A Tale of Two Paths: Toward a Hybrid Data Plane for Efficient Far-Memory Applications

Lei Chen, Shi Liu (co-first author), Chenxi Wang*,
Haoran Ma, Yifan Qiao, Zhe Wang, Chenggang Wu,
Youyou Lu, Xiaobing Feng, Huimin Cui, Shan Lu,
Guoqing Harry Xu*

Microsoft

Existing Far-Memory Data Paths

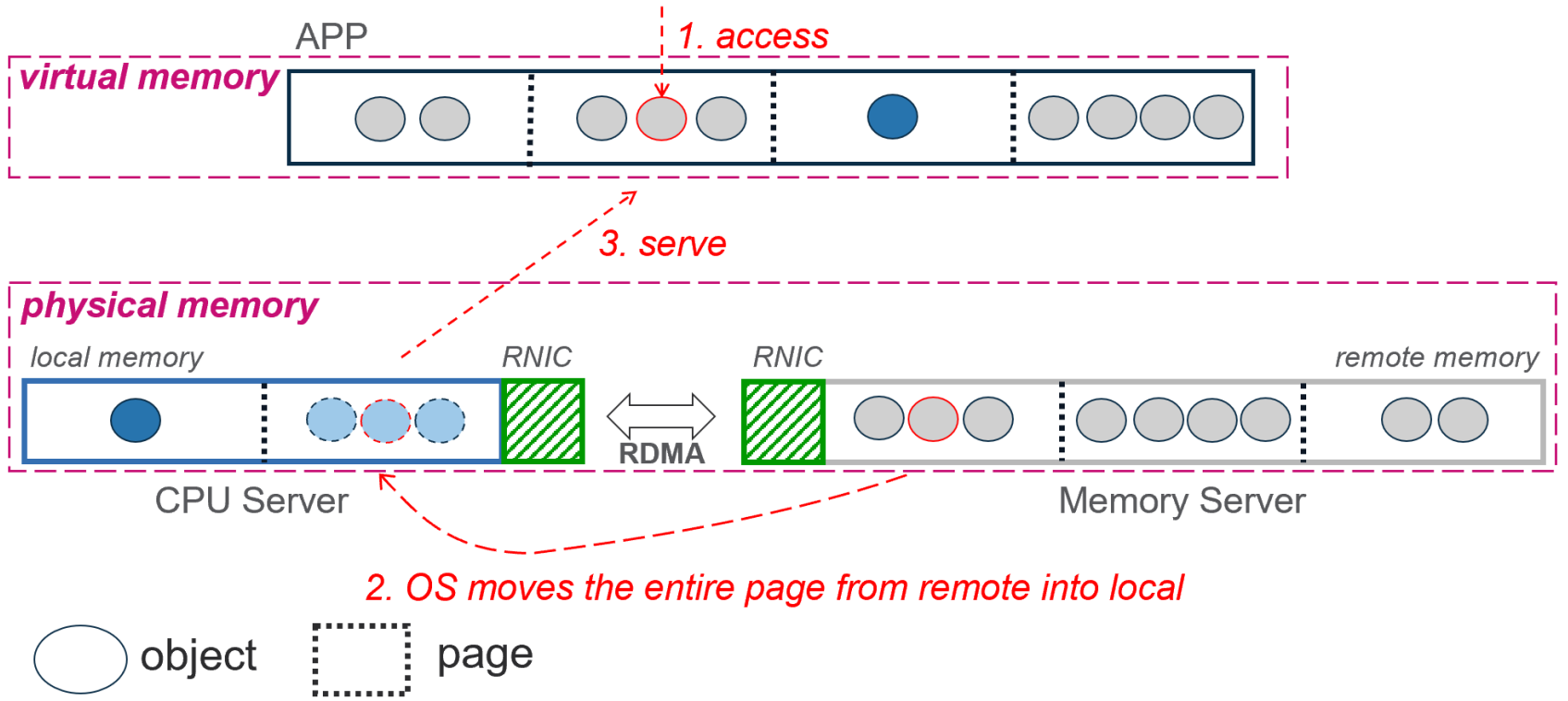
OS-Paging Data Path

Pros:

- page-granularity
- transparent

Cons:

- I/O amplification



Existing Far-Memory Data Paths

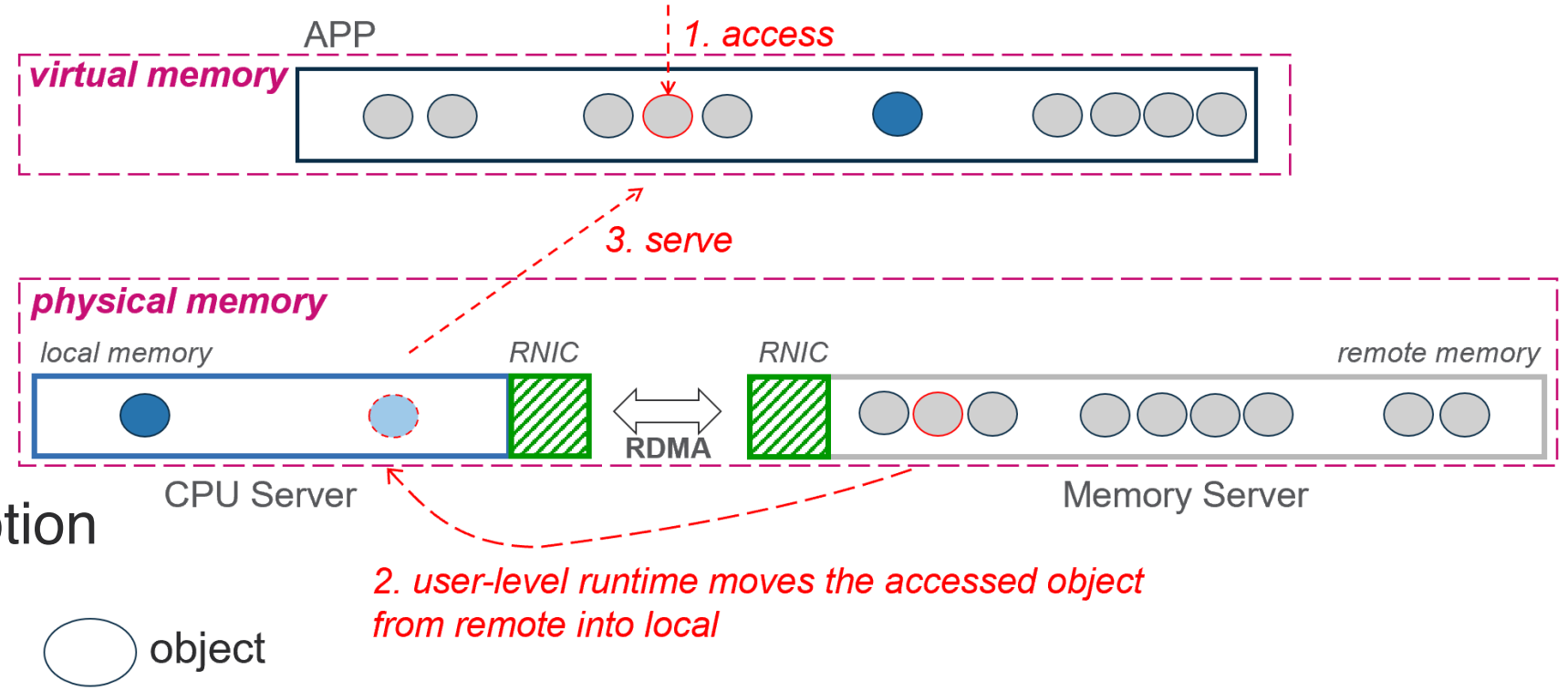
Runtime-Object Data Path

Pros:

- object-granularity
- no I/O amplification

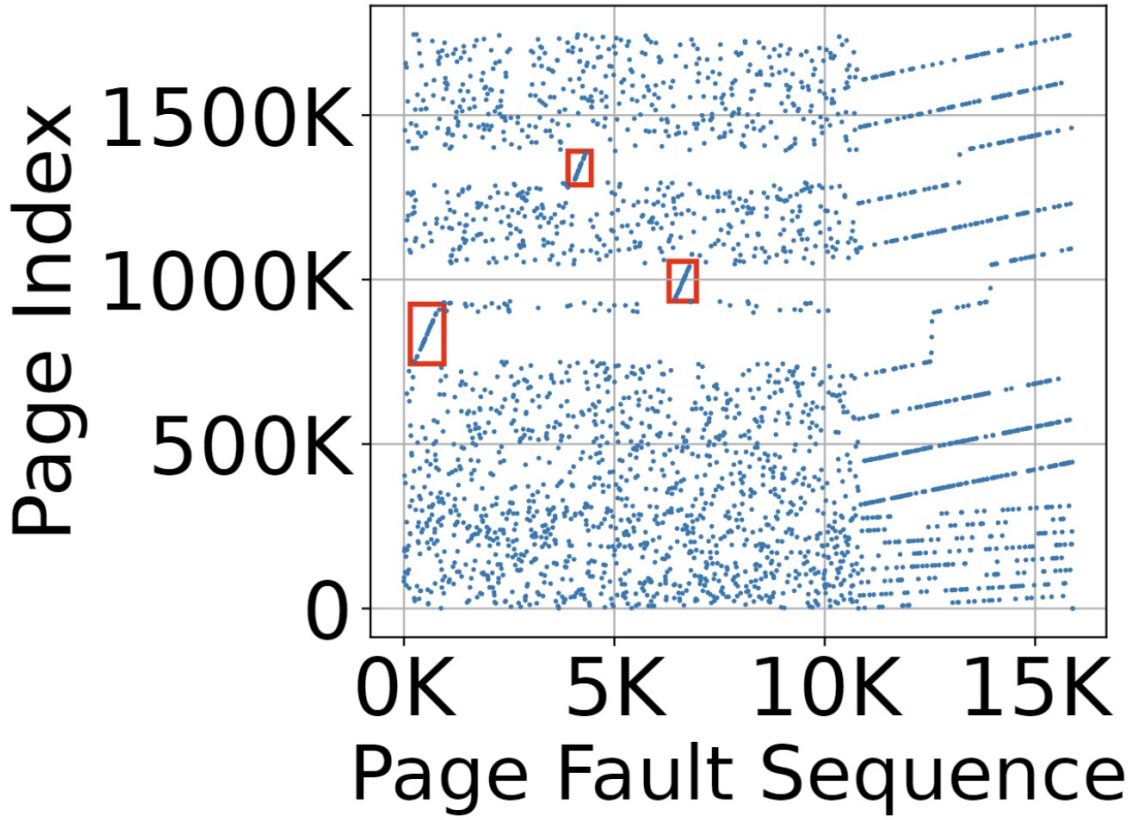
Cons:

- not transparent
- high CPU consumption



Hybrid Data Plane

Motivation: Existing data paths have their own Pros and Cons, suitable for different access patterns

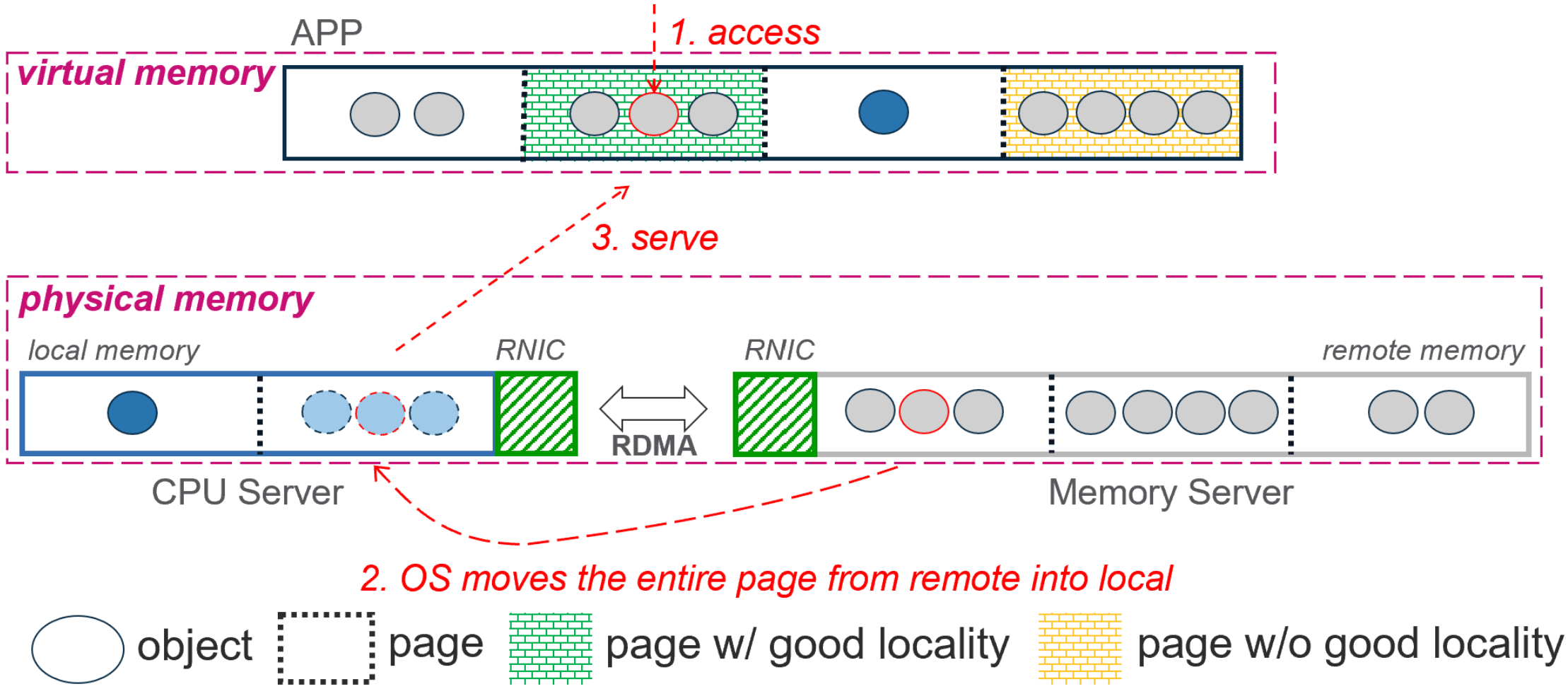


Hybrid Data Plane

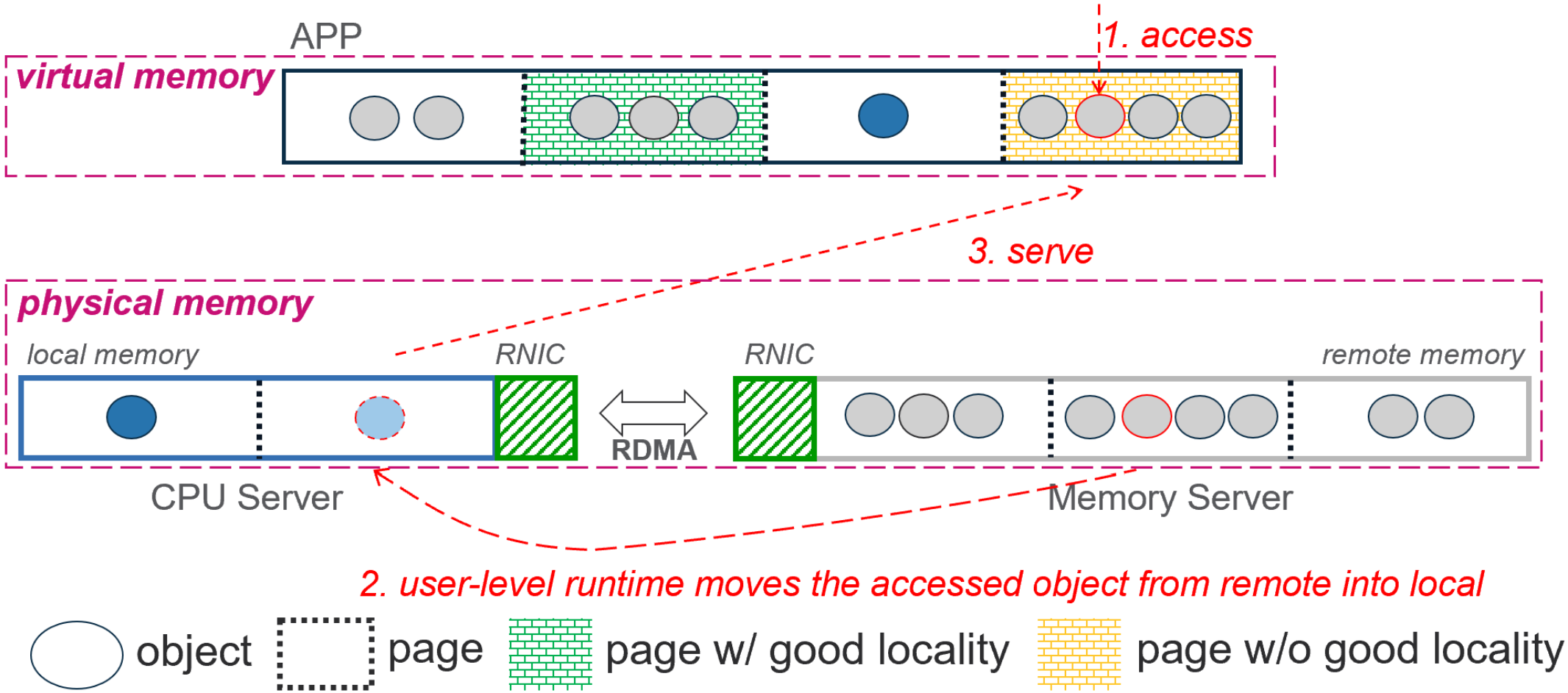
Motivation: Existing data paths have their own Pros and Cons, suitable for different access patterns

Insight: Hybrid data plane that simultaneously enables accesses via these two data paths to provide high efficiency for real-world applications

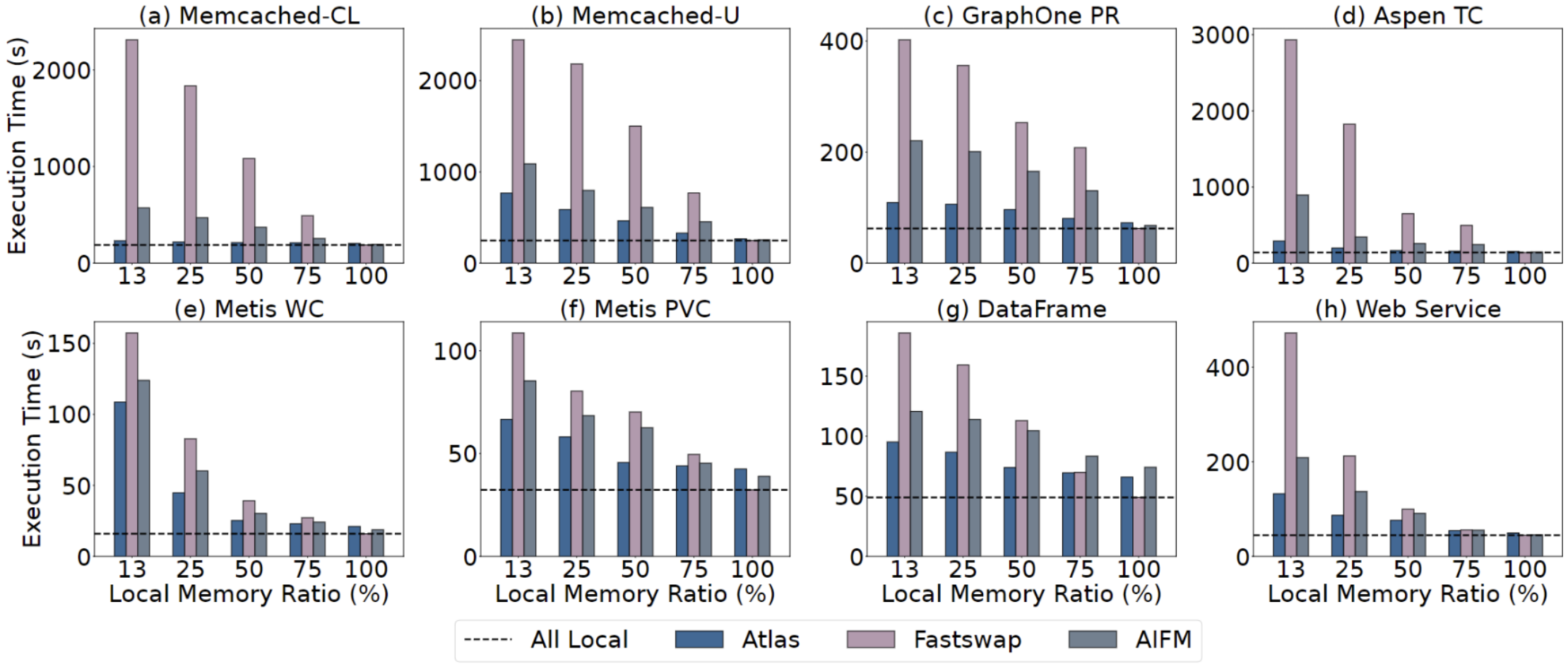
Hybrid Data Plane



Hybrid Data Plane



Hybrid Data Plane



Compared with AIFM and Fastswap, our hybrid data plane Atlas improves the throughput by **1.5×** and **3.2×**

