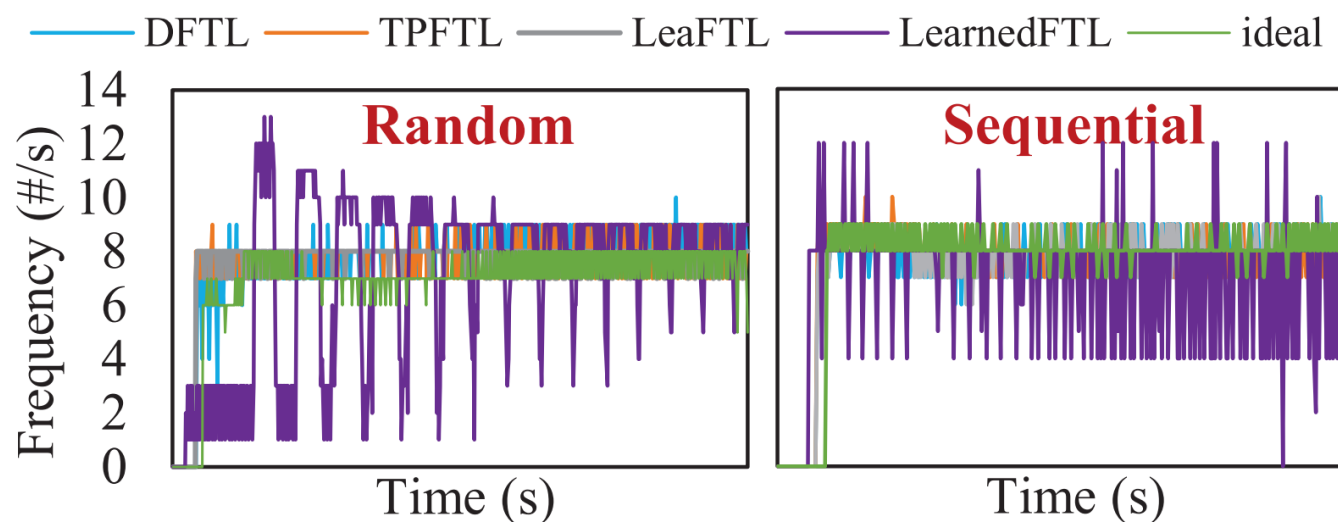
(a) Throughput



4. Evaluation

■ GC overhead

- 모델 훈련은 GC에서 이루어지며 group-based allocation 방식으로 트레이닝



Write	Random	Sequential
DFTL	4335	4572
TPFTL	4335	4304
LeaFTL	4395	4473
LearnedFTL	4188	4285

Fig. 16: The GC frequency of all FTL designs under FIO random and sequential write benchmarks.

