華中科技大學
HUAZHONG UNIVERSITY OF SCIENCE AND TECHNOLOGY

武汉光电国家研究中心
WUHAN NATIONAL LABORATORY FOR OPTOELECTRONICS

HBTree: an Efficient Index Structure Based on Hybrid DRAM-NVM

Yuanhui Zhou[1], Taotao Sheng[1], Jiguang Wan*[1,2]

1. WNLO, Huazhong University of Science and Technology, Wuhan,Hubei, China
2. Shenzhen Huazhong University of Science and Technology Research Institute

NVMSA 2021

目录
CONTENTS

01

Background

Background – Persistent Memory

- Emerging Npn-Volatile Memories(NVMs)
  - PCM, STT-MRAM, RRAM

- Characteristics of NVM
  - low latency
  - non-volatility
  - low power consumption
  - high storage capacity
  - byte addressing

- Optane DC PMM
  - first 3D X-Point persistent memory (PM) product

Background – Indexing structures

- Conventional indexing structures not suitable NVM

- Optimizing indexing structures for NVM based on conventional indexes
  - Optimize a single index structure to accommodate full NVM memory
    - Path Hash，Leveling Hash，NV-Tree ……
  - reduce the consistency overhead on NVM and speeds up failure recovery
    - FAST&FAIR，wB+Tree，CDDS-Tree……
  - study hybrid structure and use DRAM to optimize system performance
    - FPTree，HiKV，LB+Tree……

02

Motivation

motivation

- B+Tree more suit NVM
  - Simple structure
  - Excellent Scan performance
  - Random Read and Write
- DRAM-NVM
- Challenges
  - Data consistency
  - Insert and Split overhead
  - Recovery time