University of Chinese Academy of Sciences

Flagger: Cooperative Acceleration fo r Large-Scale Cross-Silo Federated Learning Aggregation

Xiurui Pan, Yuda An, Shengwen Liang, Bo Mao,
Mingzhe Zhang, Qiao Li, Myoungsoo Jung, Jie Zhang

CHASELab
Computer Hardware And System Evolution Laboratory

PEKING UNIVERSITY

CHINESE ACADEMY OF SCIENCES

UNIVERSITAS AMOIENSIS

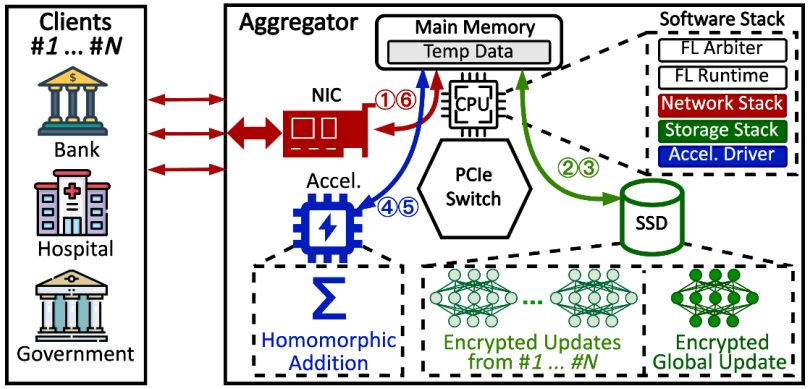
KAIST

大规模跨机构联邦学习的近数据协同加速架构

问题挑战：

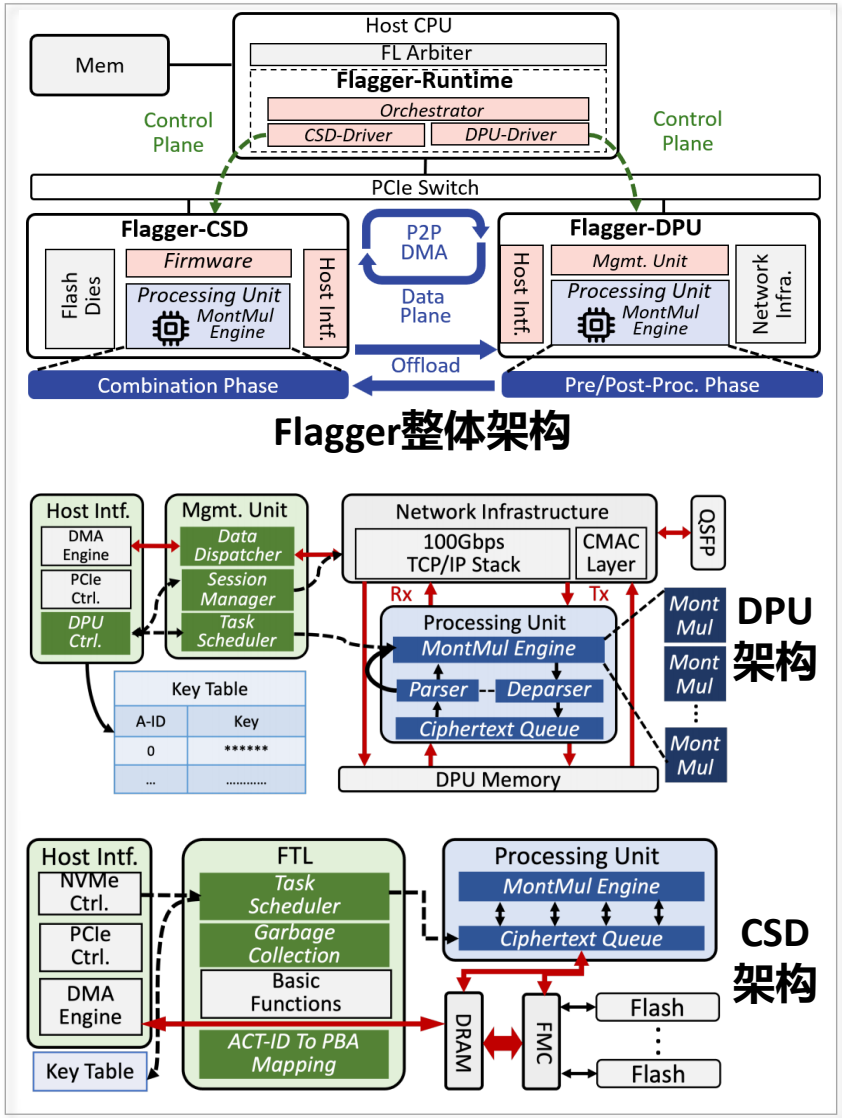
大规模跨机构联邦学习聚合端面临**性能瓶颈**

- 低速广域网导致冗长的**网络传输开销**
- 复杂的**同态加密计算开销**
- 同态加密导致膨胀的**密文存储开销**
- 架构冗余导致的**数据搬移开销**



初步分析：

- **网络传输**：传输密文导致的端到端时延高达29.6%
- **计算单元**：计算资源闲置率高达64%
- **密文存储**：膨胀的密文需要SSD存储，导致SSD的低速访存成为性能瓶颈
- **数据通路**：数据与计算分离的架构导致数据在不同设备间反复搬移，造成性能损失



解决方案：

DPU – CSD 协同设计加速联邦学习聚合

- **Flagger-Runtime**: 控制平面运行时，负责协调DPU和CSD共同加速聚合任务
- **Flagger-DPU**: 在DPU内集成在网计算的同态加密算子和完整的网络栈，将预处理/后处理的同态计算与广域网传输重叠
- **Flagger-CSD**: 在SSD内集成近存储计算的同态加密算子，直接在近存储端完成同态聚合，降低数据搬移开销

实验结果：

- **整体性能**
  - 对比传统架构聚合器，端到端性能提升6.57倍，聚合性能提升3.84倍
- **计算单元利用率**
  - 对比传统架构聚合器，资源利用率提升46.7%
- 含同态计算的**数据传输性能**
  - 最大网络带宽达到41.12Gbps
  - 最大访存带宽达到4.4GB/s

Thanks!

Xiurui Pan, Yuda An, Shengwen Liang, Bo Mao,
Mingzhe Zhang, Qiao Li, Myoungsoo Jung, Jie Zhang

CHASELab
Computer Hardware And System Evolution Laboratory

PEKING UNIVERSITY

KAIST

FUYAO: DPU-enabled Direct Data Transfer for Serverless Computing

Guowei Liu 🏠 Laiping Zhao 🏠 📱 Yiming Li 🏠 Zhaolin Duan 🏠

Sheng Chen 🏠 Yitao Hu 🏠 Zhiyuan Su 🏛 Wenyu Qu 🏠

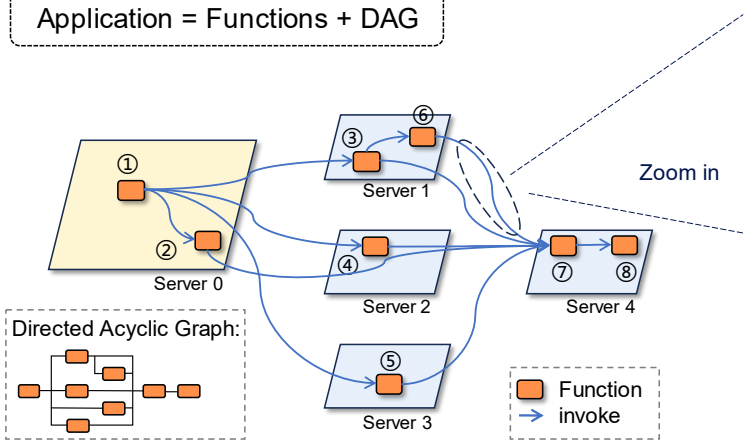
🏠 : Tianjin Key Lab. of Advanced Networking, **Tianjin University**
🏛 : Inspur Electronic Information Industry Co., Ltd.
📱 : Corresponding author

29th ACM International Conference on Architectural Support for Programming Languages and Operating Systems

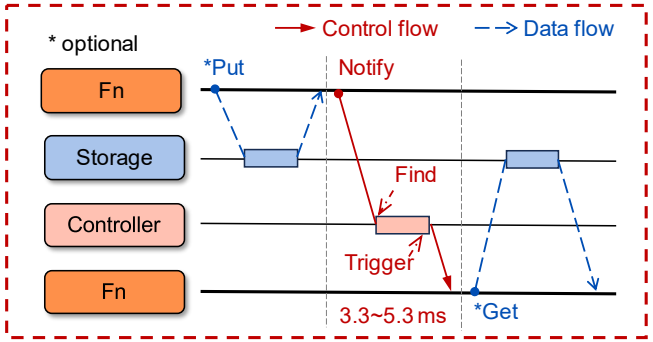
1 Introduction

Example: Deploying an application on the current serverless system

Application = Functions + DAG



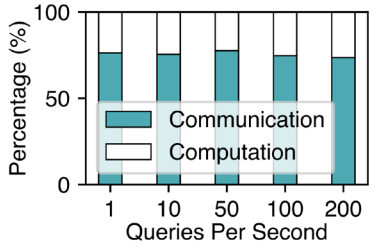
Third-party Forwarding:



"Couples control flow and data flow"

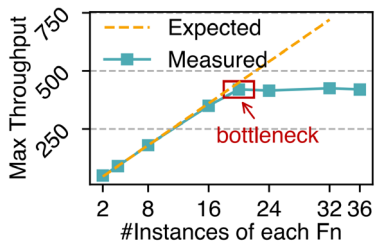
2 Motivation

Problems caused by Third-party Forwarding:



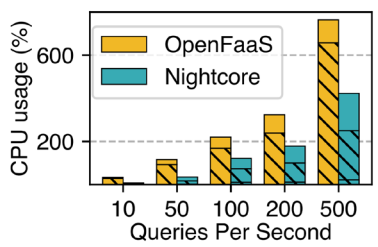
❌ High Comm. Overhead

- Comm. Ratio > 70%
- Increased Overall Latency



❌ Auto-scaling Failures

- Controller becomes a bottleneck
- Throughput not increase with function instance numbers



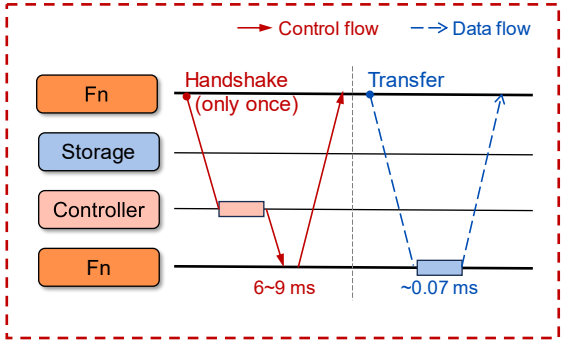
❌ High Control Overhead

- Total CPU core usage > 7.5
- Control overhead accounts for 80%

3 Key Ideas & Challenges

Key Ideas: Decouple control flow from data flow

Direct Transfer

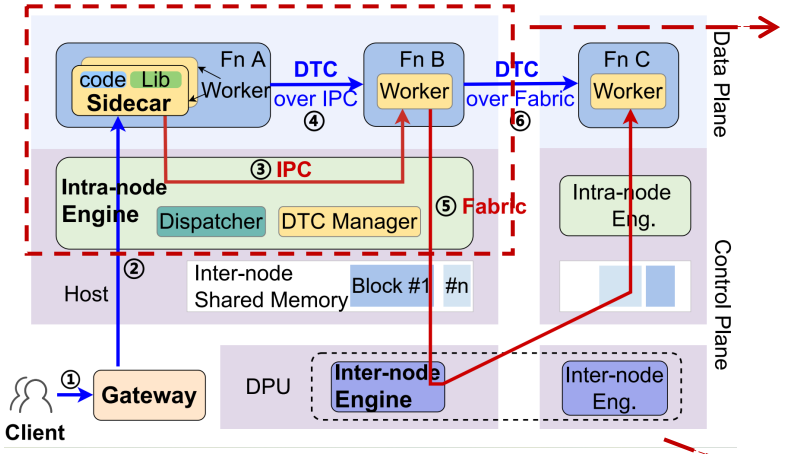


However, it's hard to do this, due to:

- Highly Dynamic
- Location-oblivious
- Establishing overhead

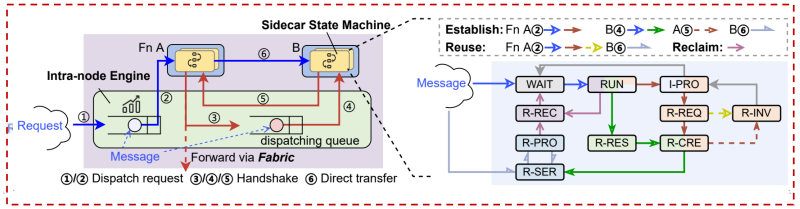
4 Design Overview

The Architecture overview of FUYAO

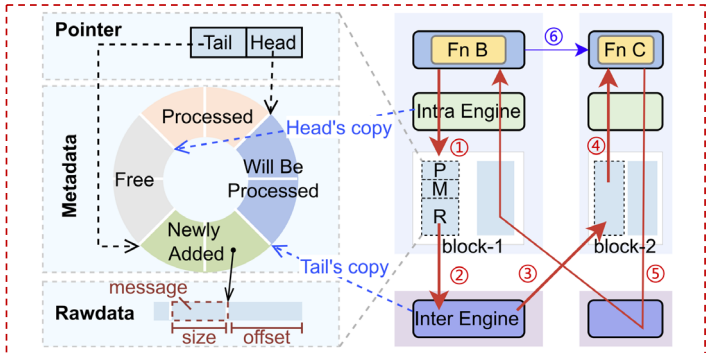


- Provides four data transfer method: IPC\...
- Suitable for functions within or across nodes

Intra-node direct transfer design:

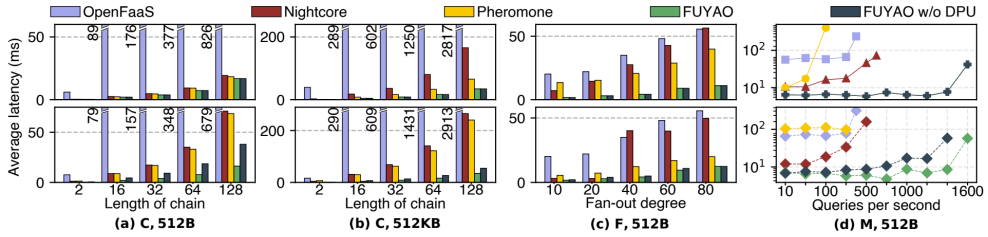


Inter-node direct transfer design:

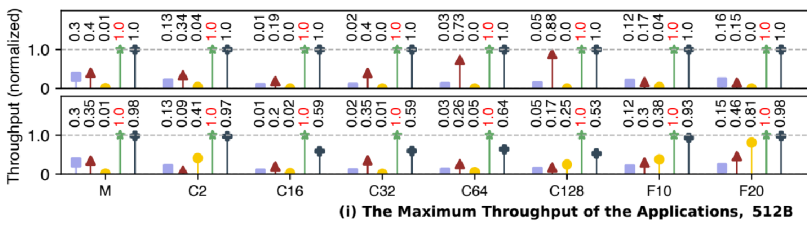


5 Evaluation

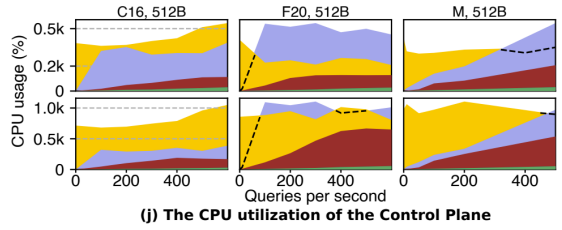
Compared with OpenFaaS, Nightcore[ASPLOS'21], Pheromone[NSDI'23]



Low latency



High throughput



Low control overhead

ACM ChinaSys

Rethinking an eBPF-based lightweight & unified solution for Edge networking

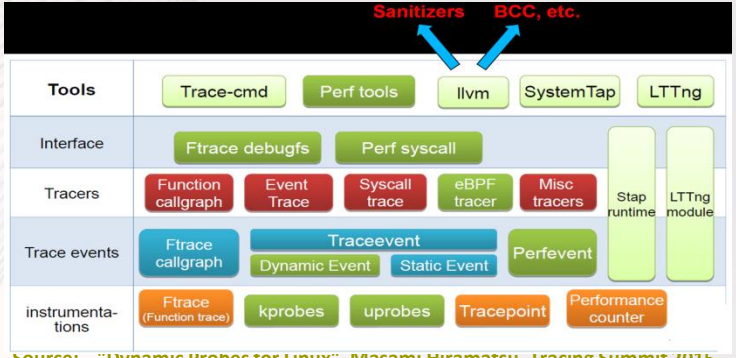
Feng Li (李枫)

Jun 16, 2024

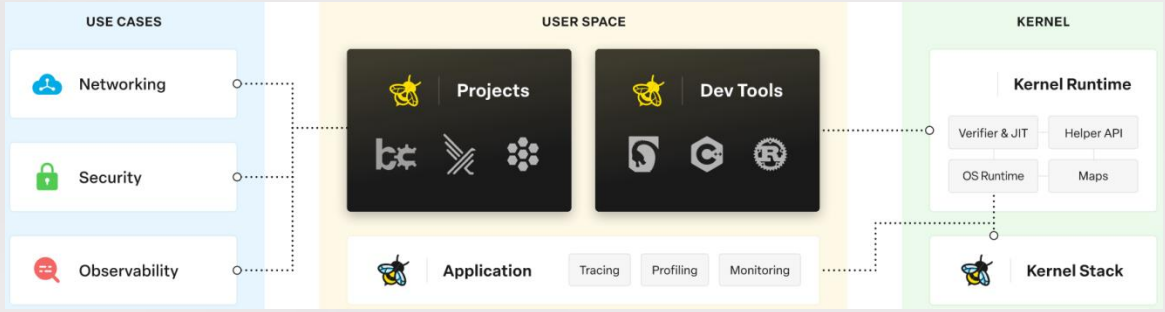
An indie developer from China：

- The main translator of the book «Gray Hat Hacking The Ethical Hacker's Handbook, Fourth Edition» (ISBN：9787302428671) & «Linux Hardening in Hostile Networks, First Edition» (ISBN: 9787115544384)
- Pure software development for ~15 years (~11 years on Mobile dev)
- Actively participating Open Source Communities:
  https://github.com/XianBeiTuoBaFeng2015/MySlides/Conf
  https://github.com/XianBeiTuoBaFeng2015/MySlides/LTS
- Recently, focus on infrastructure of Cloud/Edge Computing, AI, IoT, Programming Languages & Runtimes, Network, Virtualization, RISC-V, EDA, 5G/6G

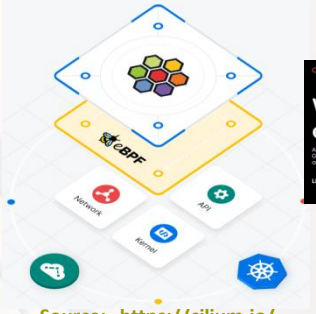
hkli2013@126.com



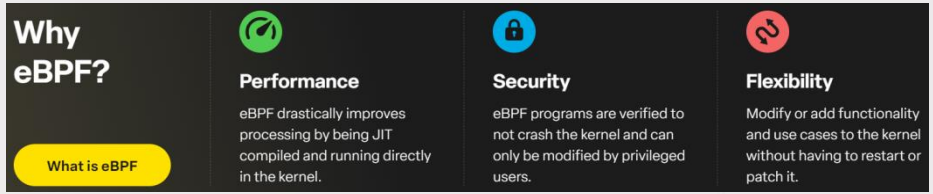
Source: "Dynamic Probes for Linux", Masami Hiramatsu, Tracing Summit 2015.