4. Evaluation

■ GC overhead

- 모델 훈련은 GC에서 이루어지며 group-based allocation 방식으로 트레이닝

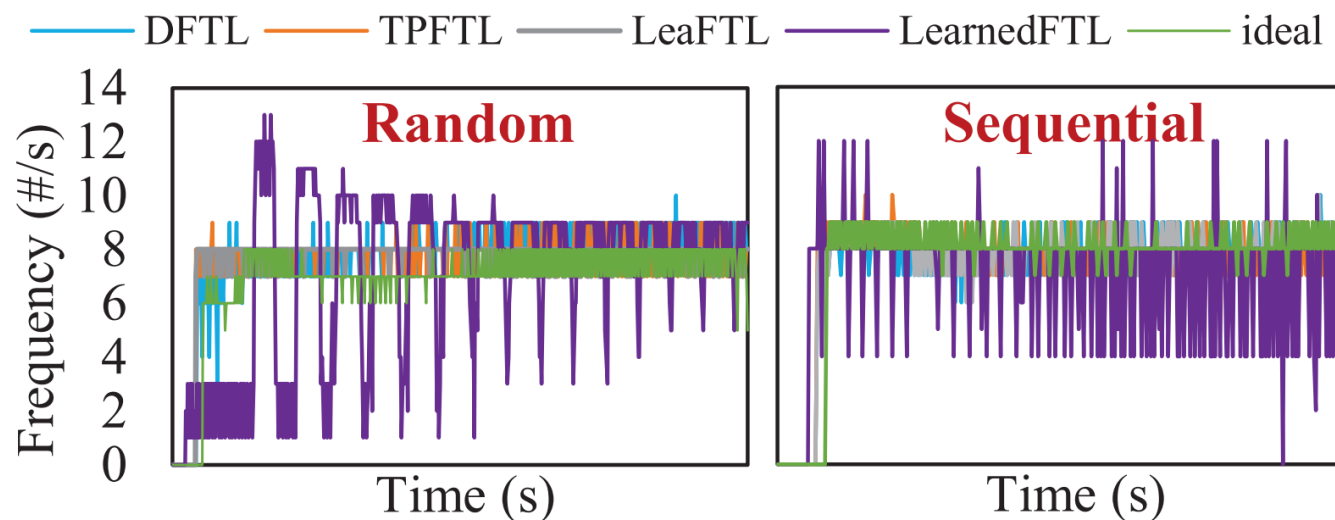
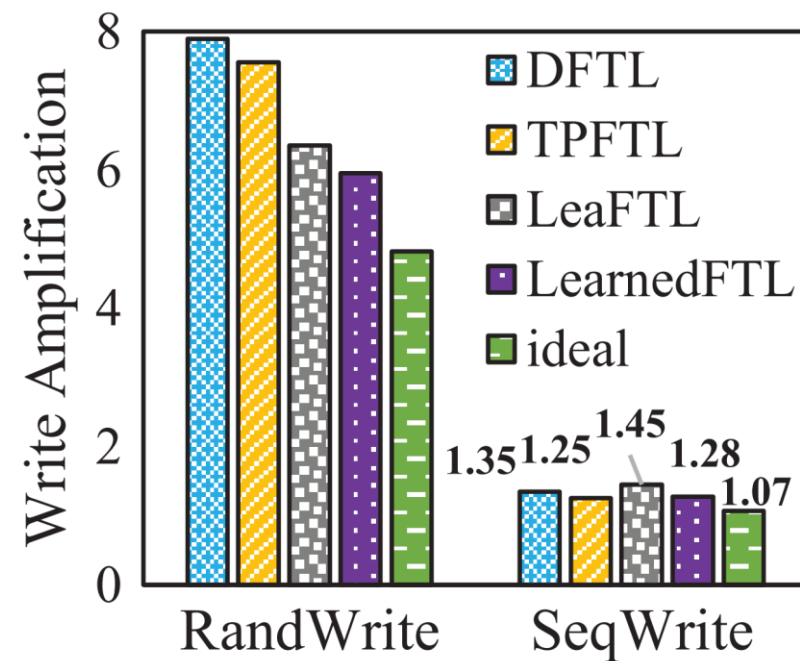


Fig. 16: The GC frequency of all FTL designs under FIO random and sequential write benchmarks.



(c) Write amplification

4. Evaluation

■ Training and sorting overhead

- Model training & LPN-sorting
- GC 동안 GTD entry group에 64개의 LPN을 정렬하고 트레이닝 작업이 트리거 된다.

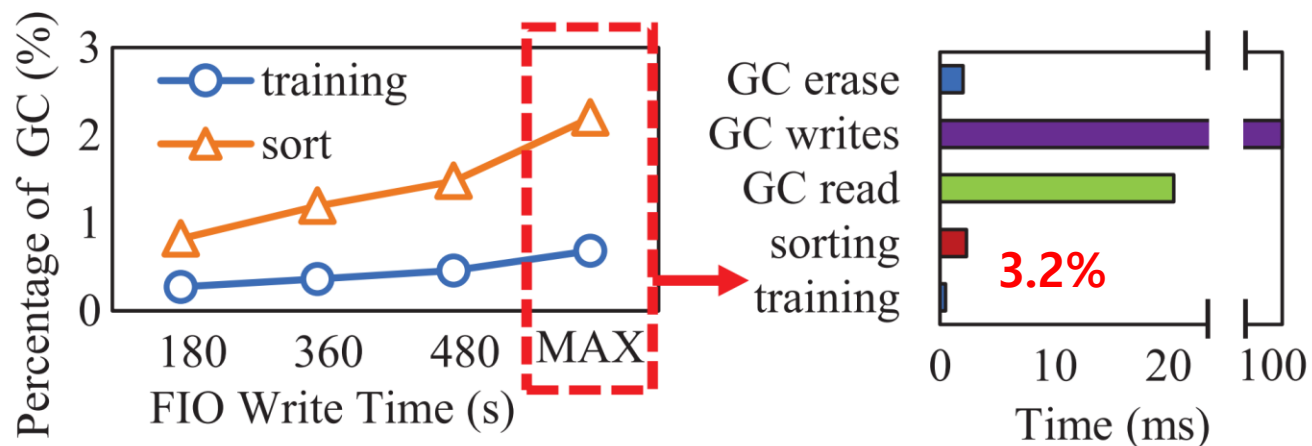


Fig. 17: The time overhead of sorting and training under different running times of FIO random writes (MAX means almost all pages are valid during GC).