Operational Efficiency

- Rich KV operations
  - — Load/Put/Get/Delete

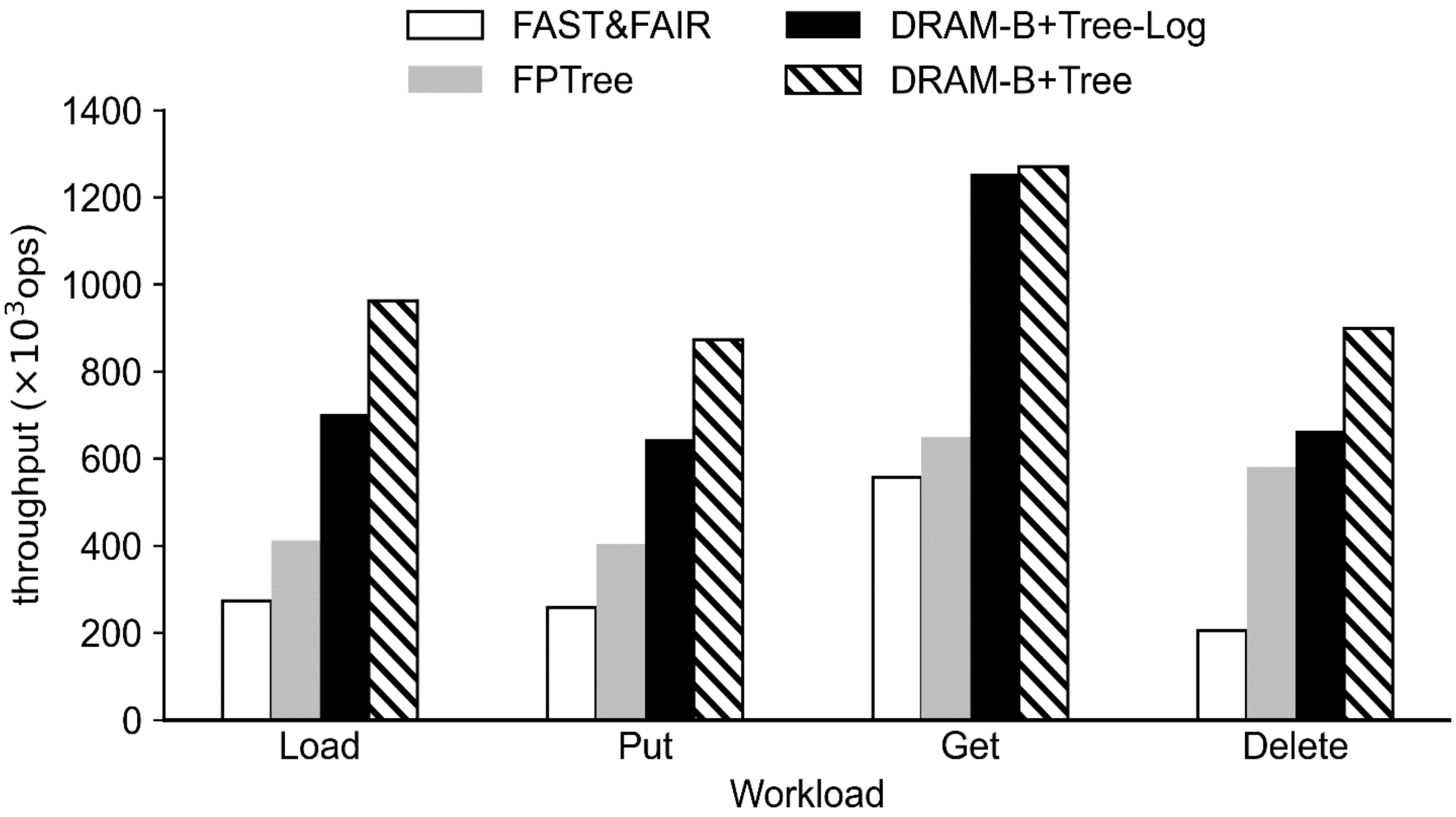
- observe

  DRAM-B+Tree (No persistence)

  > DRAM-B+Tree-log (log big)

  > FPTree

  > FAST&FAIR



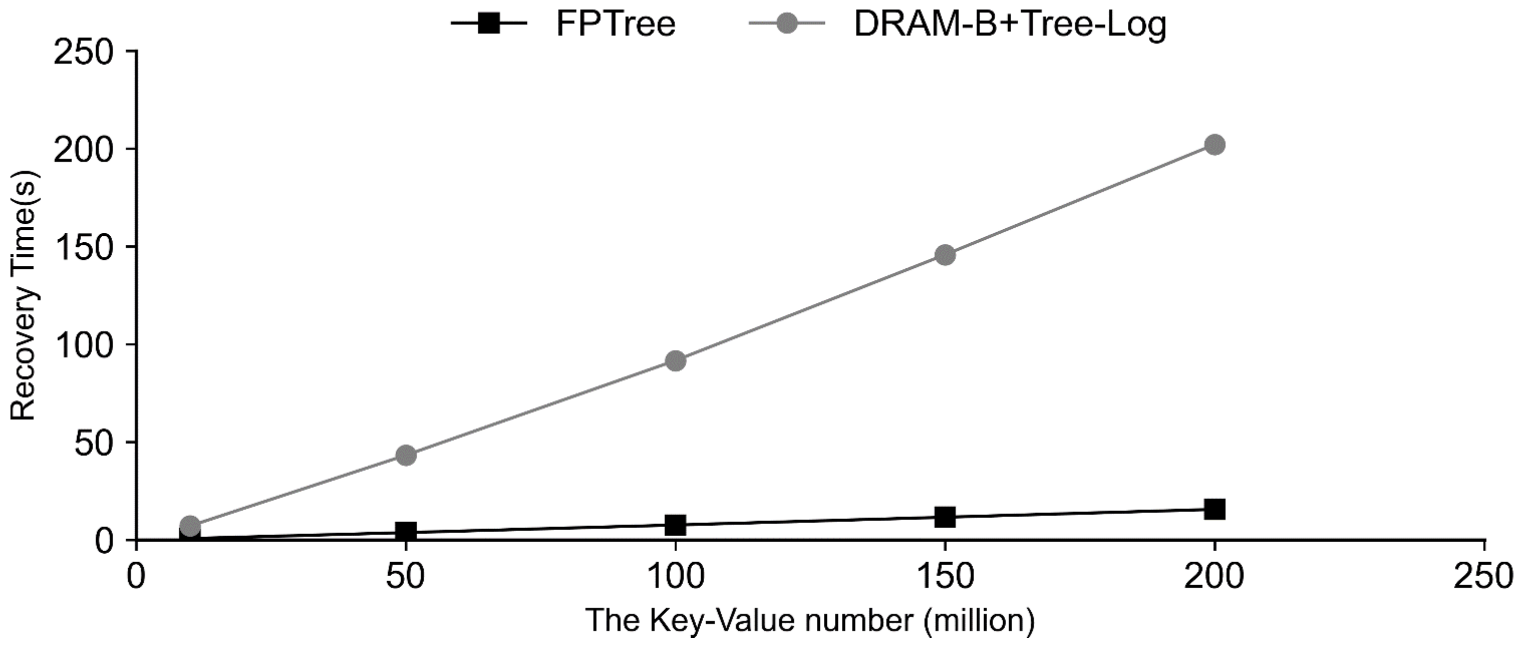
How to take advantage of the DRAM-NVM hybrid structure

Recovery Time

How to reduce the recovery time?

How to reduce the log?