Source: https://ebpf.io/

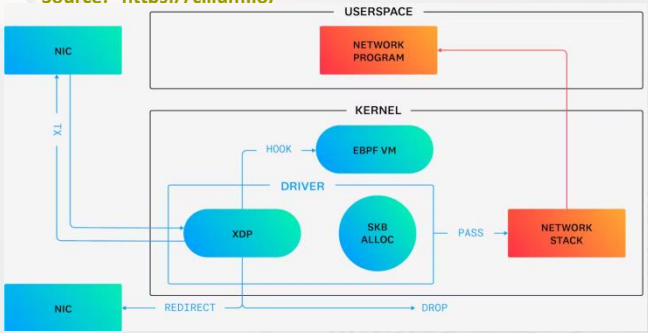


Source: https://ebpf.io/get-started/

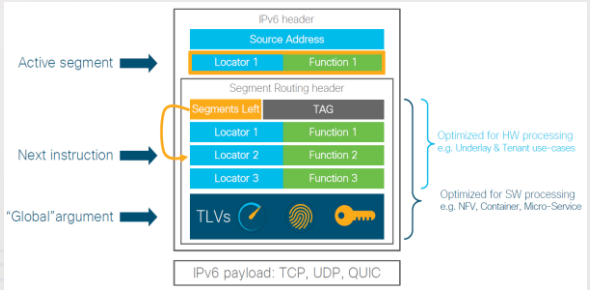


Source: https://ebpf.io/

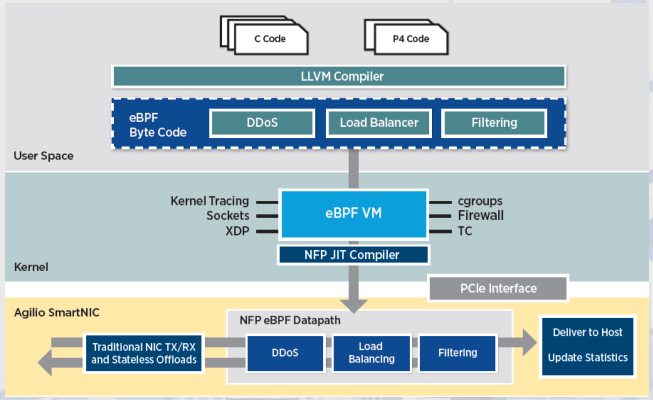
Source: https://cilium.io/



Source: https://www.datadoghq.com/blog/xdp-intro/

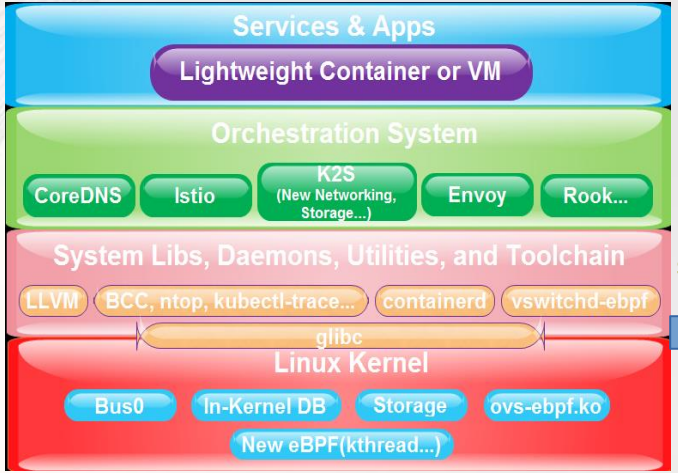


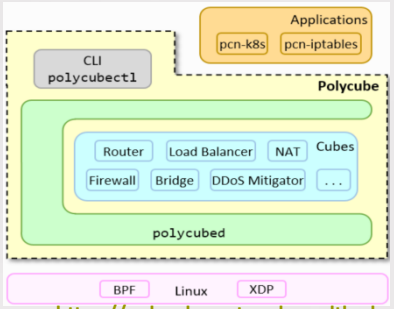
Source: http://www.segment-routing.net/images/20190130-bcn-cl-BRKRST-3122-rev7f-km2.pdf



Source: https://www.netronome.com/m/documents/PB_Agilio-eBPF.pdf

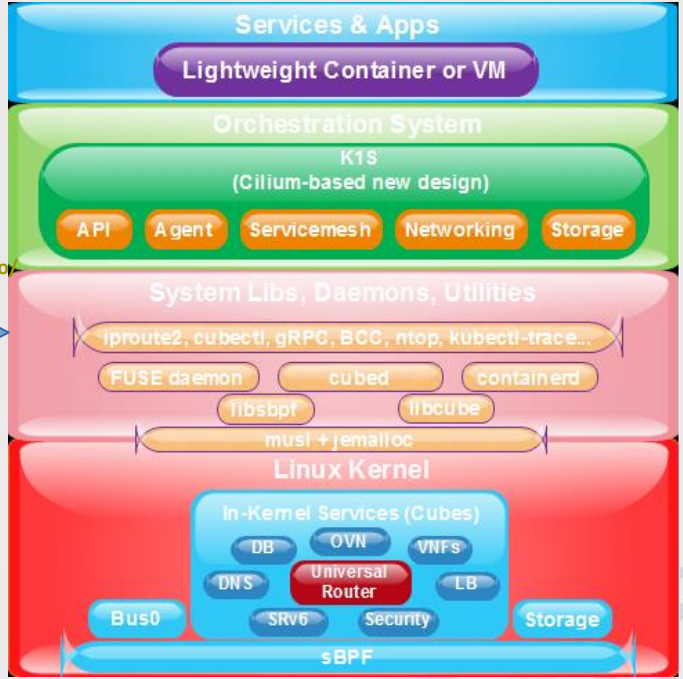
System architecture for the first stage of our approach:



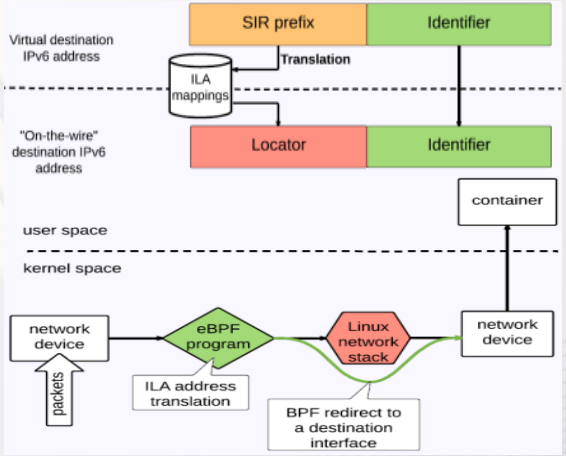


Source: https://polycube-network.readthedocs.io/en/latest/intro.html

System architecture for the second stage of our approach:

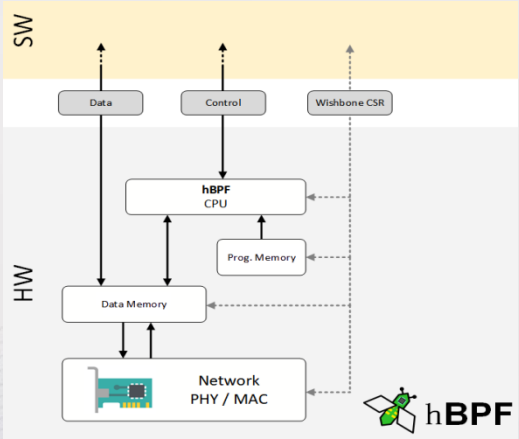


ILA for Container and PoC implementation:



Source: https://www.delaat.net/sc/sc17/posters/ila-poster-SC17.pdf

eBPF in hardware:



Source: https://github.com/rprinz08/hBPF/blob/main/doc/images/hbpf-overview.png

A customized SmartNIC for Edge Computing in our approach:

- A FPGA-based SmartNIC that fits inside the USB port;
- At least two interfaces:
  a USB port for host connection, and a RJ45 or Mini I/O connector for networking;
- The main logic in FPGA is to implement the desired functionalities of the SmartNIC, conversion of I/O signals, and so on;
- Support for USB 4.0 is mostly preferred(so the host must also support USB 4.0).
- ...

The main reason that this design do not use PCIe interfaced SmartNIC is base on the consideration of add more flexibility to our overall solution for Edge Computing.

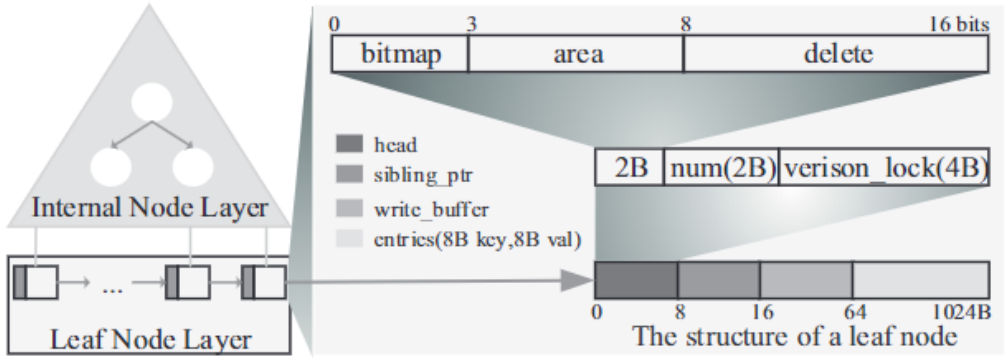
WPB+-tree:A Write-Optimized PM-oriented B+-tree with Aligned Flush and Selective Migration

Presenter : Mingjie Li

貴州大學
GUIZHOU UNIVERSITY

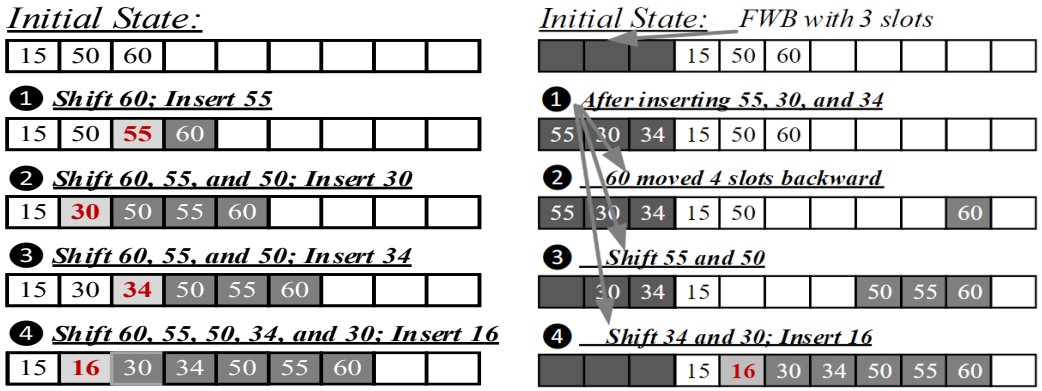
Design Overview

The structure of WPB+-tree



A leaf node is composed of a **metadata field**, a **FWB**, and a **slot array** that stores key-value entries.

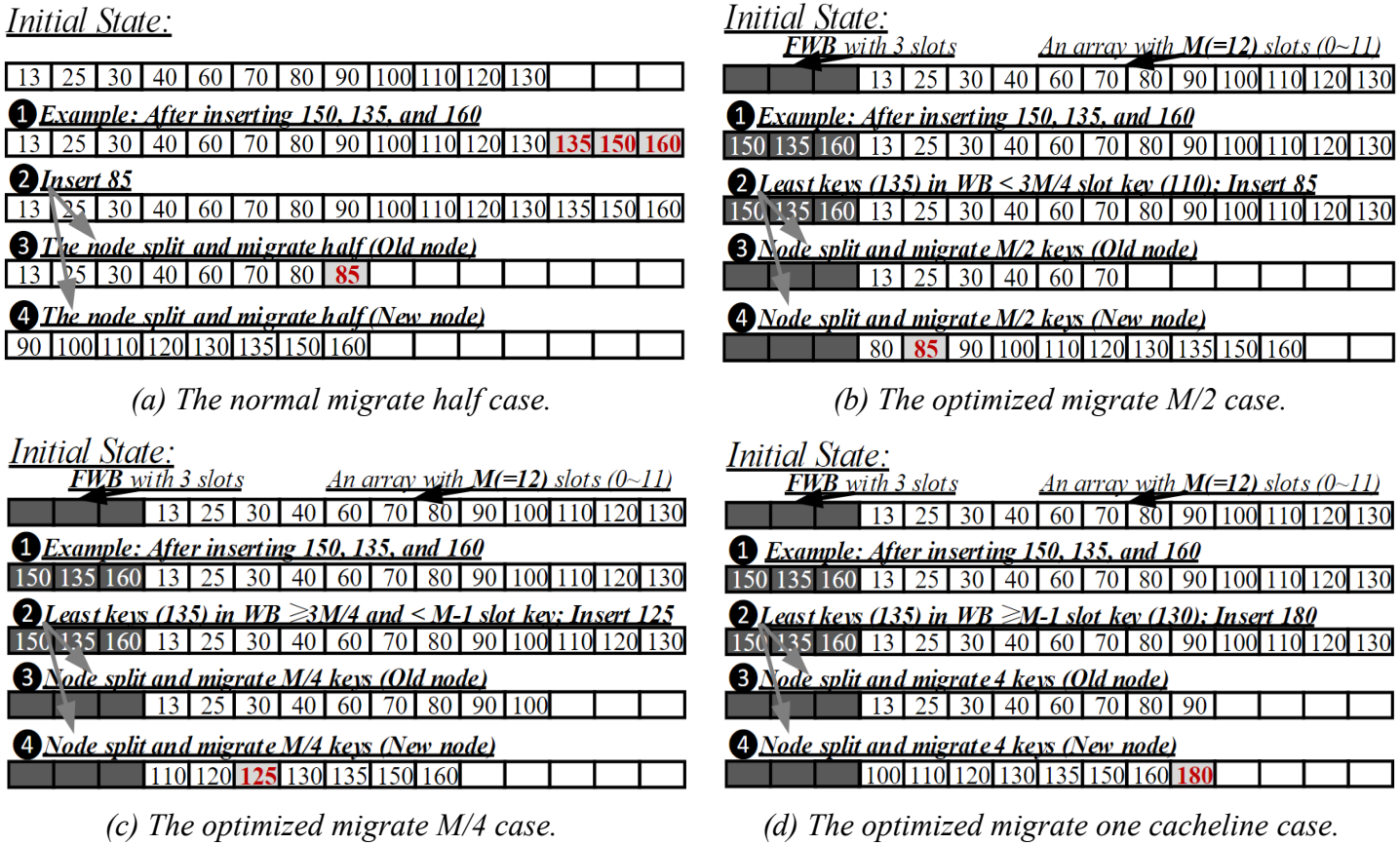
Design1: Flush-aligned Write Buffer



(a) The normal case. *(b) The optimized case.*

By using **FWB**, we issue fewer flush instructions.

Design2: Selective Migration of Node Entries



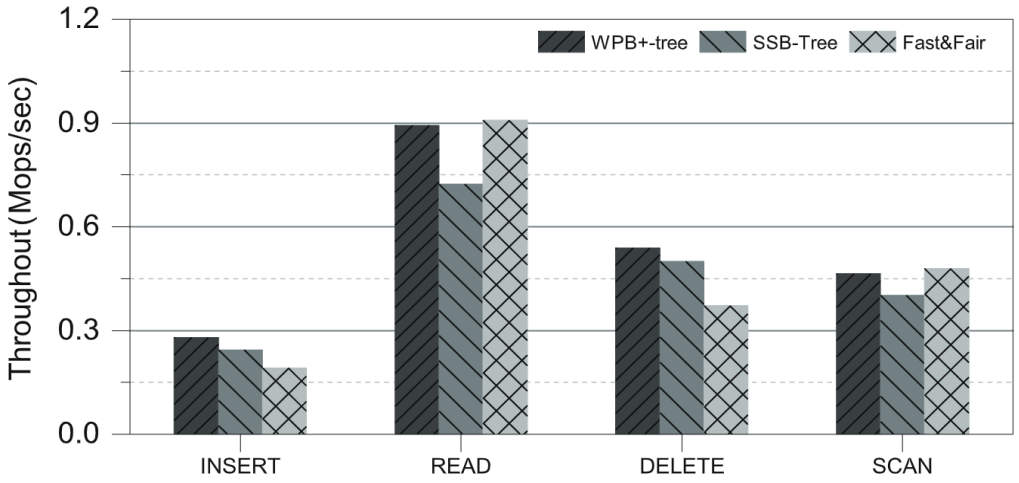
(a) The normal migrate half case. *(b) The optimized migrate M/2 case.*

(c) The optimized migrate M/4 case. *(d) The optimized migrate one cacheline case.*

To decline entry migrations during node splits, we propose the **Selective Migration of Node Entries (SMNE)** mechanism to determine the number of migrated entries according to the key distribution.