4. Evaluation

■ Training and sorting overhead

- Model training & LPN-sorting
- GC 동안 GTD entry group에 64개의 LPN을 정렬하고 트레이닝 작업이 트리거 된다.

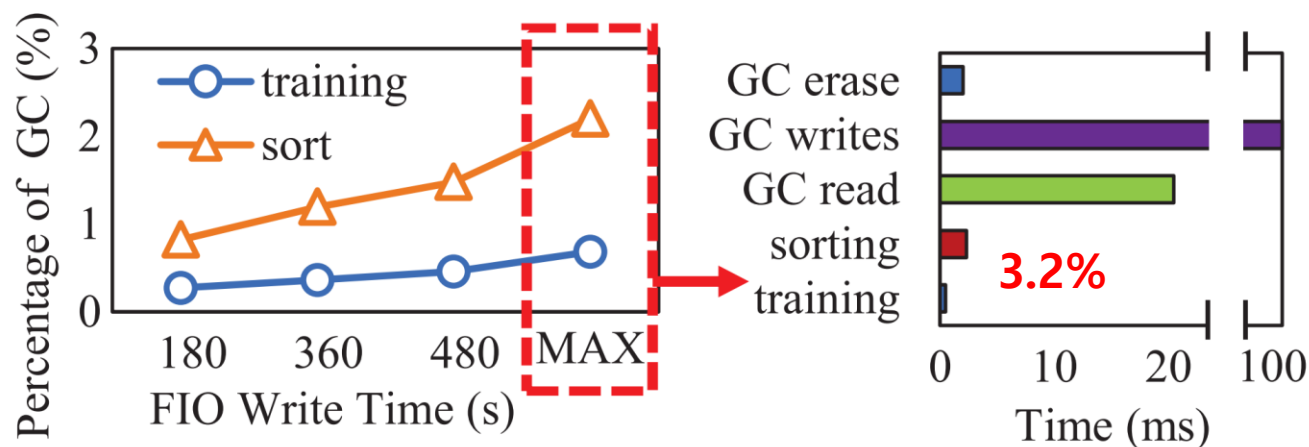
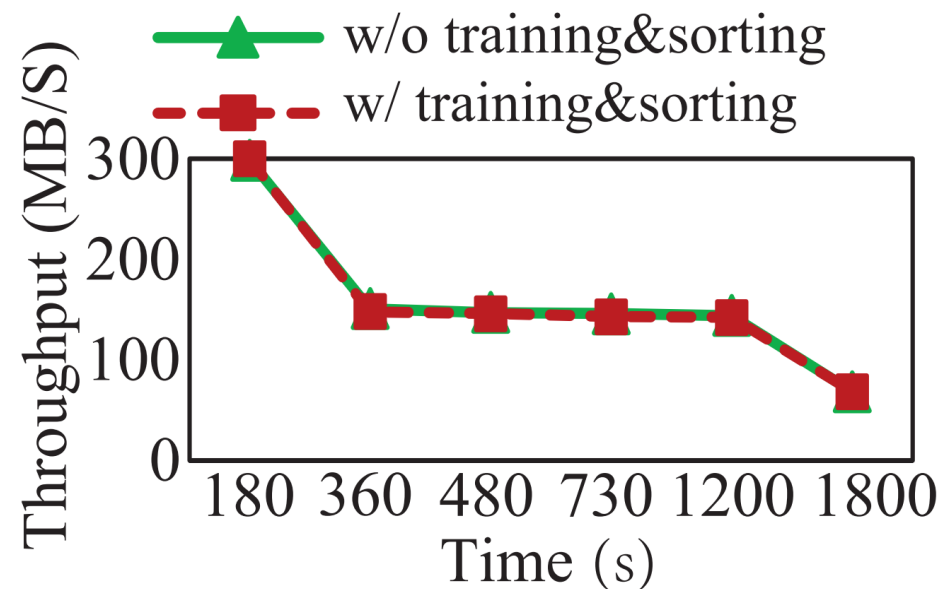


Fig. 17: The time overhead of sorting and training under different running times of FIO random writes (MAX means almost all pages are valid during GC).