Make some data into NVM



Recover time for FPTree and DRAM-B+Tree-Log

03

Design

Design ideas

- B+ tree index structure with DRAM-NVM hybrid Memory
- Cache hot data in DRAM to improve performance
- ensure data consistency
- speed up the system failure recovery

Overall Architecture of HBTree

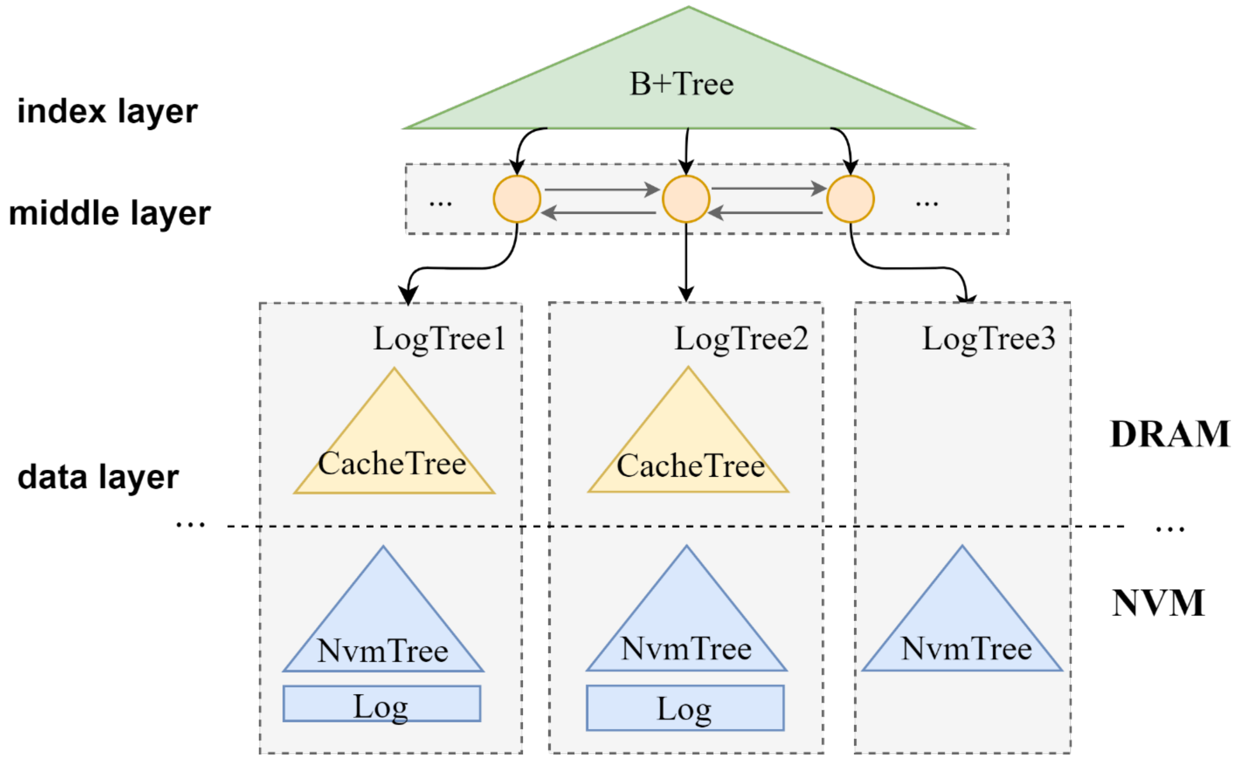
➢ HBTree: a hybrid three-layer persistent index