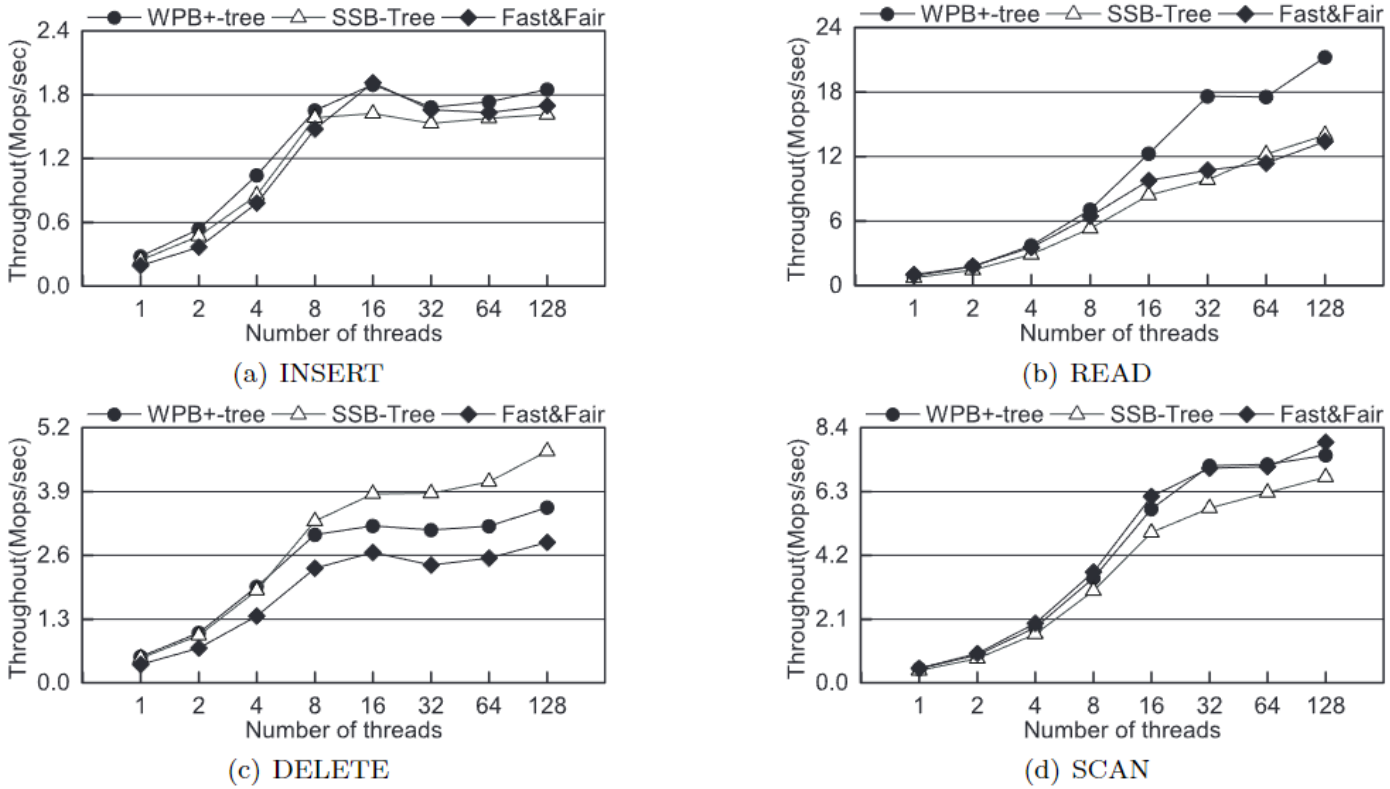
Result

Performance of single-threaded



- Write performance. Compared with SSB-Tree and FAST&FAIR, WPB+-tree delivers **13%** and **31%** higher throughput.
- Read Performance. WPB+-tree is very approximate with FAST&FAIR, while achieving **20%** higher than SSB-Tree.
- Delete Performance. WPB+-tree is similar to SSB-Tree, and **31%** higher than FAST&FAIR.
- Scan Performance. WPB+-tree is very approximate with FAST&FAIR, while achieving **14%** higher than SSB-Tree.

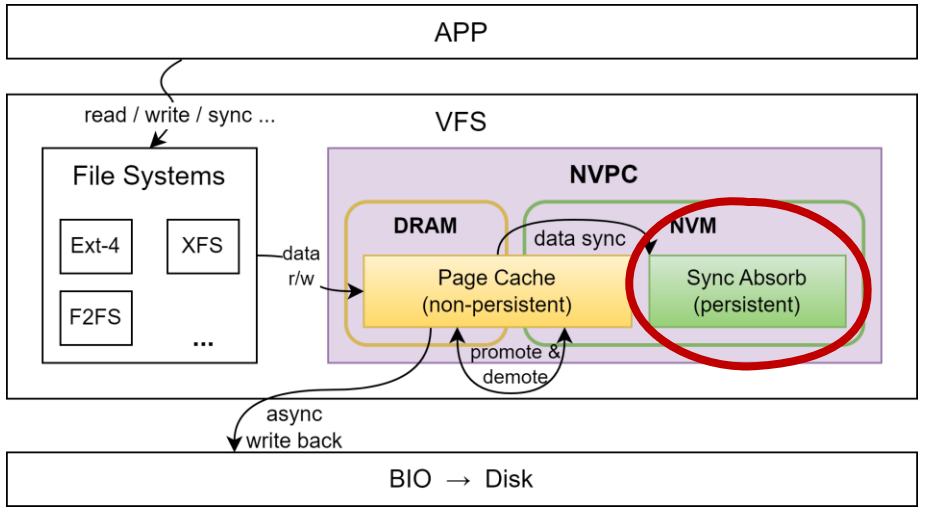
Performance comparison of multi-threaded.



- Compared with state-of-the-art solutions (i.e., FAST\&FAIR and SSB-Tree), WPB+-tree achieves up to **17%**, **40%**, and **20%** higher throughput in the insert, search, and scan operations, respectively.
- SSB-Tree shows a better throughput than WPB+tree after 8 threads. The throughput of WPB+-tree is higher than FAST&FAIR and can gradually approach SSB-Tree.

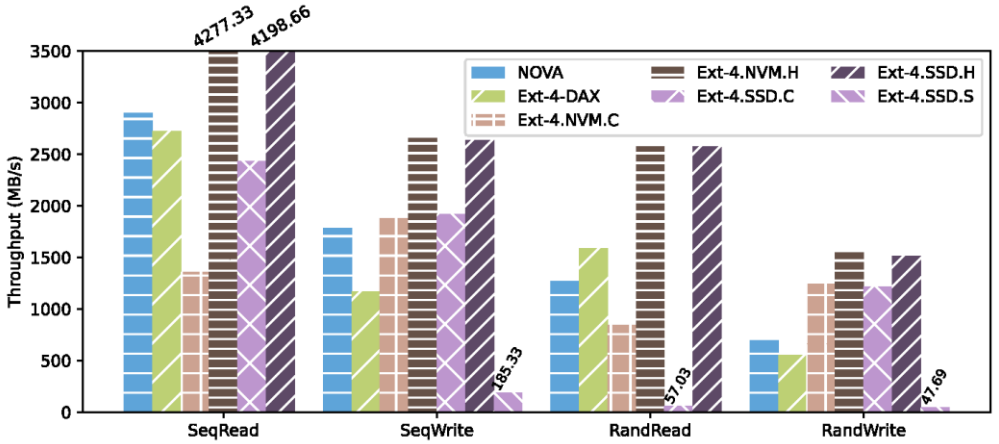
Boosting File Systems Elegantly: a Case for a Transparent NVM Page Cache

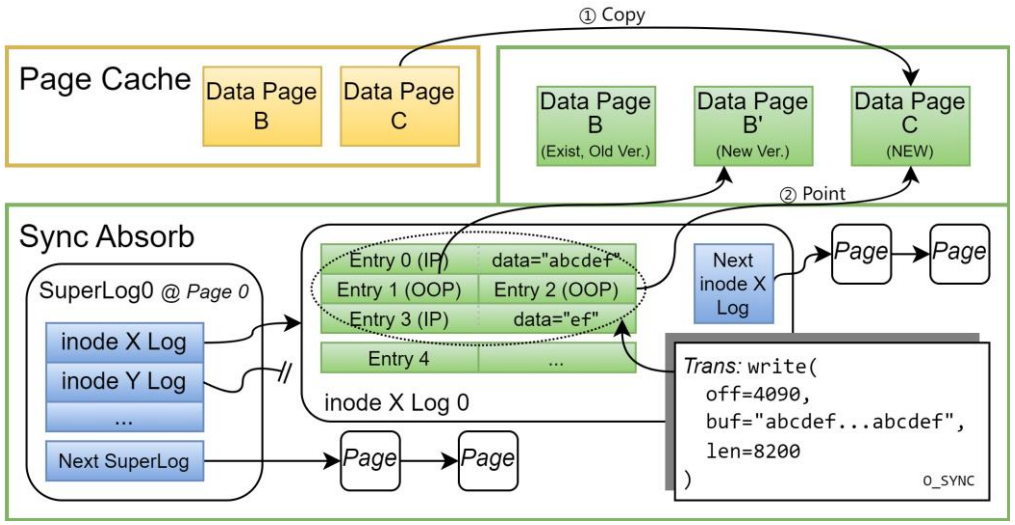
Authors: 王国毓, 车喜龙, 魏皓阳, 陈硕, 何璞昳, 胡俊成



Presenter: Guoyu Wang (王国毓)
Jilin University

Email: wgy21@mails.jlu.edu.cn WeChat & Tel: 13274475156





Shortages for traditional FS:

Cold ops & ***Sync writes!***

Tiered memory

SPFS
P2CACHE

⇨ Slow & Unstable

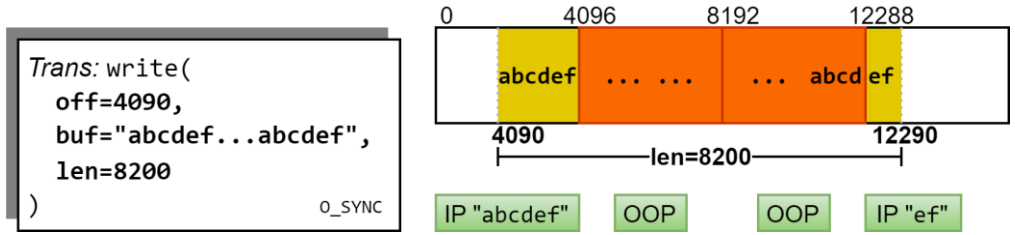
Principles:

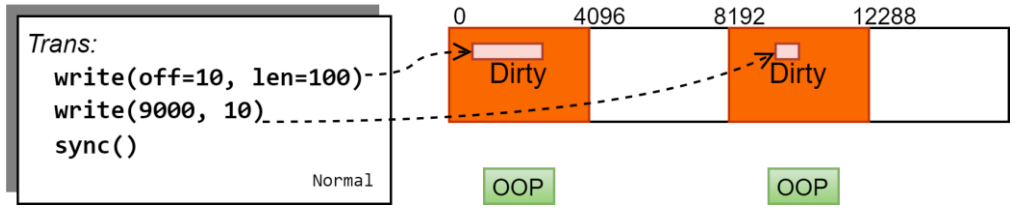
1. **Full transparency**: to user & to FS
2. **No consistency change**: protect the sync
3. **No perf downgrade**: only SPEED UP!
4. **Lightweight**: like a WAL of the FS

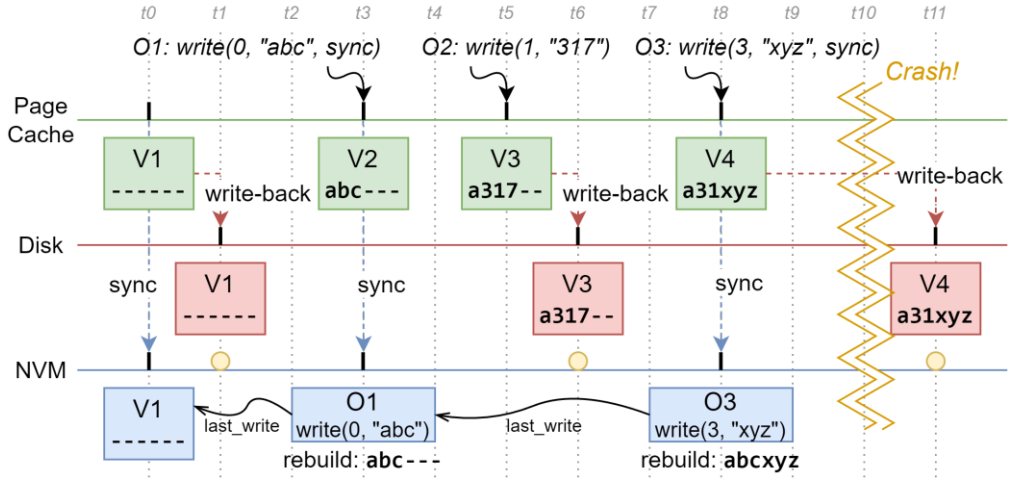
Design:

1. On demand sync absorb
2. Optimized for both large and small writes
3. Active sync prediction for more performance
4. Consistency problems solved

Email: wgy21@mails.jlu.edu.cn WeChat & Tel: 13274475156



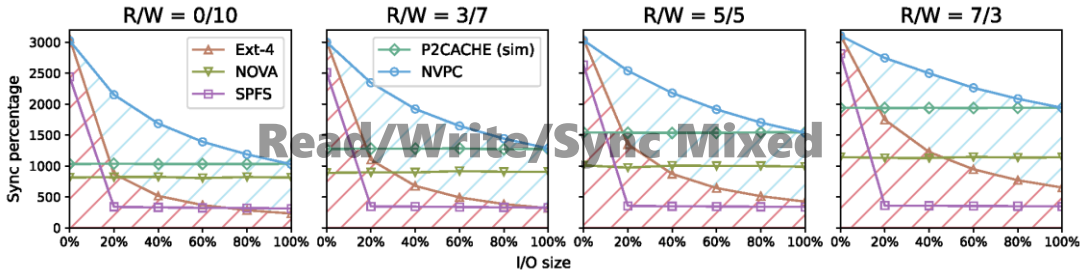


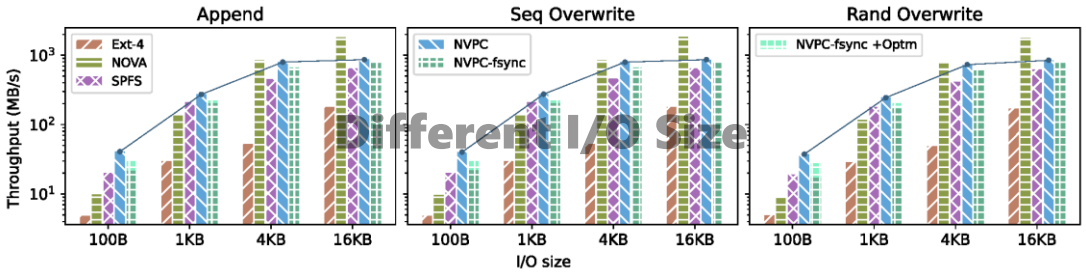


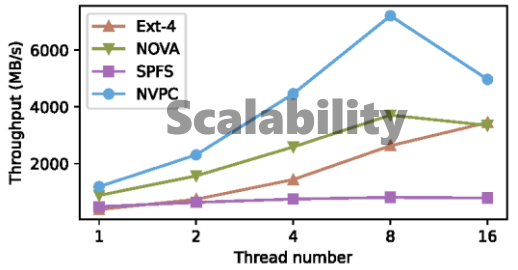
Design:

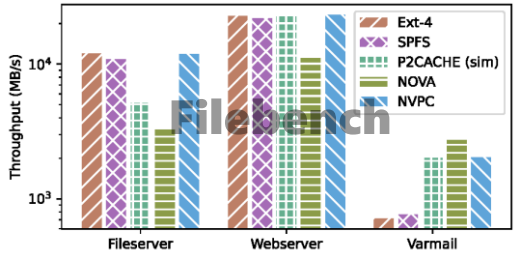
1. On demand sync absorb
2. Optimized for both large and small writes
3. Active sync prediction for more performance
4. Consistency problems solved

Performance:









Email: wgy21@mails.jlu.edu.cn WeChat & Tel: 13274475156

CCL-BTree: A Crash-Consistent Locality-Aware B+-Tree for Reducing XPBuffer-Induced Write Amplification in Persistent Memory

Zhenxin Li , Shuibing He , Zheng Dang, Peiyi Hong ,
Xuechen Zhang †, Rui Wang , Fei Wu

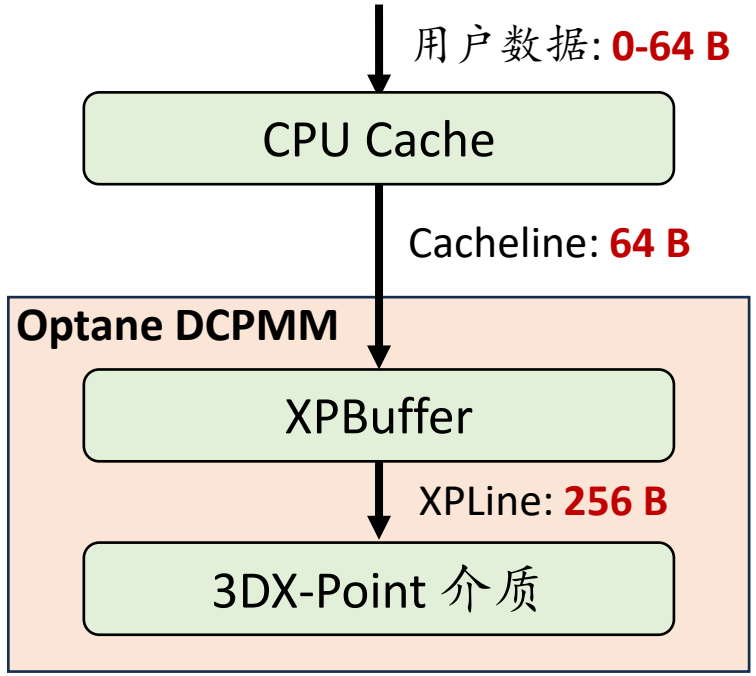
浙江大学
Zhejiang University

† WASHINGTON STATE UNIVERSITY

EuroSys 2024

研究动机：持久性B+树中的写放大问题

- 小粒度（<64B）的写操作会在两个层级中遇到写放大问题



缓存行写放大（Cacheline-induced amplification： CLI）

$$CLI = \frac{\text{写入XPBuffer的数据量}}{\text{用户写入数据量}}$$

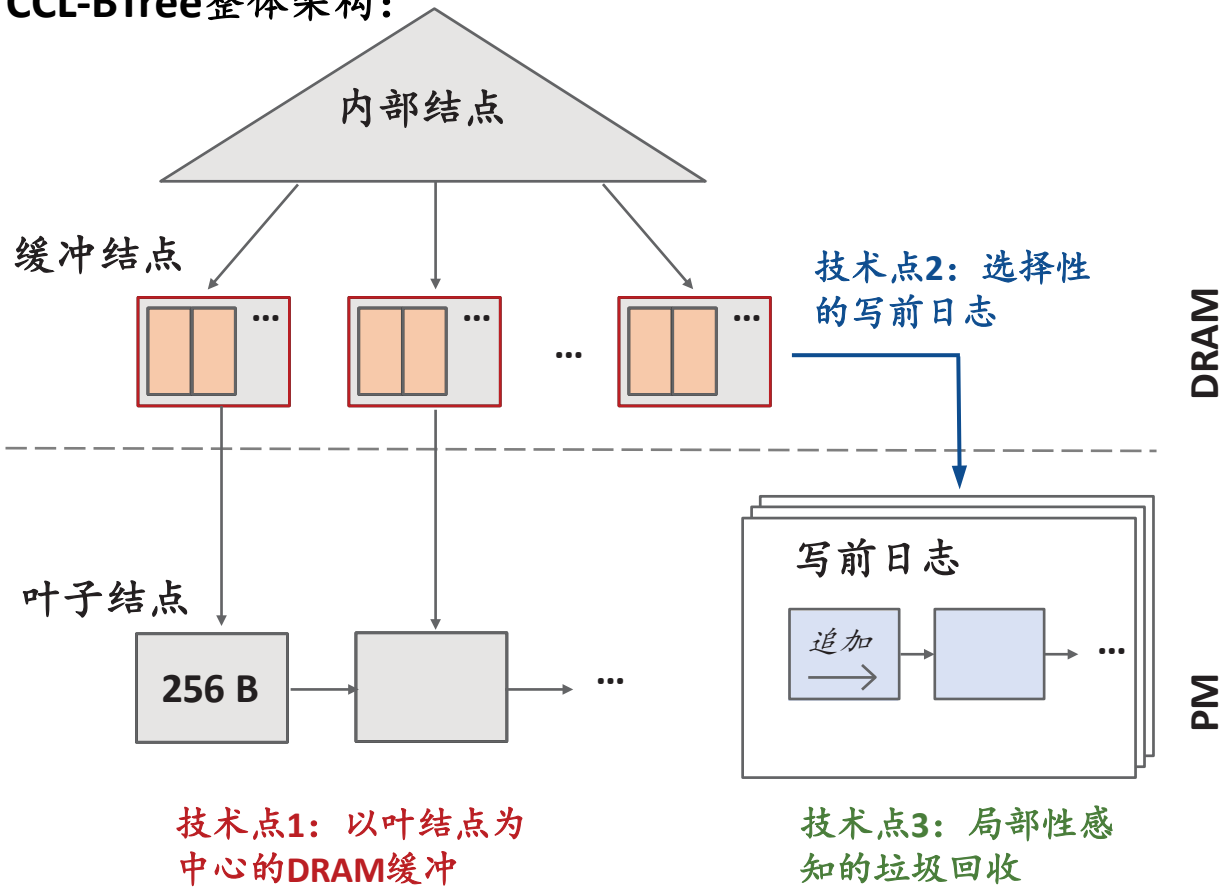
XPBuffer写放大（XPBuffer-induced amplification： XBI）

$$XBI = \frac{\text{写入介质中的数据量}}{\text{用户写入数据量}}$$

- 在写入线程数量较多时，XBI写放大决定着写性能
- 现有的持久性B+树索引存在严重的XBI写放大问题

CCL-BTree设计方案

CCL-BTree整体架构：



- **两个主要的结构：**
  - **缓冲结点**：在缓冲结点中缓存多个KV，并将其批量刷新到PM中的叶子结点。
  - **写前日志**：维护缓冲结点中数据的崩溃一致性。
- **三个主要技术点：**
  - **以叶结点为中心的缓冲**：减少写KV数据时的XBI写放大。
  - **选择性的写日志**：减少写日志数据时的XBI写放大。
  - **局部感知的垃圾回收**：减少垃圾回收过程中的XBI写放大。