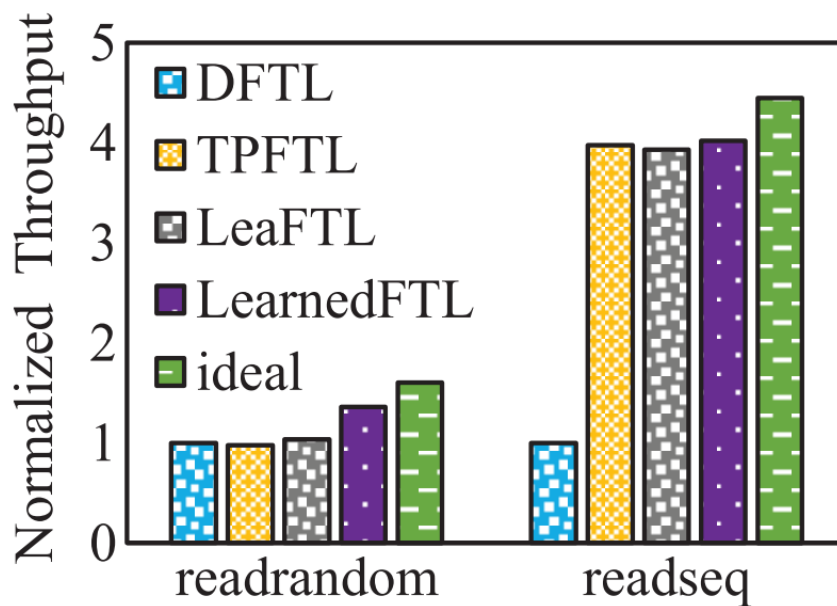


(a) Write and GC

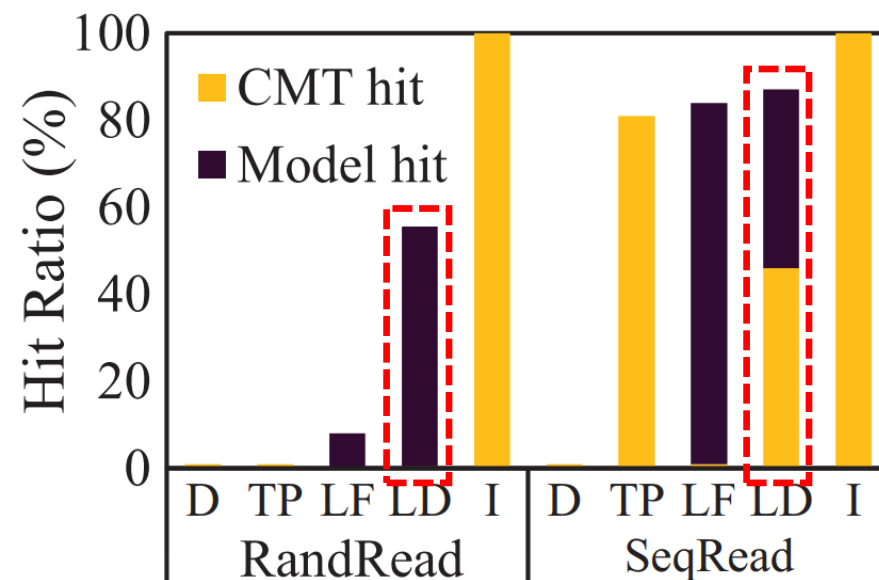
4. Evaluation

Real-World Applications

- **RocksDB** evaluation with EXT4 filesystem
- db_bench tool 사용(단일 스레드)



(a) Normalized throughput



(b) CMT and model hit ratio

5. Conclusion

- ① Address translation 효율을 높이기 위해 learned index 사용
 - Random read access에서 miss 시 발생하는 double read 감소
 - In-place update linear model로 learned index를 효율적으로 사용
- ② 효율적인 model 학습을 위해 Virtual PPN을 사용
 - 채널부터 페이지까지 고정된 수의 SSD 계층의 PPN 계산 순서를 변경
 - 기존 연속적인 LPN에 불연속적인 PPN 대신 연속적인 PPN으로 구현 가능
- ③ Group-based allocation과 GC/rewrite를 통한 model training을 사용해 training overhead 감소

Thank you

LearnedFTL: A Learning-based Page-level FTL for Reducing Double Reads in Flash-based SSDs

Shengzhe Wang, Zihang Lin, Suzhen Wu, Hong Jian, Jie Zhang, Bo Mao

Computer Science and Engineering Department, University of Texas at Arlington, Arlington, TX, USA *School of Computer Science, Peking University

2024 IEEE International Symposium on High-Performance Computer Architecture (HCPA)

2024. 08. 07

Q&A

Presentation by Kim MinSeong, Wee DaYeon
kms0509@dankook.ac.kr, wida10@dankook.ac.kr