Index layer

- a B+tree on DRAM
- Does not persistent

Middle layer

Data layer



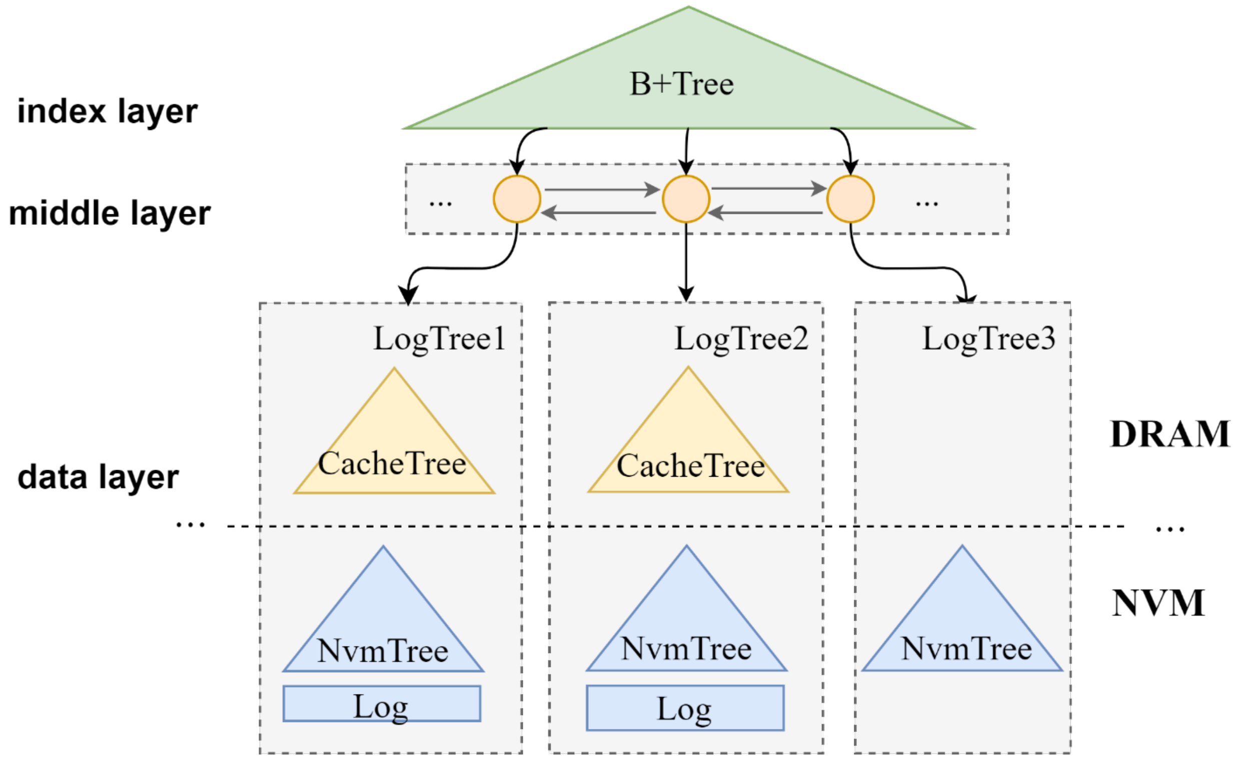
Hotspot Statistica Algorithm

➢ HBTree: a hybrid three-layer persistent index

- Index layer
- **Middle layer**
  - Stores LogTree metadata
  - Identify hot NVMTree

    $$T_{(t+\Delta t)} = A * T_t + Operate_{\Delta t}$$

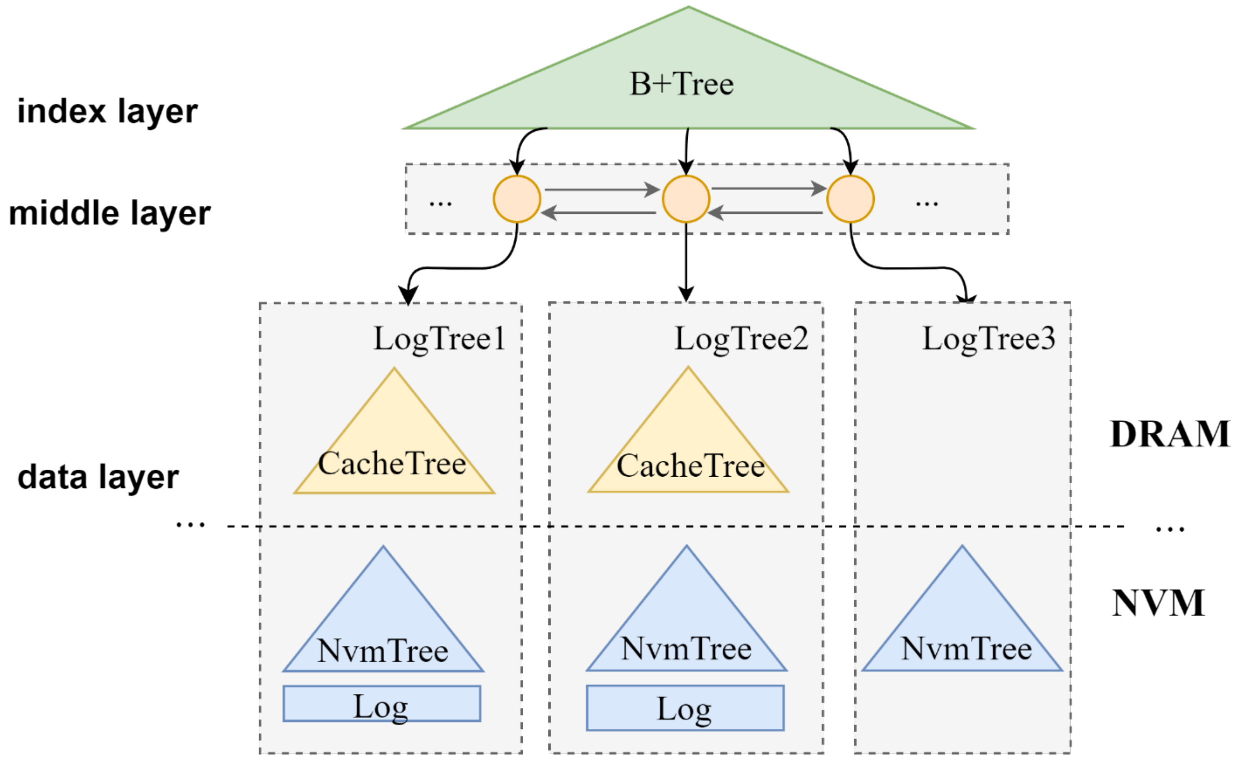
    A default 0.5
  - backup to NVM
- Data layer