- **效果：CCL-BTree能够减少81%的XBI写放大，并将插入性能最多提升9.35倍**
- **开源代码：https://github.com/ISCS-ZJU/CCL-BTree**

第26届中国计算机系统研讨会

The 26th ChinaSys Workshop

元数据导向的逻辑缺陷检测方法

Detecting Metadata-Related Logic Bugs in Database Systems via Raw Database Construction

宋建森*，窦文生，高钰，崔紫玉，郑莹莹，王栋，王伟，魏峻，黄涛

50th International Conference on Very Large Databases (VLDB 2024)

联系方式：songjiansen20@otcaix.iscas.ac.cn

ISCAS

中国科学院软件研究所

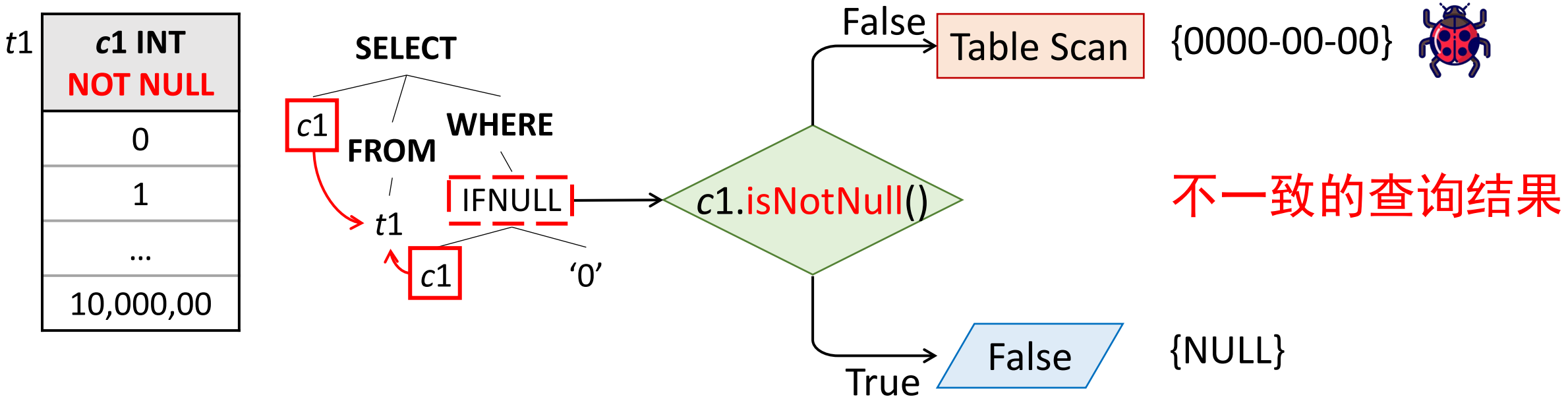
Institute of Software Chinese Academy of Sciences

中国科学院大学

University of Chinese Academy of Sciences

元数据相关的逻辑缺陷

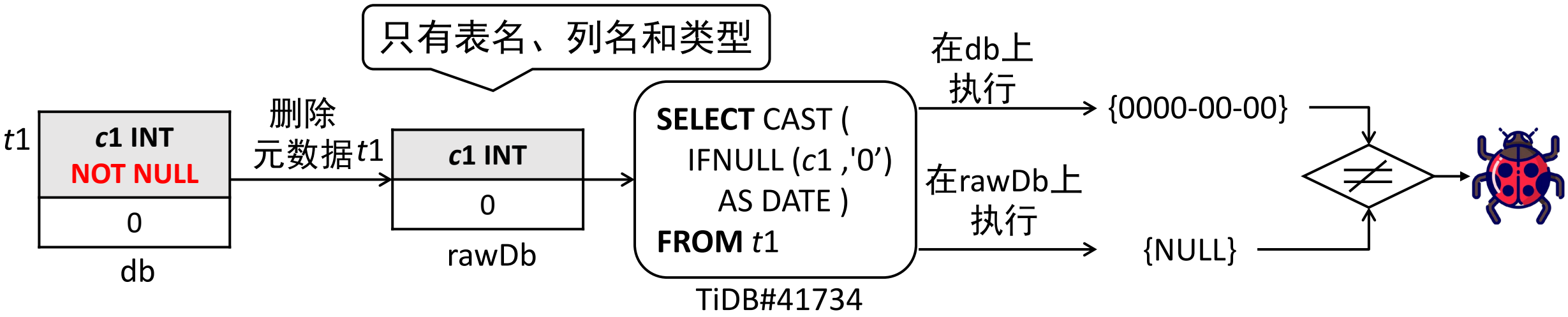
- 元数据描述了数据的组织方式，在查询优化中发挥着重要作用
- 不正确的元数据优化会引入逻辑错误，导致不正确的查询结果



[1] TiDB#41734. https://github.com/pingcap/tidb/issues/41734

元数据导向的逻辑缺陷检测方法

- 构造数据库元数据与查询语句执行结果之间的等价蜕变关系
  - 共计检测42个缺陷，其中38个被确认为是新的缺陷，16个已被修复
  - 相关工作至多能够检测13个本方法发现的元数据相关的逻辑缺陷



17th IEEE International Conference on Software Testing, Verification and Validation (ICST) 2024

Differential Optimization Testing of Gremlin-Based Graph Database Systems

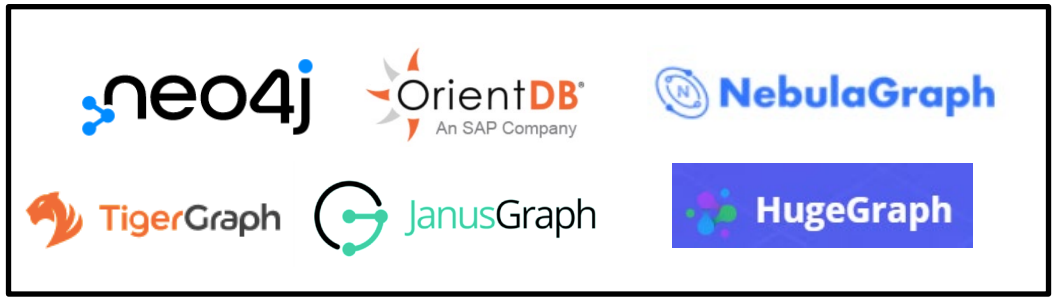
Yingying Zheng, Wensheng Dou, Lei Tang, Ziyu Cui, Jiansen Song, Ziyue Cheng, Wei Wang, Jun Wei, Hua Zhong, Tao Huang

ISCAS
中国科学院软件研究所
Institute of Software Chinese Academy of Sciences

中国科学院大学
University of Chinese Academy of Science

Optimization Bugs in Graph Database Systems (GDBs)

- GDBs support efficient storage and queries for graph data and support various optimization strategies to accelerate graph queries



- Incorrect optimizations in GDBs can introduce optimization bugs, which cause GDBs to return incorrect query results

```
g.withoutStrategies(LazyBarrierStrategy)
  .withoutStrategies(HugeVertexStepStrategy)
  .E().bothV().not(__.in('acting'))
      -- v:{1,2,3,4} ❌
```

(a)

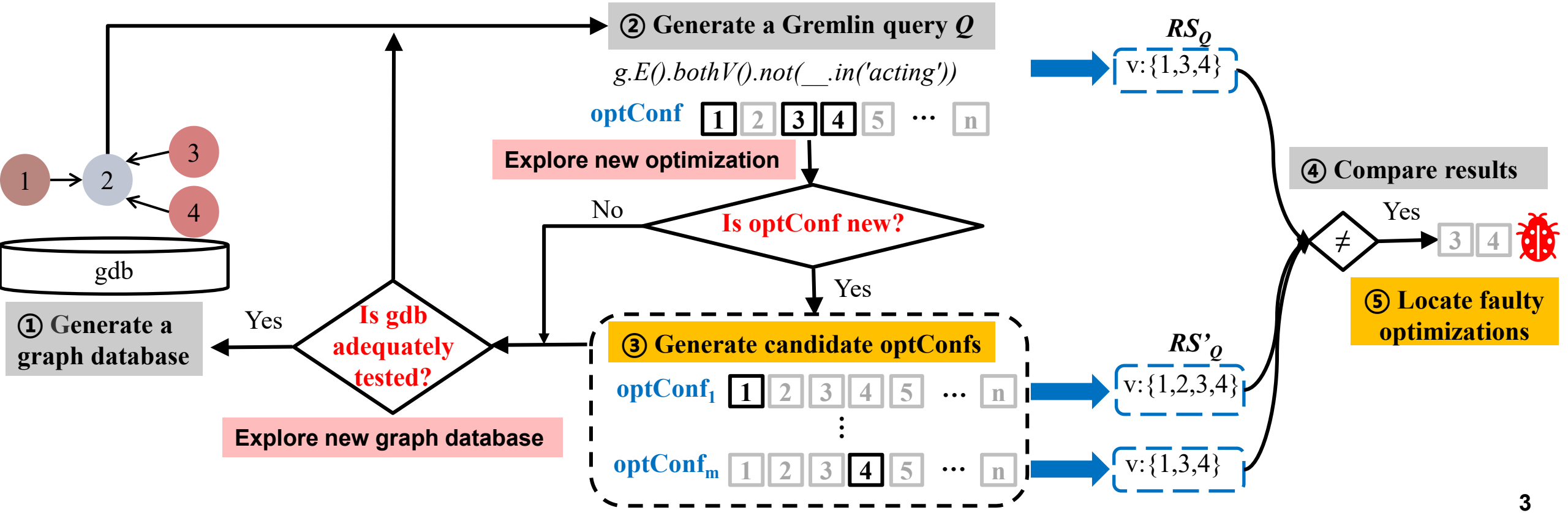
```
g.withStrategies(LazyBarrierStrategy)
  .withStrategies(HugeVertexStepStrategy)
  .E().bothV().not(__.in('acting'))
      -- v:{1,3,4} ✓
```

(b)

An optimization bug detected by our approach in HugeGraph

Differential Optimization Testing (DOT)

- DOT executes a query *Q* with two different optimization configurations that should retrieve the same query result
  - 28 optimization bugs in six popular GDBs are detected
  - 19 out of 28 bugs cannot be detected by the existing approaches



ChunkGraph: Large Graph Processing with Chunk-Based Graph Representation Model

USENIX ATC'24

Rui Wang[1,2] **Weixu Zong[1]** Shuibing He[1] Xinyu Chen[1]

Zhenxin Li[1] Zheng Dang[1]

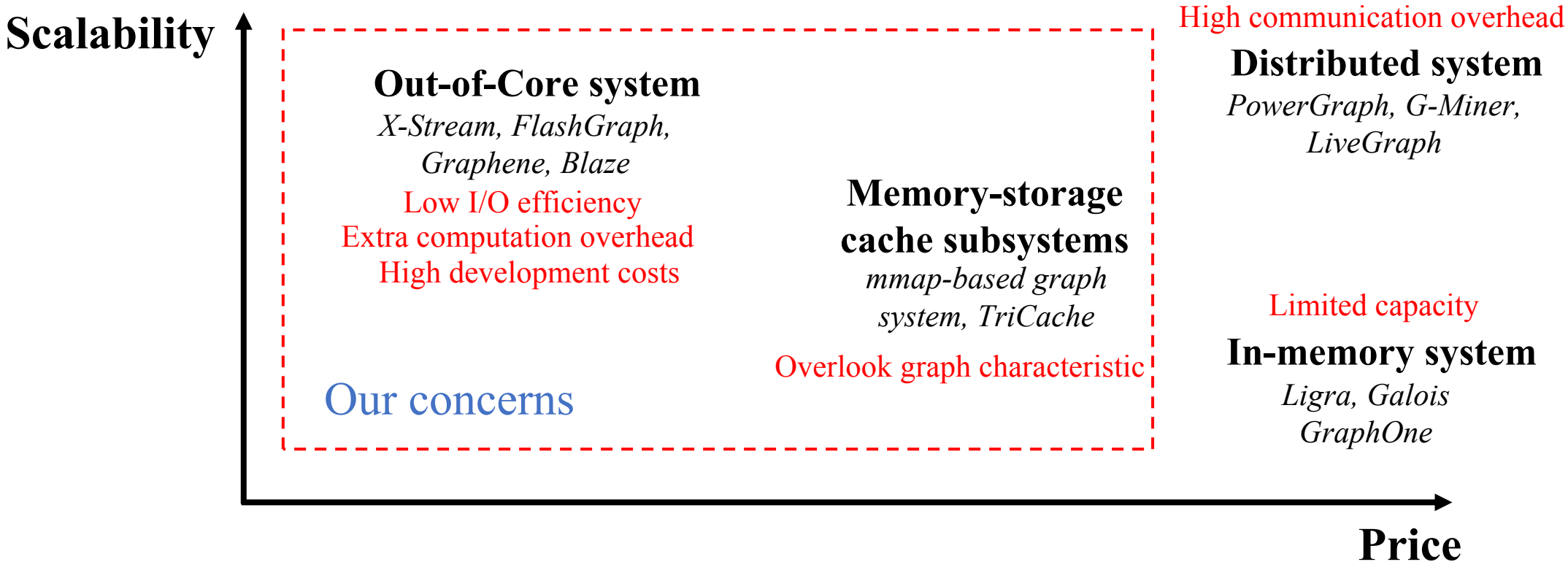
[1]Zhejiang University

[2]Hangzhou High-Tech Zone (Binjiang) Institute of Blockchain and Data Security

浙江大学

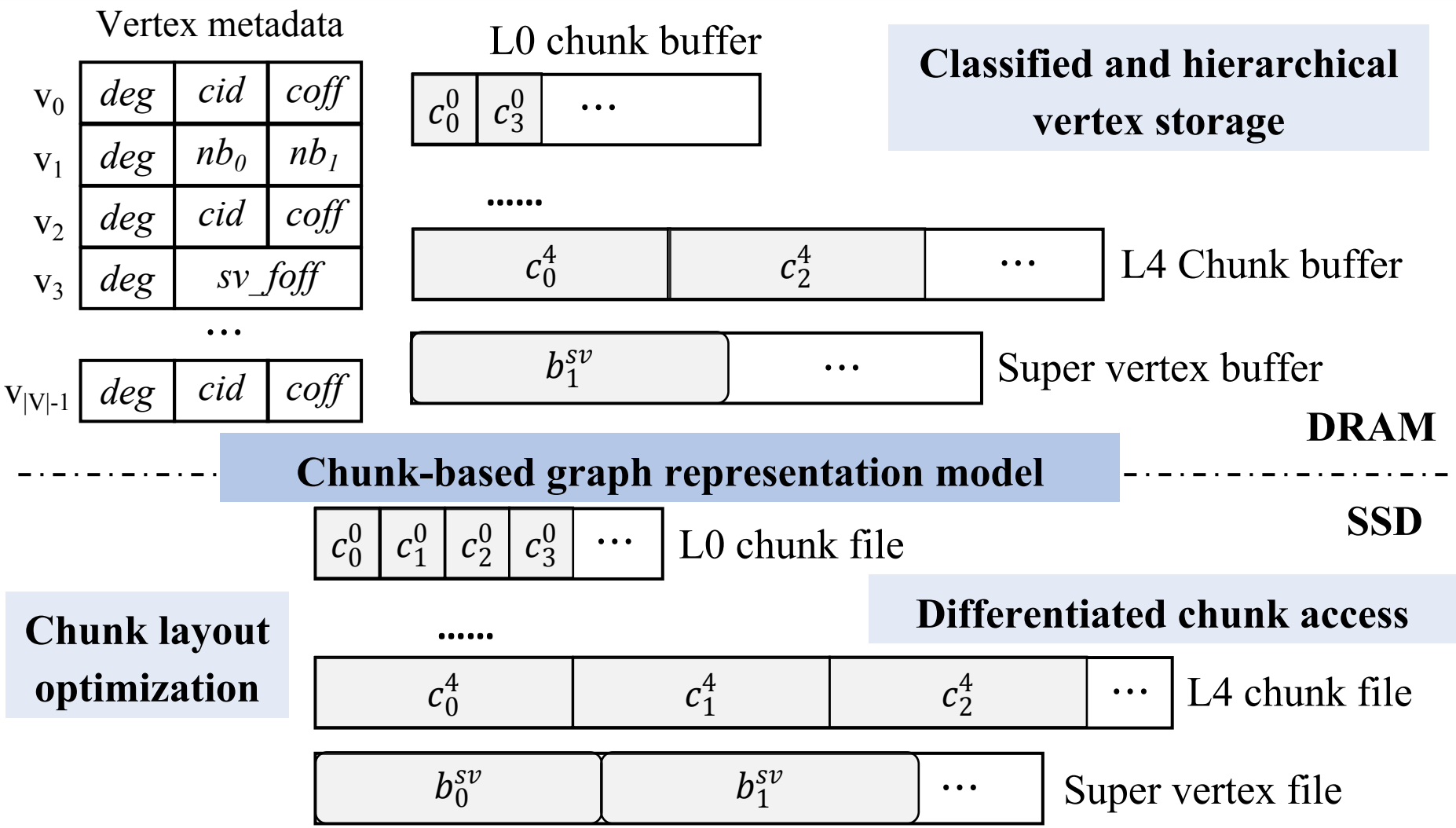
Zhejiang University

Real-world graph datasets are continuously growing



ChunkGraph: Better scalability, Better I/O efficiency, Lower development costs

ChunkGraph Design



ChunkGraph outperforms SOTA external systems and general cache systems

Exploit both SMART Attributes and NAND Flash Wear Charateristics to Effectively Forecast SSD-based Storage Failures in Clusters

Yunfei Gu[1,2], Chentao Wu[1,2*], Xubin He[2]

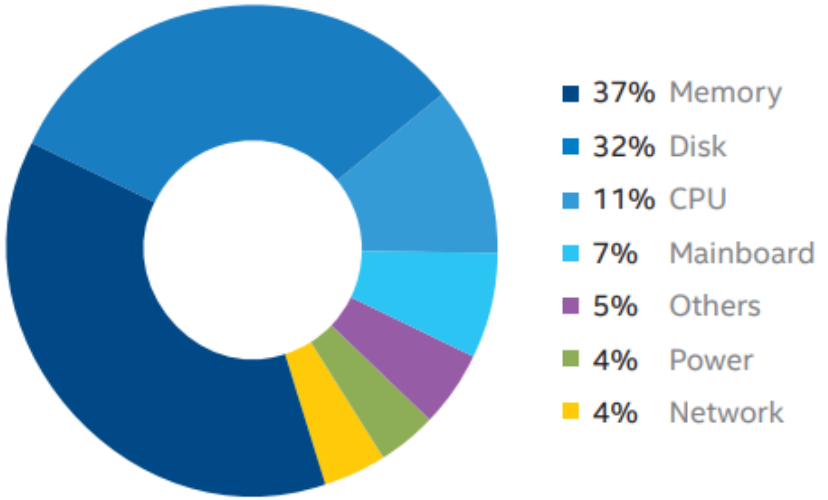
[1]*Department of Computer Science and Engineering, Shanghai Jiao Tong University*

[2]*Shanghai Key Laboratory of Trusted Data Circulation andGovernance in Web3, Shanghai Jiao Tong University*

[3]*Department of Computer & Information Sciences, Temple University*

大规模集群中SSD故障预测研究背景

随着数据中心存储系统的快速扩展与规模日益增大，存储系统可靠性已成为影响大规模数据中心集群性能的重要因素。



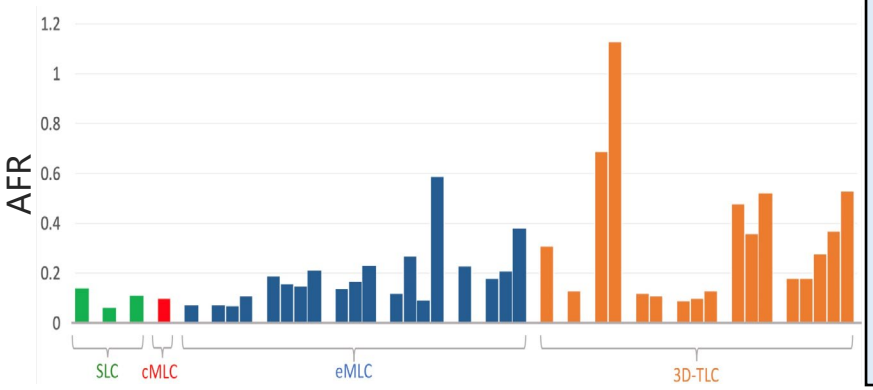
京东云数据中心系统故障占比分布

- 在云计算、人工智能、大数据时代，随着数据中心存储系统的快速扩展，存储系统可靠性成为影响数据中心集群性能的重要原因之一。
- 当前数据中心的系统故障的主要原因包括：硬盘故障和内存故障，硬盘故障占比高达32%。
- 各大云厂商例如阿里云、腾讯云、华为云、京东云致力于发展智能运维系统AIOps，通过故障预测机制来预测硬盘故障，及时发出运维工单请求保障集群的可靠性，提供较高的服务质量。

Source: 英特尔MCA+MFP助力京东构建稳定、高效的云服务，https://www.intel.com/content/dam/www/central-libraries/us/en/documents/2023-12/mca-mfp-helps-jd-stable-and-efficient-cloud-services.pdf.

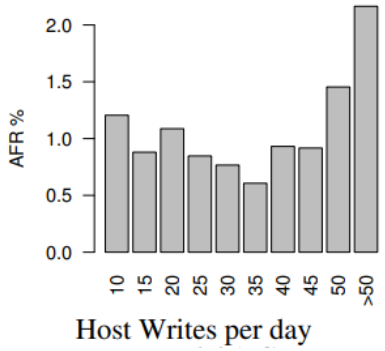
大规模集群中SSD故障预测研究背景与挑战

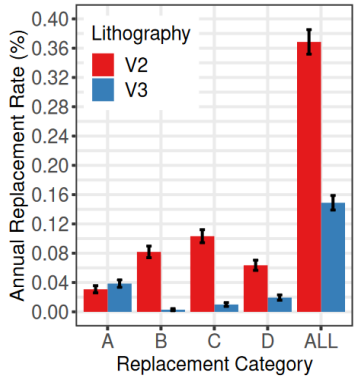
固态硬盘SSD故障率受多种因素制约，仅利用SMART信息会导致故障预测精度不高。



Source: CDC report 2023

SSD故障率与闪存技术相关



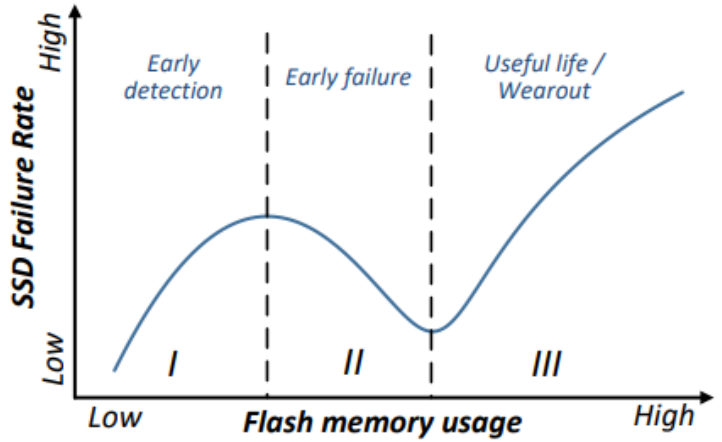


SSD故障率与写磨损和光刻工艺相关

Source: SSD Failures in Datacenters: What? When? and Why?. In Proc. of SYSTOR'16.

SSD故障曲线

SSD故障规律受NAND磨损曲线主导

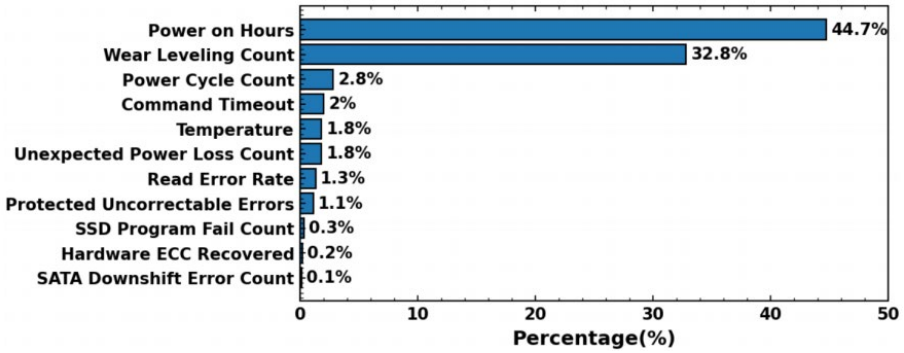


SSD SMART数据几乎无效，故障因素复杂，预测性能较差

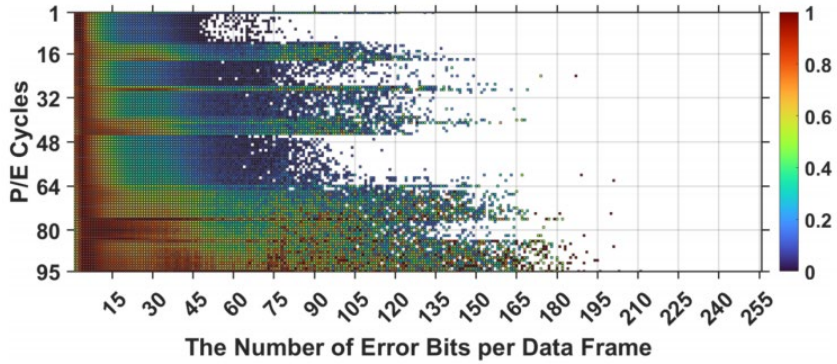
SSD故障与闪存介质层面、固件层面和系统层面多种因素相关，SMART日志基本无效

大规模集群中SSD故障预测——老化特性

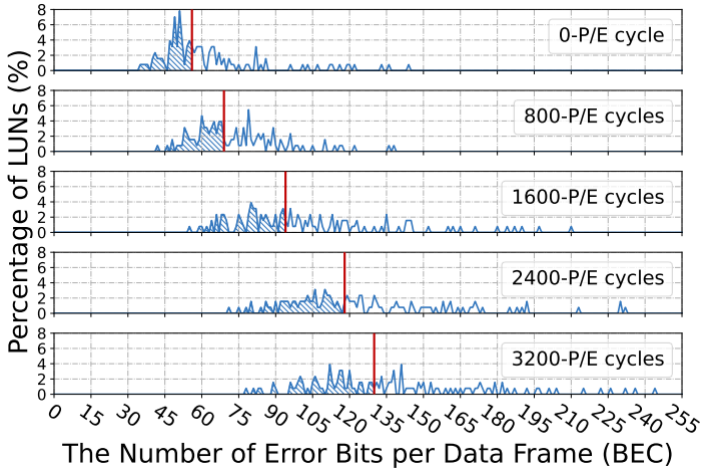
扩展先前SSD Die粒度闪存介质早期故障预测研究至在大规模存储系统中SSD全生命周期的故障预测研究。



SSD SMART属性与SSD故障Spearman相关系数分析



3D-TLC SSD 闪存Die中某LUN老化过程中数据帧位错误数(BEC)的热力图



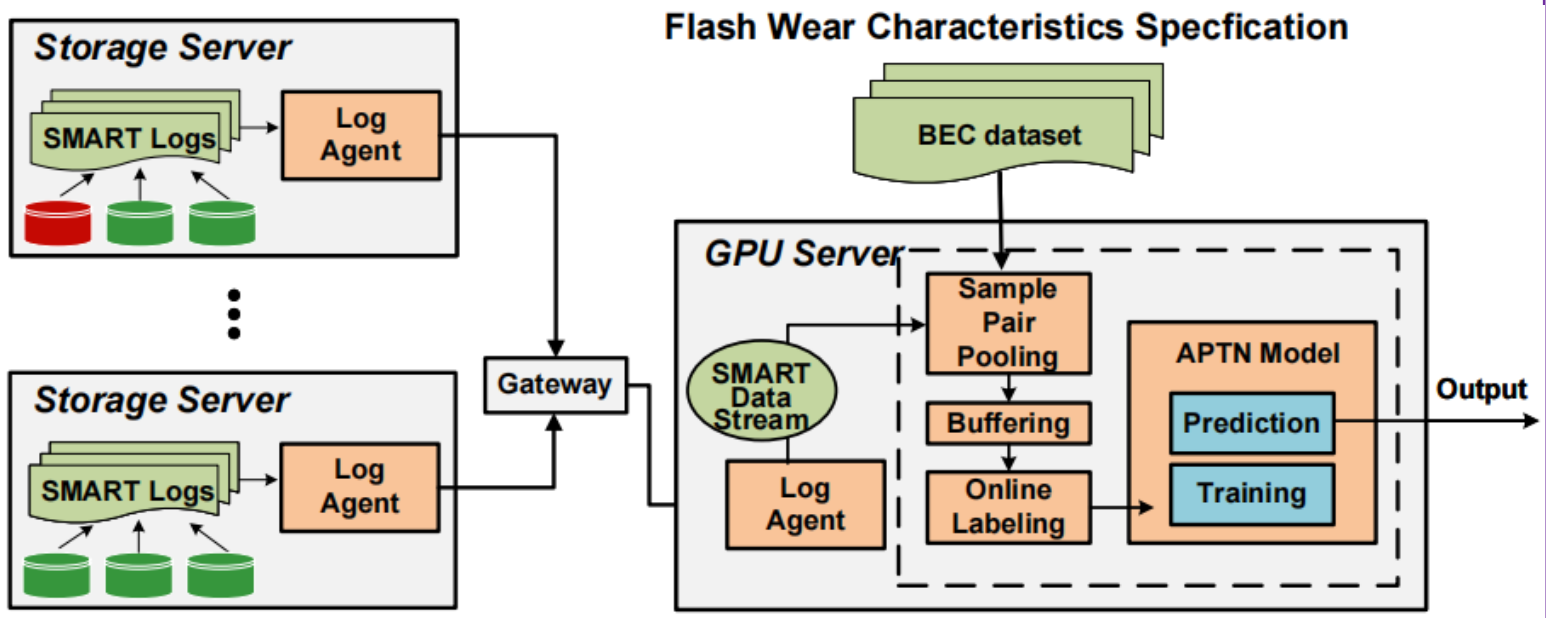
3D-TLC SSD 闪存Die中LUN（逻辑单元号）中数据帧最大位错误数的统计分布图

- SSD厂商反应SSD SMART属性相较HDD而言，相关性大幅度减弱，难以支撑有效、准确地故障预测。
- 现有SSD SMART属性中，Power on Hours和Wear Leveling Count属性相关性较高，符合闪存特性。
- 通过FTL固件层面进行老化磨损测试，并获取3D-TLC的老化特性。
- 随着P/E周期的增加,最大BEC分布向右移动,表明该驱动器在经历更多P/E周期后,失效概率会增加。
- 挑战：SSD老化磨损特性为NAND闪存介质的静态属性，与云系统中SMART数据不在一个维度，不能直接结合使用。

大规模集群中SSD故障预测——提出APTN（1）

提出基于SSD闪存介质磨损特性和SMART数据的故障预测框架和AI预测算法(APTN)

大规模存储集群中SSD故障预测框架

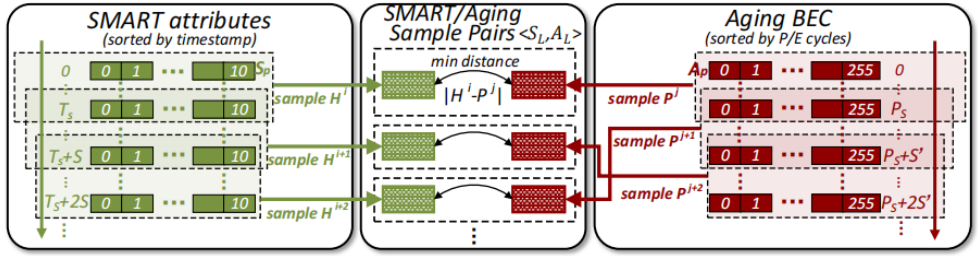


四个模块

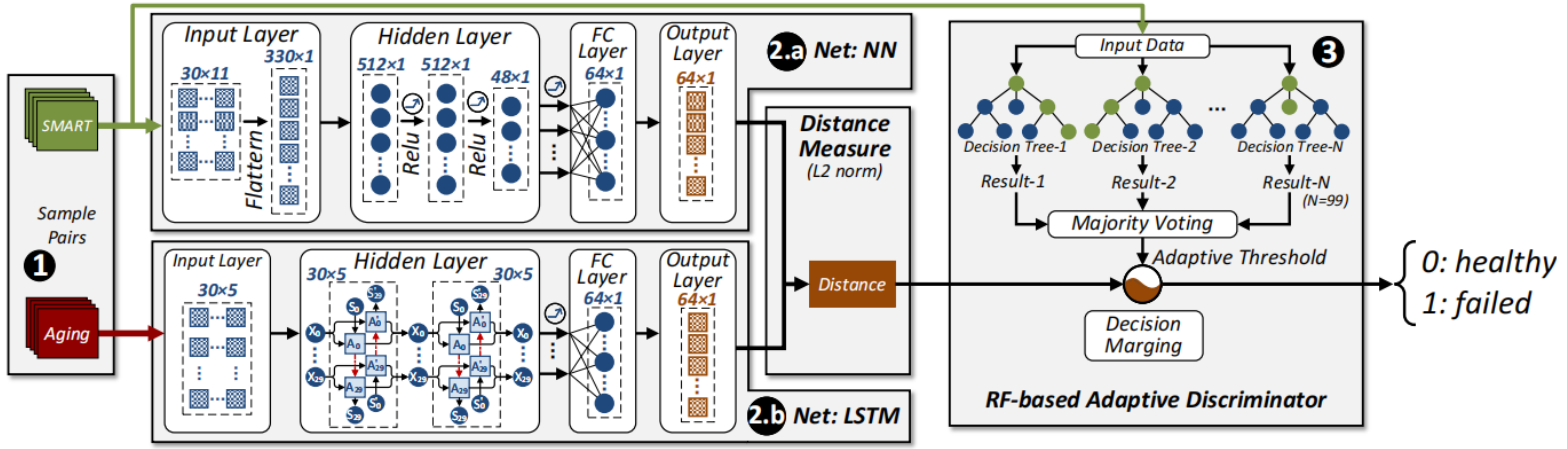
- 从存储服务器收集SMART日志,并传输到推理节点，与SSD闪存磨损特性数据进行样本配对；
- 离线预训练APTN模型；
- 在线预测SSD是否故障；
- 基于在线学习框架持续迭代训练APTN模型。

大规模集群中SSD故障预测——提出APTN（2）

利用伪孪生网络(APTN)，将SSD BEC老化数据和SMART数据映射到高维稀疏空间中，并计算欧氏距离进行相似性比较，从而预测SSD是否可能发生故障。



SMART属性数据与BEC老化数据配对示意图

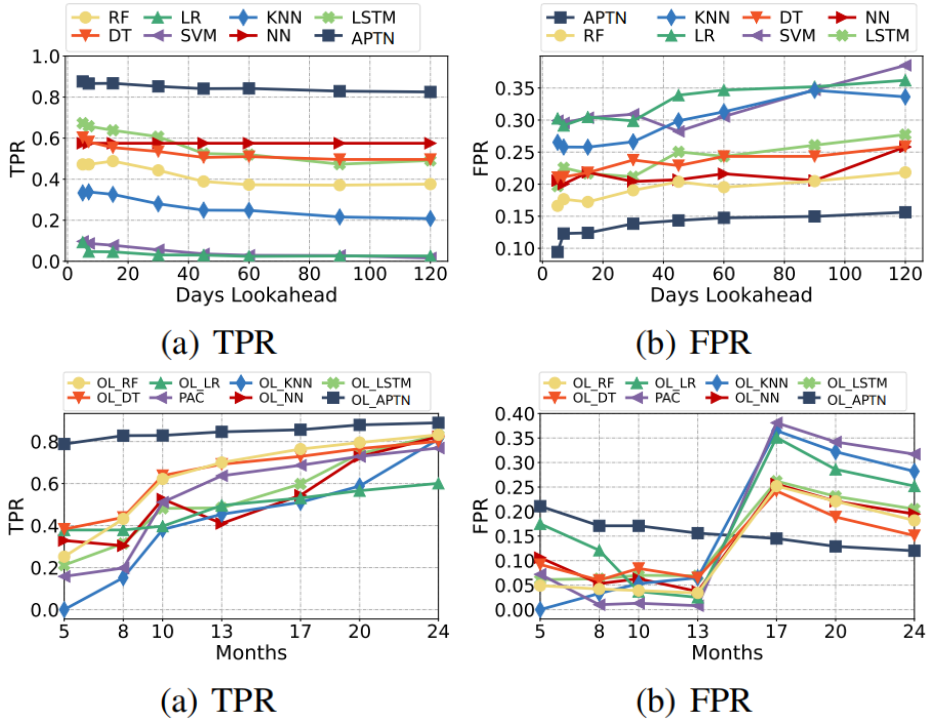


APTN (Aging-aware Pseudo-Twin Network)的SSD故障预测方法的网络结构图

- SSD BEC老化数据非实时收集，而是通过SSD制造商的内部团队在固件层面进行的离线磨损测试得到。
- SMART数据代表了SSD运行时获取的实时统计信息，与BEC老化数据存在差异，无法直接结合使用。
- APTN将此两类不同维度数据以P/E cycle为线索，在高维稀疏空间映射，实现高精准、低误判的故障预测。

大规模集群中SSD故障预测——测试结果

大规模存储系统，实现在不同提前时间前提下，仍具备高精准低误判的SSD故障预测，对智能运维系统(AIOps)实现在不同提前时间下的细粒度高效反应机制，提供可能性。



离线预测和在线预测的精准率和误判率

$$C-MTTDL = \frac{MTTDL}{Cost} \approx \frac{MTTF}{(1-\frac{k\mu}{\mu+\gamma})(C_a \times FP + C_b \times FN)}$$

C-MTTDL评价模型平均提升统计表

Method	TPR	FP	FN	MTTDL (years)	C-MTTDL(hours/dollars)
RF	47.2%	433	6148	154.7	0.96
DT	60.3%	979	4625	198.1	1.31
LR	9.3%	272	10559	94.8	0.37
PAC	9.7%	119	10514	95.1	0.38
KNN	33.3%	703	7759	125.7	0.60
NN	57%	529	4949	186.9	1.36
LSTM	67.3%	428	3801	233.5	2.19
APTN	**88%**	1133	1438	**478.7**	**5.66**