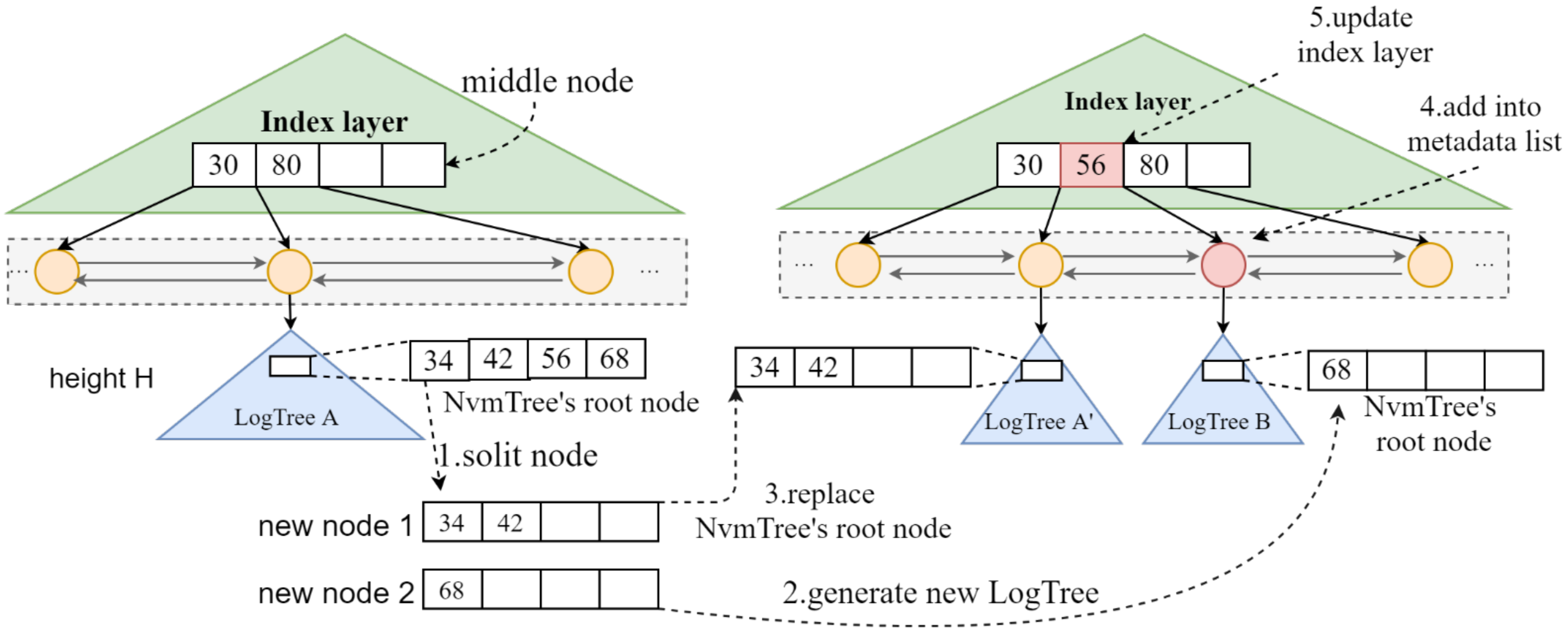
LogTrees

➢ HBTree: a hybrid three-layer persistent index

- Index layer
- Middle layer
- **Data layer**
  - Data persistence
  - Fast recovery
  - Highspeed access
  - Consistency



Dynamic Extension



CacheTree Management

- CacheTree Create
  - Copy NVMTree to DRAM to generate CacheTree
    - Write operation is recorded in a log
    - The read request first looks for the log and then the NVMTree
  - Log be played back to CacheTree
- CacheTree Syschronization
  - Only data marked dirty is synchronized
  - Old log is replaced by new log
- CacheTree Recycle
  - Read paused and update dirty data to NVMTree
  - Release CacheTree nodes and log record to replay
  - NVMTree work and delete log

Consistency

- Copy-on-write
  - NVMTree: over 8B write
  - CacheTree Syschronization and CacheTree Recycle

- The log is exploited to ensure consistency

Recovery

- Middle layer Recovery
  - traversing the persistent metadata node linked list in NVM
  - The NVMTree in the split continues
- LogTree Recovery
  - Playback logs ensure the integrity of the NVMTree
  - Create CacheTree based on middle layer hot data information
  - Playback logs to recovery CacheTree
- The index layer be recreated directly through the middle layer

- NOTE:
  - CacheTree recovery within different LogTrees can be performed concurrently

04

Evaluation

Evaluation methodology

- Platform:
  - CPU: two 24-core Intel Xeon Gold 5218R CPUs(2.3GHz)
  - DRAM: DDR4 64GB
  - OS： linux (kernel version 5.10.1)
  - NVM: Intel Optane DC Persistent Memory 128G * 2
- Workloads
  - YCSB： 8B key, 8B value