- 可提前7/15/30/45/60/90/120天实现实现高精准和低误判的SSD故障预测。将业界对3D-TLC SSD故障预测的准确率提升至90.1%水平，提升约40%。
- 实现经济分析指标C-MTTDL(成本-平均无故障时间)提高约2.5倍到15倍,大幅提高了存储系统的可靠性,且成本较低。

谢谢！

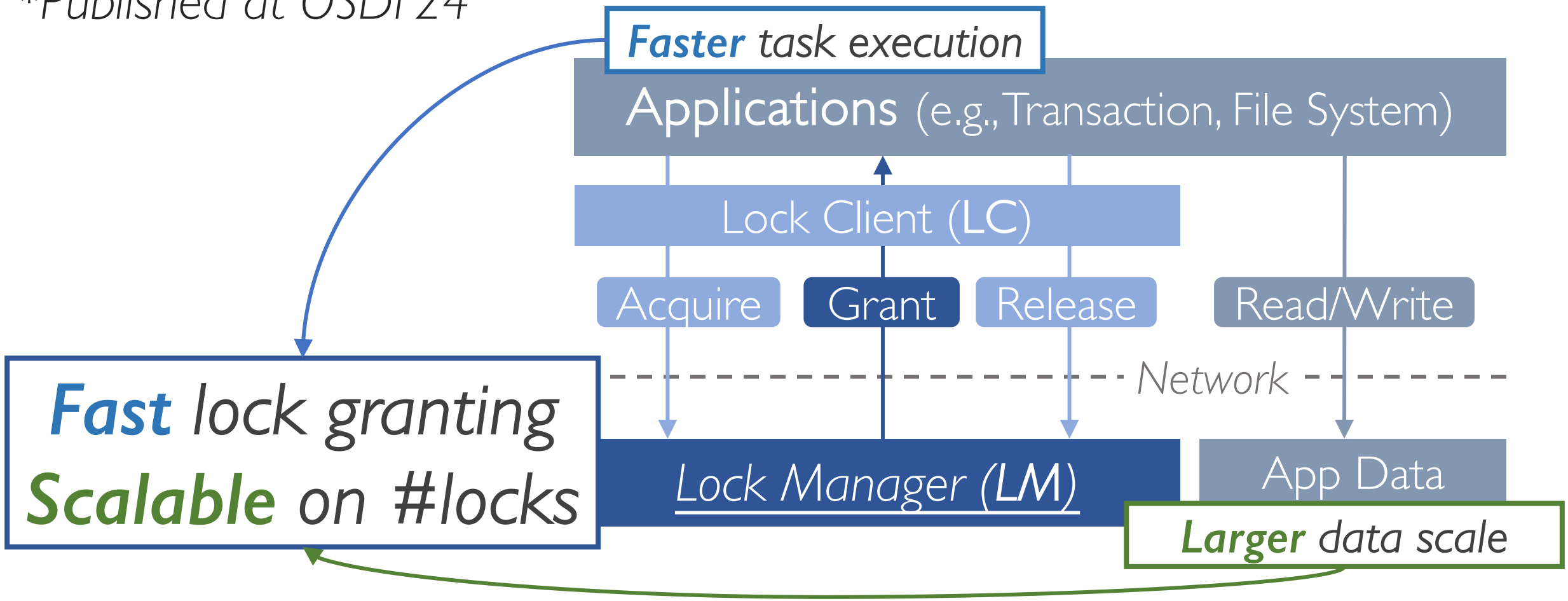
汇报人：谷云飞

2024年06月16日

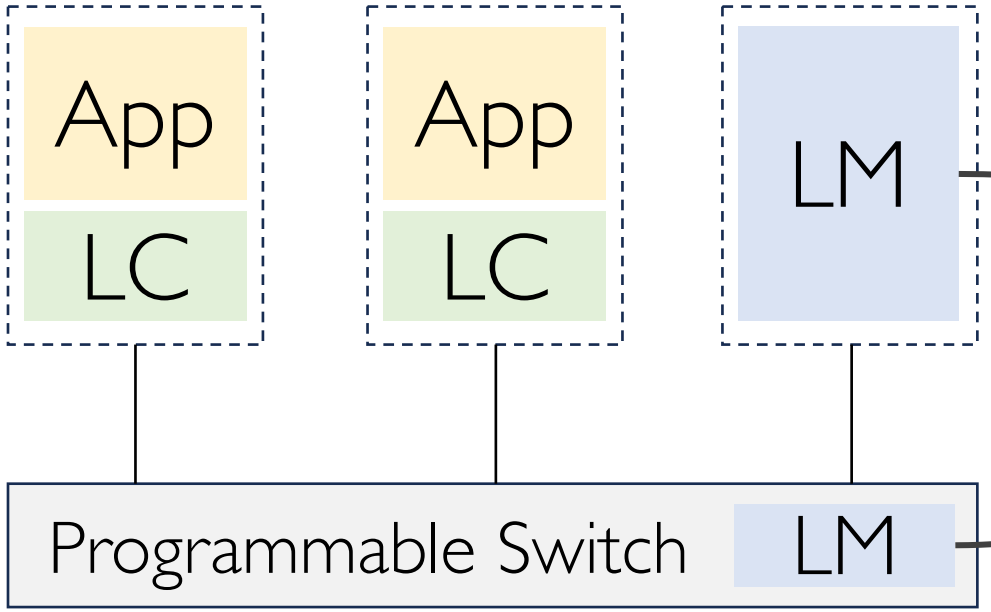
Fast and **Scalable** In-network Lock Management using **Lock Fission**

*Presenter: **Hanze Zhang** (IPADS, SJTU)*

**Published at OSDI'24*



Existing LMs fail to achieve both goals!



Limited CPU processing speed

Slow!

	Queueing delay	*Supported #locks*
SrvLock	Up to 100µs	>Million-scale
NetLock	< 1 µs	Thousand-scale

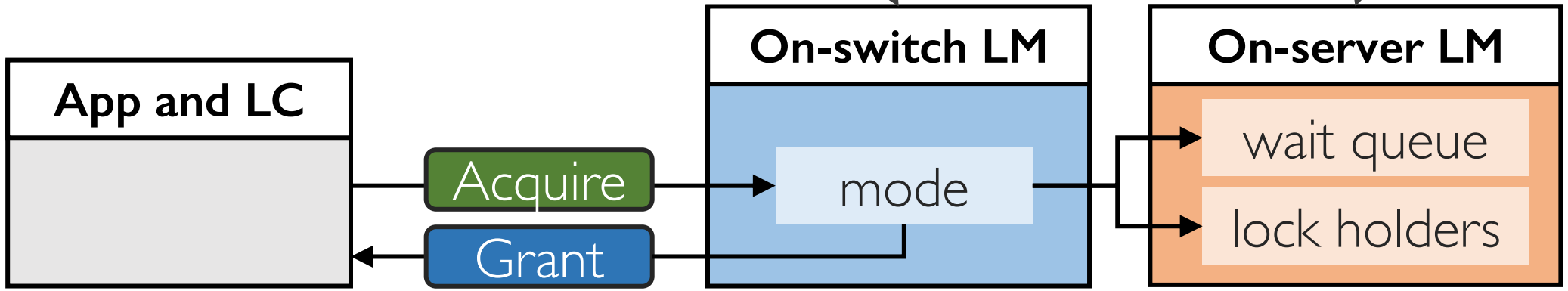
Unscalable!

Limited switch memory capacity

*What **FissLock** (our work) wants to achieve:*	< 1 µs	Million-scale

FissLock achieves both goals with *lock fission*

Decouple lock acquisition into grant decision and metadata maintenance!



Fast grant;
Memory efficient

Off critical-path

Evaluation results:

90% tail latency cut, 2x transaction throughput boost *compared to SOTA*

Scales to millions of locks when maintaining peak performance

HADB: Hotness-Aware Key-Value Store with Persistent Memory

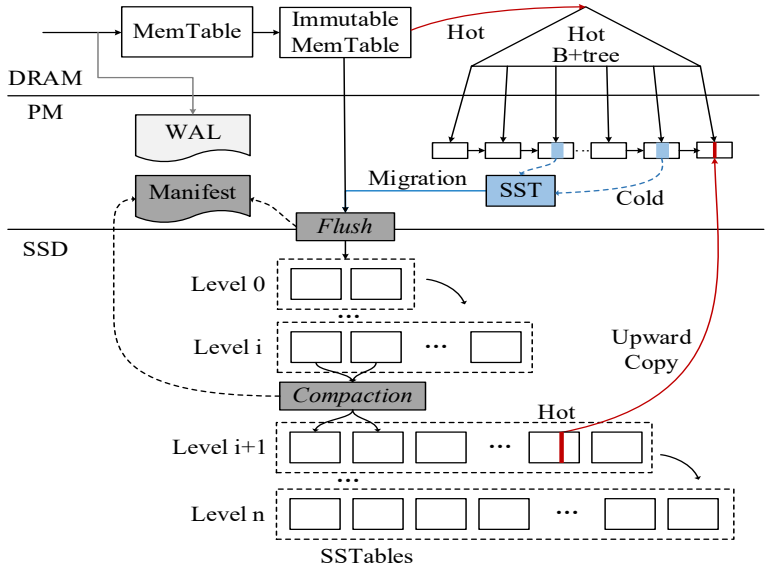
Yunlin Tan，Chaoshu Yang，Runyu Zhang，Mingjie Li，

Pengpeng Tian ， and Xianyu He

Guizhou University

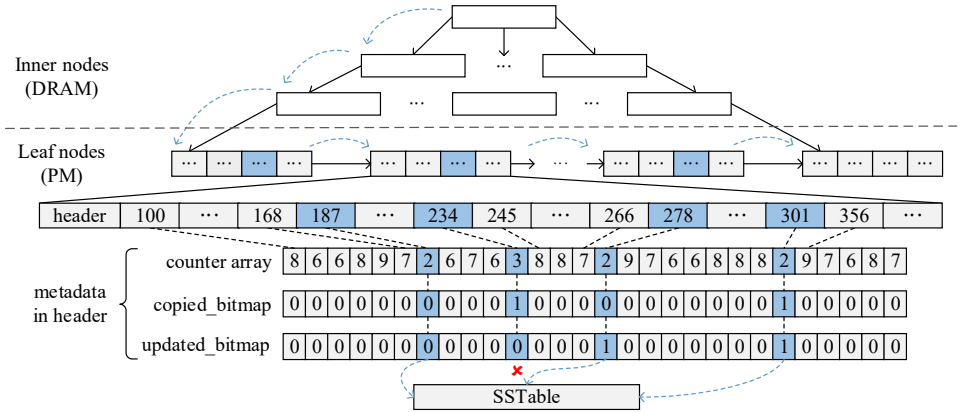
Design Overview

The structure of HADB

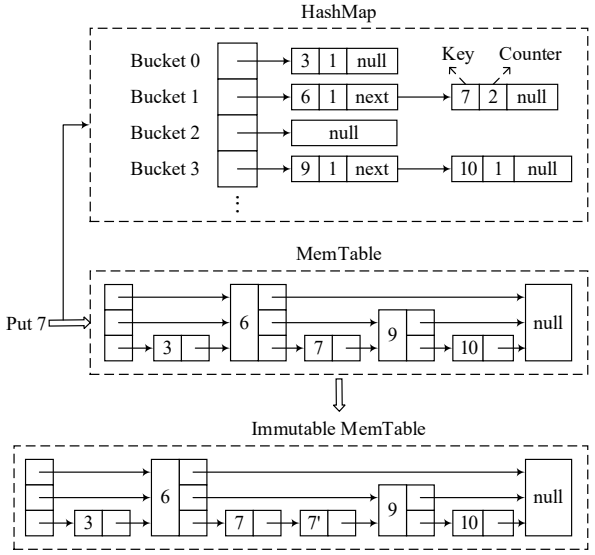


Based on LevelDB, HADB use PM as a middle layer for directly storing the hot data.

Design1: Adaptive Cold Data Migration



Design2: Time-Aware Recognition of Hot Data



(a) HashMap Access Frequency Statistics

Algorithm 1: Identify hot data

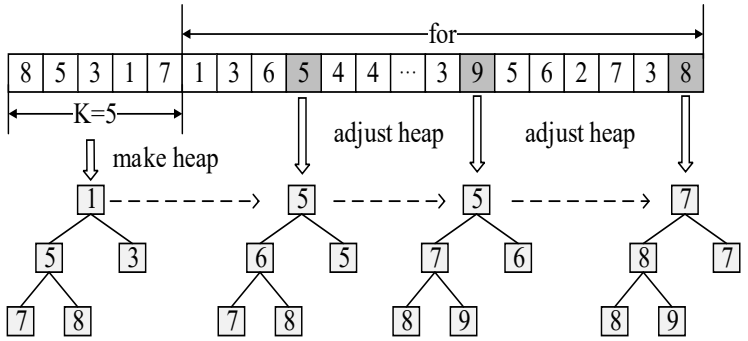
Input : index (the B$^+$-tree index); iter (the iterator of Immutable Memtable); m_map (an unordered_map to record the number of read/write in the cycle from Memtable to immutable Memtable).

Output: hot data are inserted into B$^+$-tree.

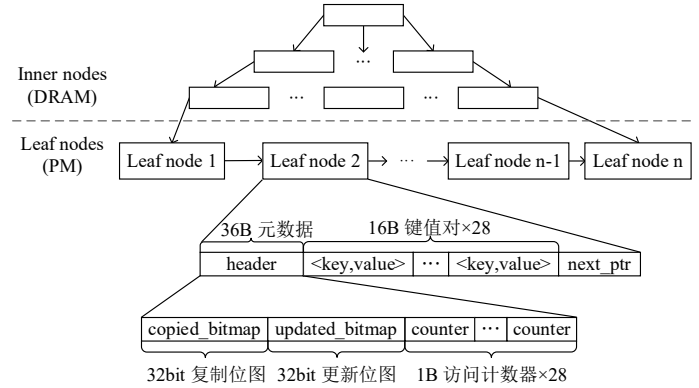
```
1  builder = new TableBuilder;
2  while iter→Valid() do
3      key = iter→key();
4      if index→Get(key) or m_map(key) >= n then
5          index→Insert(key,value);
6      else
7          builder→Add(key,value);
8      end
9  end
10 m_map.clear();
11 builder→Finish();
```

(b) Algorithm for Identifying Hot Data

Design3: Fine-grained Data Migration



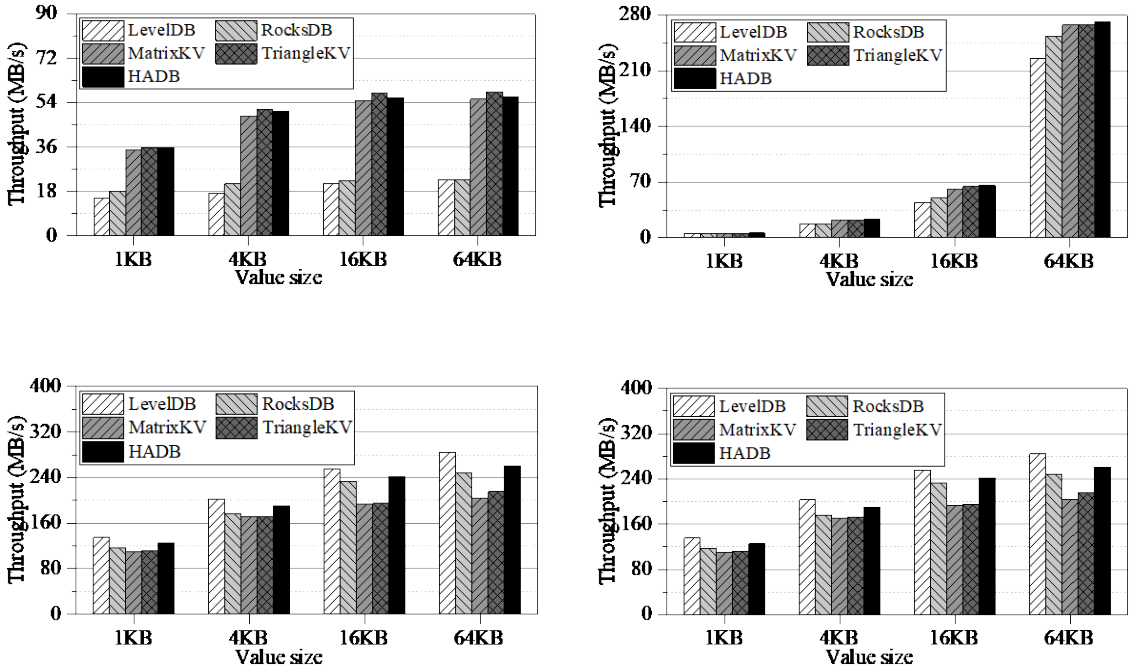
(c) Constructing a Min-Heap Demonstration Diagram



(d) Structure of B+ Tree Leaf Node

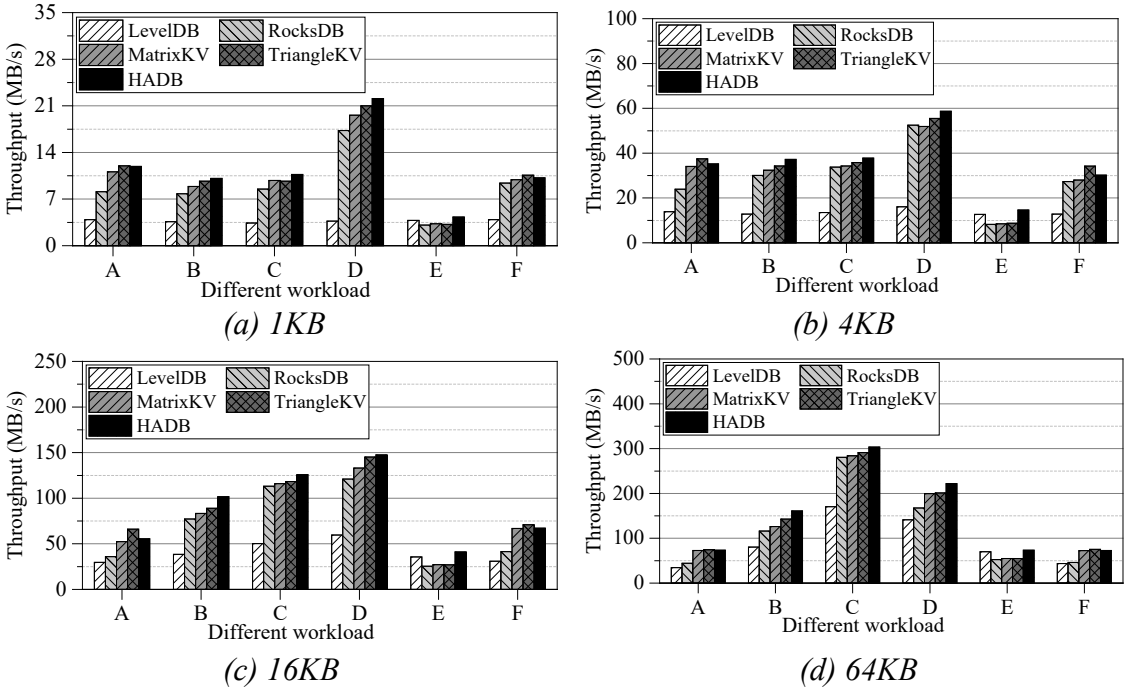
Evaluation

Performance of db_bench Benchmark



- **Random write performance.** The throughput of HADB is **1.36** to **1.91** times better than LevelDB and **1.39** to **1.98** times better than RocksDB.
- **Random read performance.** In terms of random read performance, HADB improves by an average of **15.68%** to **49.32%** compared to LevelDB, and by an average of **9.26%** to **38.95%** compared to RocksDB.