    Load: 200 million, others: 10 million
- Compared systems：
  - FPTree
  - FAST&FIRE

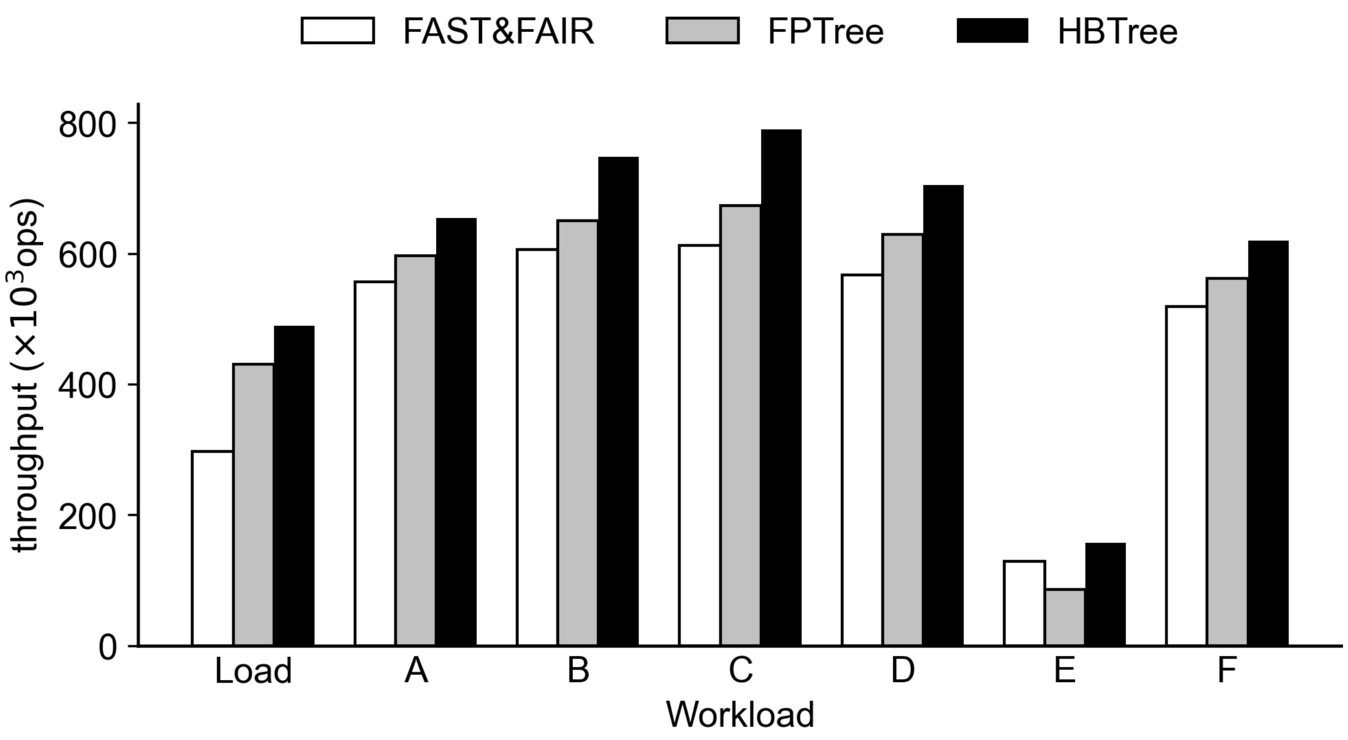
Worklaods	Requestdistribution	Op
Load	Uniform	100% Put
A	Zipfian	50%Get，50%Update
B	Zipfian	95%Get，5%Update
C	Zipfian	100%Get
D	Latest	95%Get，5%Put
E	Zipfain/Uniform	95%Scan，5%Put
F	Zipfian	50%Put，50%RMW

Operation Efficiency

B+Tree node is 512B

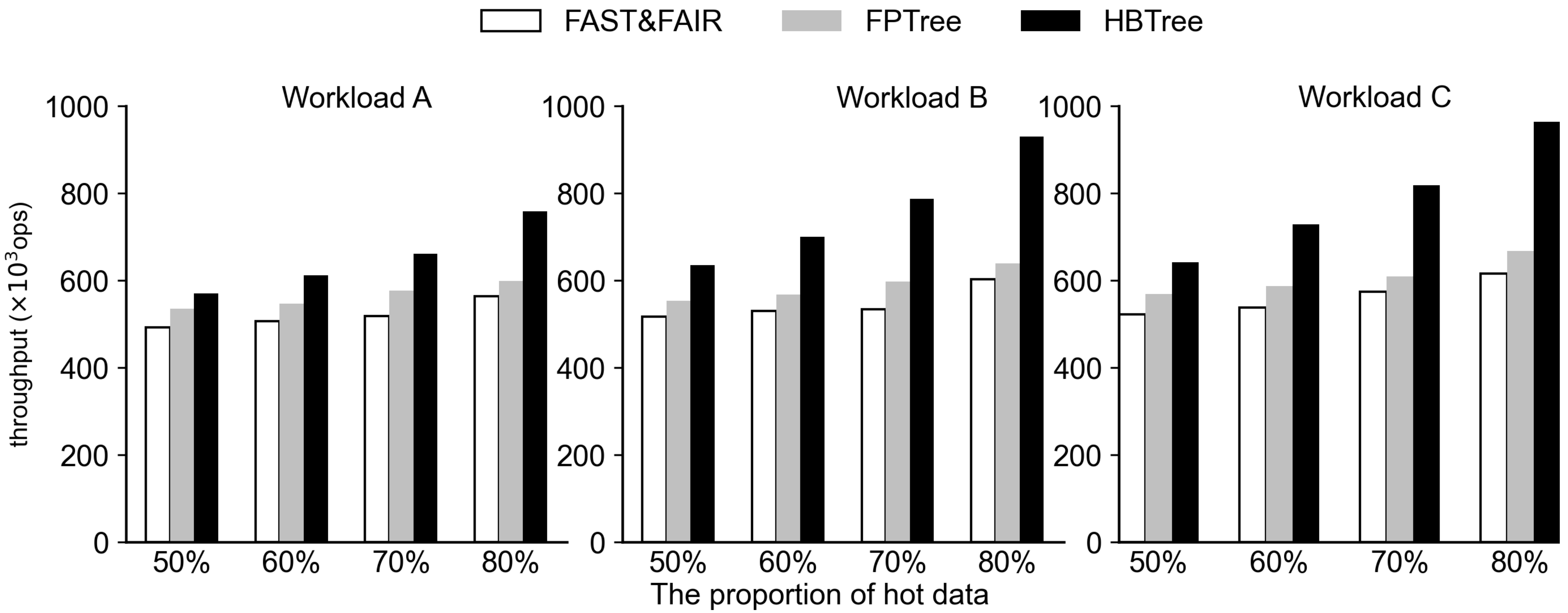
DRAM cache size is 500MB

- Load
  - Better than FAST&FAIR
- write more (A、F)
  - improvement small
- Read more (B、C、D or E)
  - Better



Throughput on the YCSB workloads

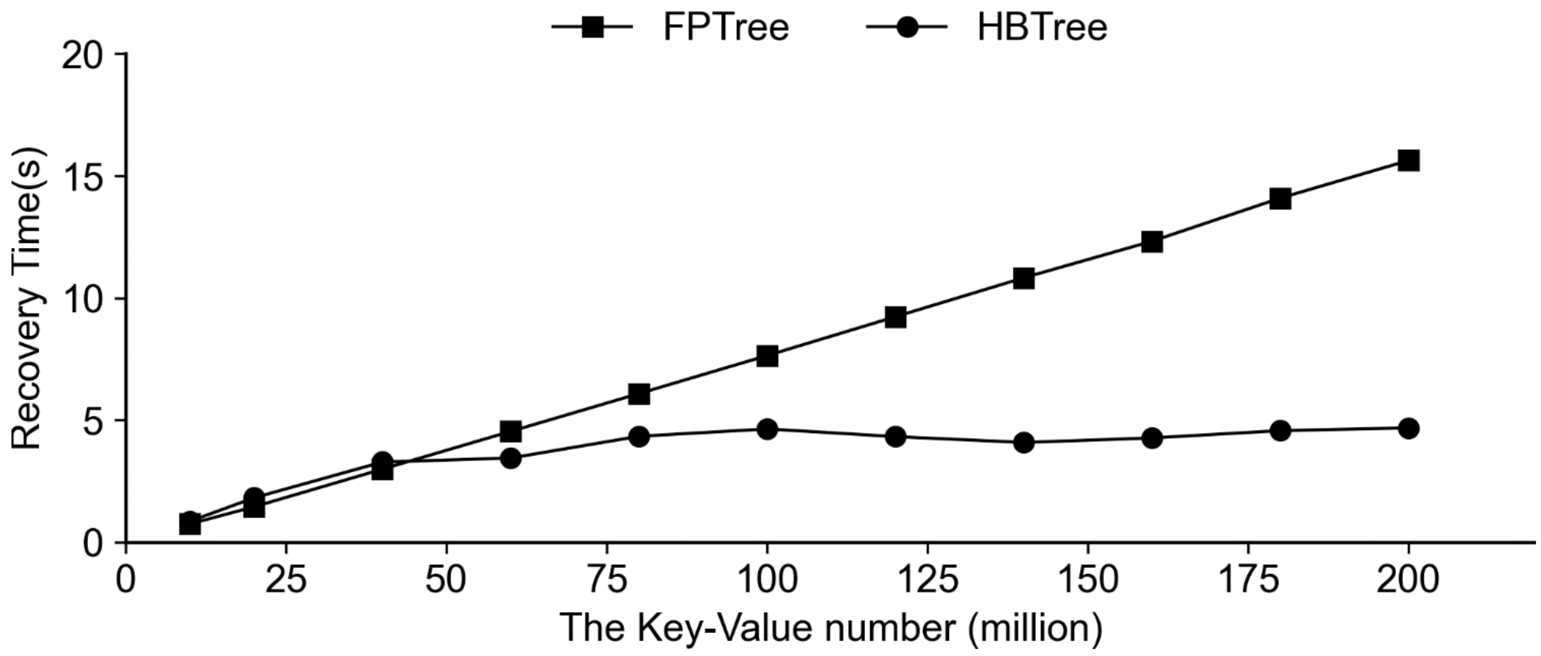
Performance with Hotness Data



Throughput under different data hotspots

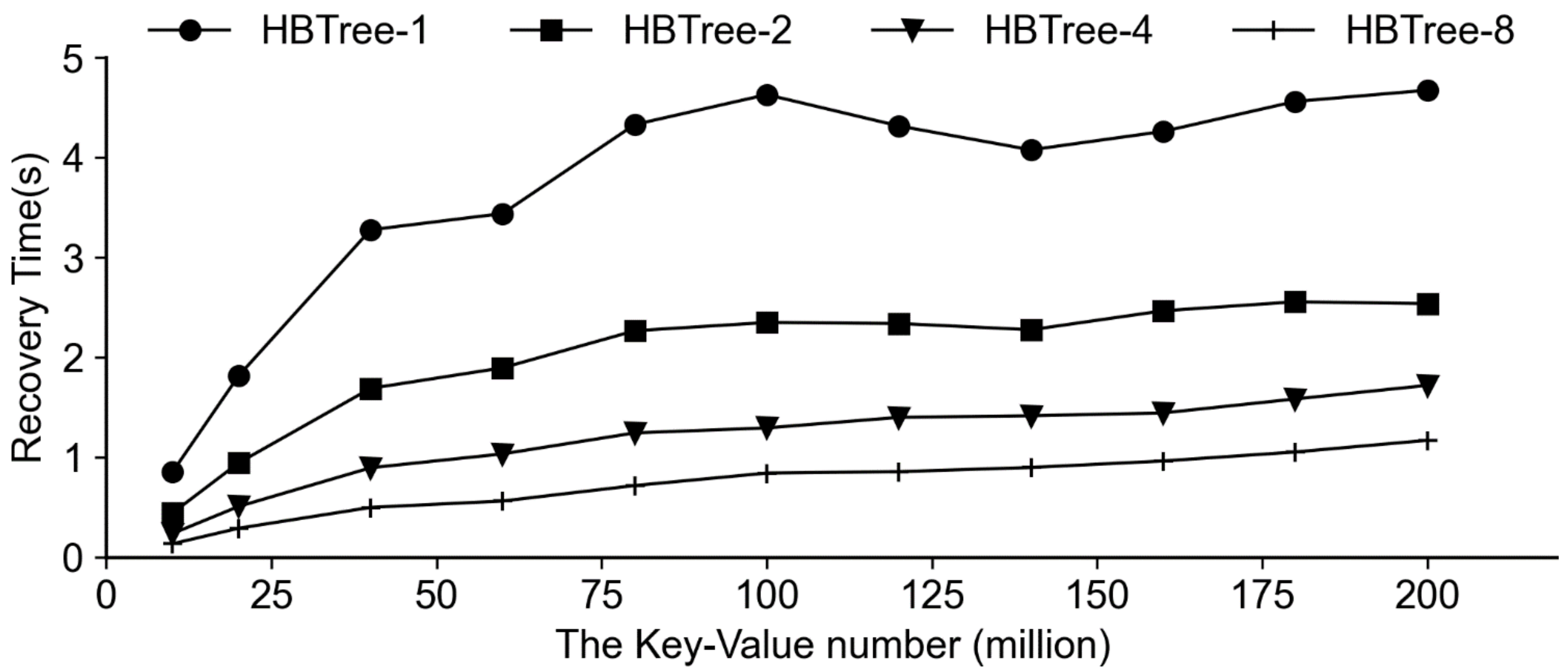
Recovry Time

- When the data volume is small
  - HBTree is closer to FPTree
- With the increasing data
  - HBTree remains level
  - FPTree is still increasing
- 30%



Recovery time for HBTree and FPTree in various data volumes

Recovry Time



Recovery time for different threads of HBTree

華中科技大學
HUAZHONG UNIVERSITY OF SCIENCE AND TECHNOLOGY

武汉光电国家研究中心
WUHAN NATIONAL LABORATORY FOR OPTOELECTRONICS

Thanks listening

Yuanhui Zhou zhouyuanhui@hust.edu.cn

Taotao Sheng

Jiguang Wan* jgwan@hust.edu.cn