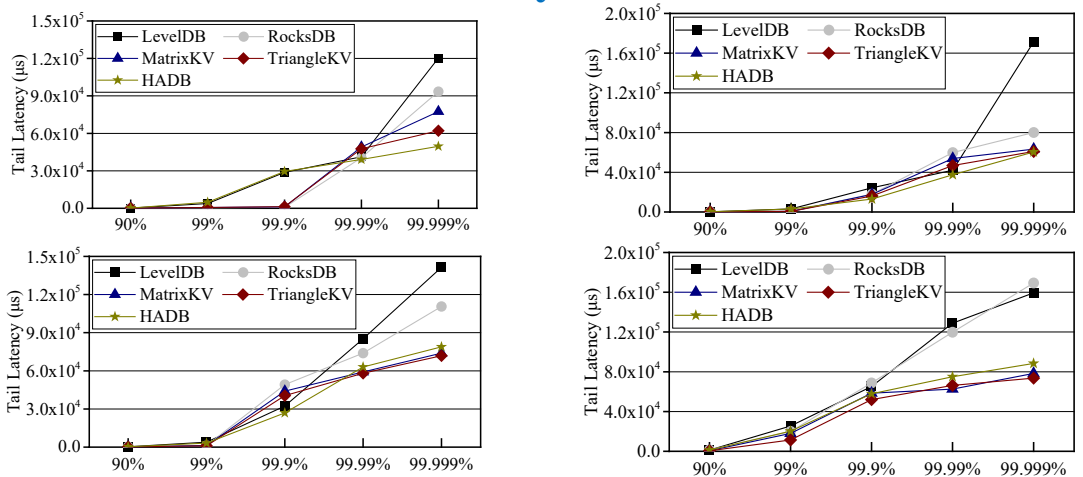
Performance of Mixed Workload



(a) 1KB *(b) 4KB*

(c) 16KB *(d) 64KB*

Performance of Tail Latency



ScalaAFA: Constructing User-Space All-Flash Array Engine with Holistic Designs

Shushu Yi, Xiurui Pan, Qiao Li, Qiang Li,
Chenxi Wang, Bo Mao, Myoungsoo Jung, Jie Zhang

CHASELab
Computer Hardware And System Evolution Laboratory

PEKING UNIVERSITY

Alibaba

KAIST

基于软硬协同设计的用户态全闪存阵列引擎

问题挑战：

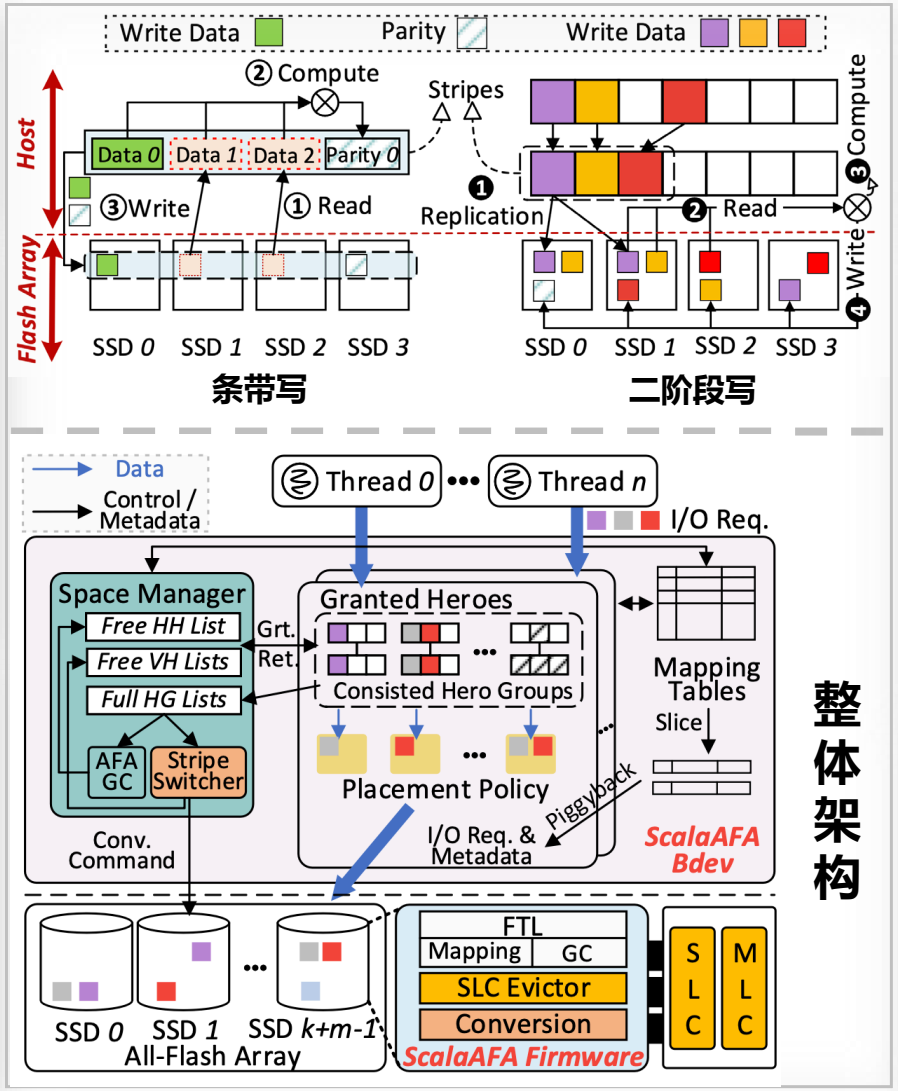
- SATA SSD 500MB/s → PCIe5 SSD 13GB/s
- 现有的闪存阵列引擎还停留在SATA时代
- 无法适配下一代高性能存储设备，发挥全部性能

现有研究：

- **mdraid**：Linux内核中默认的RAID引擎，使用host CPU并行计算校验，以提高吞吐
- **ScalaRAID**：使用更细粒度的锁，减少mdraid中的同步开销，提高scalability
- **FusionRAID**：二阶段写技术，使用复制（RAID10）作为条带化（RAID5/6）的前置，加速写处理

初步分析：

- **复制阶段**：只能达到理想性能的55.4%
  - 用户态到内核态的频繁上下文切换
  - 基于大量锁实现的线程间同步
- **条带化阶段**：性能下降88.5%
  - 校验码生成产生大量背景I/O
- 二阶段写的**内在问题**
  - 复制带来的写放大→缩短SSD寿命



解决思路：

- 用户态无锁的多线程并行I/O处理
- 利用SSD内部资源进行计算卸载 & 减少写放大

关键技术：

- 使用SPDK-comptible的消息传递机制代替锁机制管理存储空间写权限+批处理避免瓶颈
- 优化复制阶段的数据布局，使条带化阶段的计算卸载可以原地进行，无需额外的I/O来收集同条带数据
- 优化SLC buffer驱逐算法，给予复制阶段的副本更低的优先级，避免这些临时数据刷到短寿命的MLC中

实验结果：

- **带宽**：使用1/2线程可以达到4+1/6+1阵列近理想带宽
- **延迟**：写延迟较mdraid降低67.4%
- **写放大**：MLC上flash写减少38.6%
- **应用**：RocksDB+db_bench评估，吞吐提高31.9%

Thanks!

ScalaAFA: Constructing User-Space All-Flash Array Engine with Holistic Designs

https://github.com/ChaseLab-PKU/ScalaAFA

Shushu Yi, Xiurui Pan, Qiao Li, Qiang Li,
Chenxi Wang, Bo Mao, Myoungsoo Jung, Jie Zhang

CHASELab
Computer Hardware And System Evolution Laboratory

PEKING UNIVERSITY

Alibaba

KAIST

澳門大學
UNIVERSIDADE DE MACAU
UNIVERSITY OF MACAU

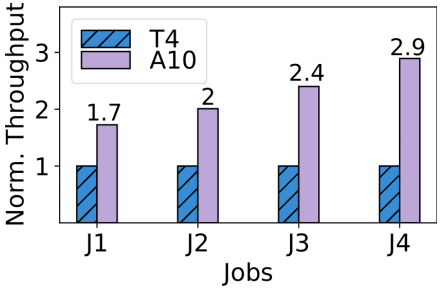
1981 2021 40
不惑新航
揚帆追夢
SET SAIL ANEW ON
THE RUBY JUBILEE

Heet: Accelerating Elastic Training in Heterogeneous Deep Learning Clusters

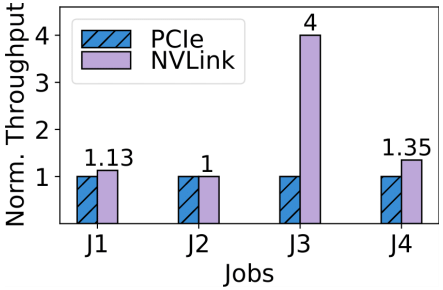
Zizhao Mo, Huanle Xu, Chengzhong Xu
University of Macau
Macau SAR, China

Heterogeneity in DL clusters

- Two critical types of heterogeneity found in today's production clusters: computation and communication heterogeneity.
- Two GPU heterogeneities diversely accelerate DL training jobs.

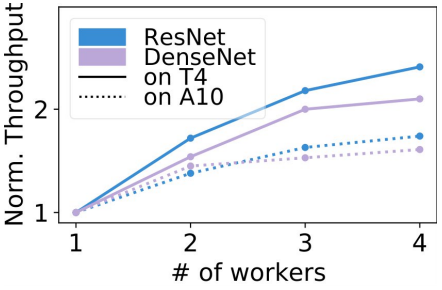


- Diverse acceleration under computation heterogeneity.

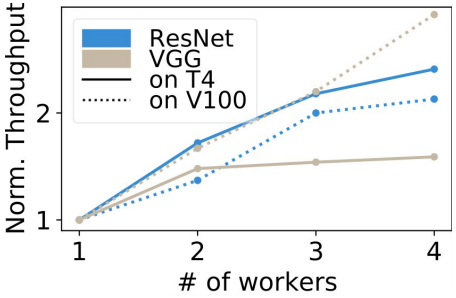


- Diverse acceleration under communication heterogeneity.

- Heterogeneities results in diverse scaling efficiency on different GPUs.



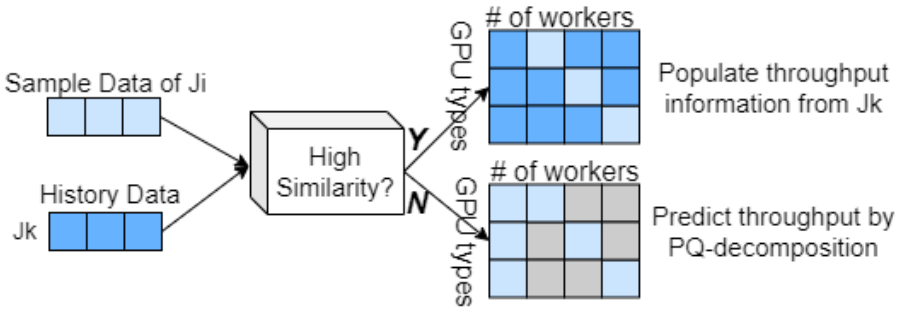
- Resnet prefers T4 (low-end).
- DenseNet prefers T4 (low-end).



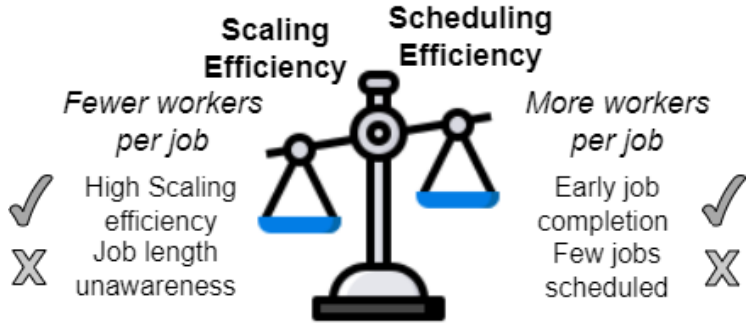
- Resnet prefers T4 (low-end).
- VGG prefers V100 (high-end).

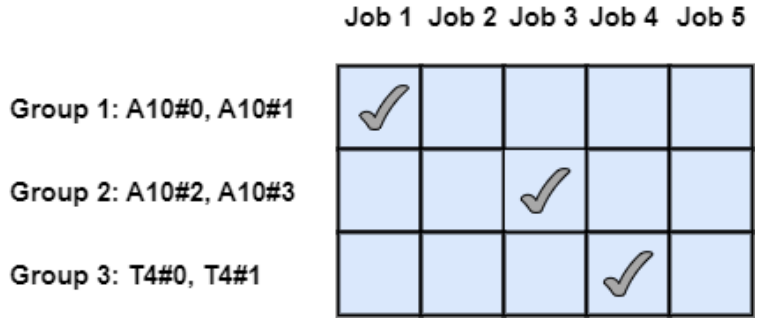
Our solution: Heet

- Fast and accurate profiling.
  - Require information from GPU types, job types, and # of workers
  - Adopt collarborative filtering-based techniques



- Co-optimize scaling and scheduling efficiency (i.e., short-job-first).
  - Delicate allocation price design
  - Restrict cross-network placement and straggler effect
  - Bipartite matching-based optimization





Improving Graph Compression for Efficient Resource-Constrained Graph Analytics

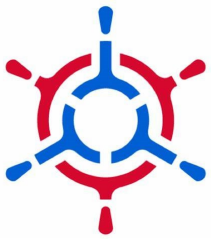
VLDB 2024

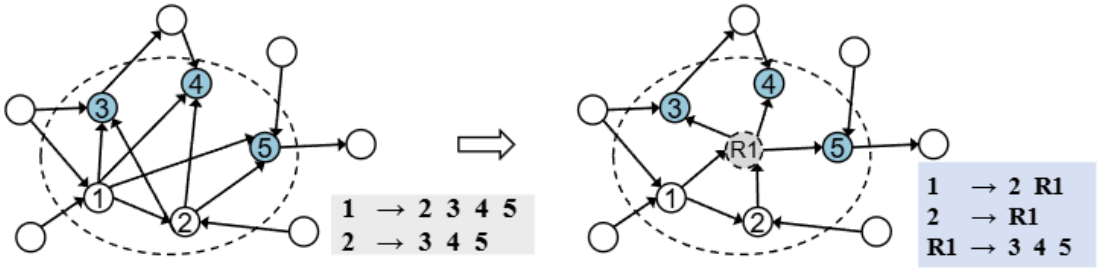
Qian Xu[◇], Juan Yang[★], Feng Zhang[◇], Zheng Chen[◇], Jiawei Guan[◇], Kang Chen[+], Ju Fan[◇], Youren Shen[★], Ke Yang[★], Yu Zhang[◇], Xiaoyong Du[◇]

[◇]Renmin University of China, [+]Tsinghua University, [★]Beijing HaiZhi XingTu Technology Co., Ltd

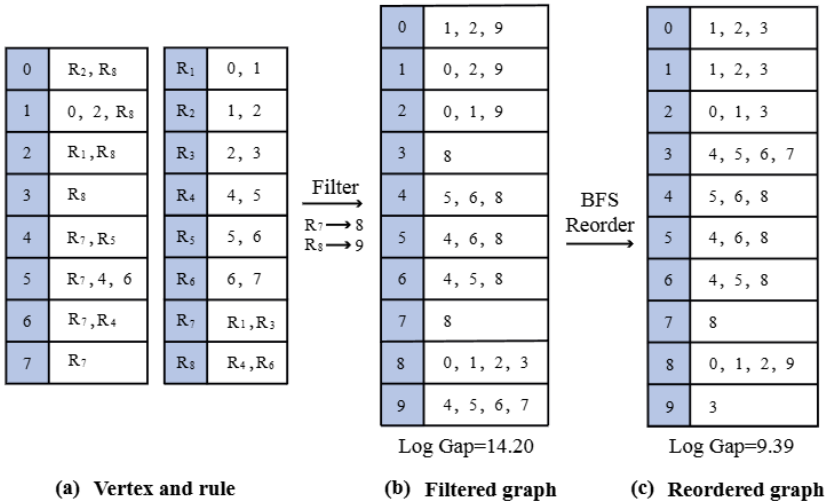
{xuqian,fengzhang,chenzheng123,guanjw,fanj,yu-zhang21,duyong}@ruc.edu.cn,

{yangjuan,shenyouren,yangke}@stargraph.cn, chenkang@tsinghua.edu.cn

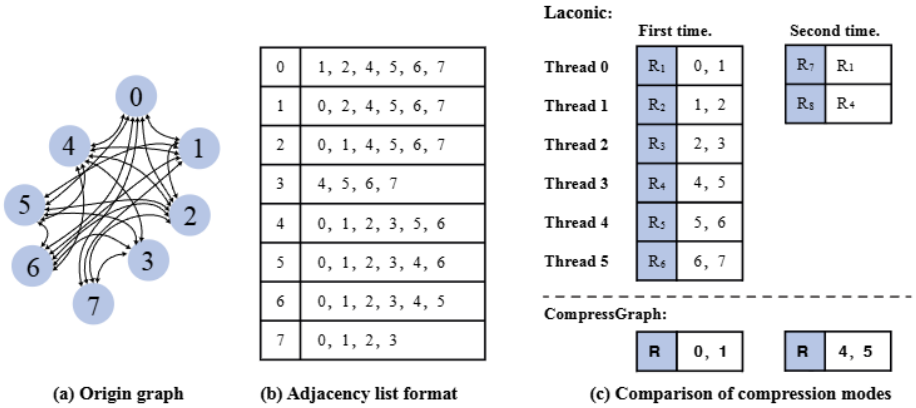




Rule Compression In Graph



Increase Compression Ratio



Rule Compression Speedup

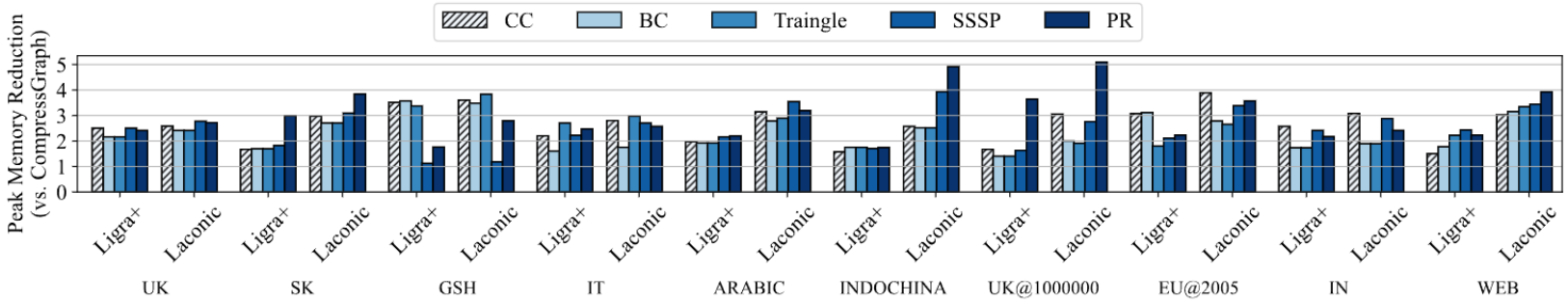


Figure 7: Peak memory reduction.

Table 4: Compression time evaluation. PFT is short for *parallel frequency threshold.*

Dataset	Ligra+ (s)	CompressGraph (s)	Laconic (s) - PFT = 5				Laconic (s) - PFT = 4			
			Iteration 1	Iteration 2	Iteration 3	Total	Iteration 1	Iteration 2	Iteration 3	Total
UK	605.03	690.43	81.67	66.82	42.13	190.62	82.51	67.42	44.58	194.51
SK	326.81	334.71	23.61	22.14	22.43	68.19	24.51	23.41	21.94	69.87
GSH	324.03	812.63	61.70	62.88	59.44	183.03	61.17	61.66	56.46	179.31
IT	175.25	198.43	15.56	14.55	13.96	44.08	15.40	13.98	12.76	42.14
ARABIC	53.54	105.34	5.08	3.62	2.88	11.58	5.11	2.80	3.78	11.70
INDOCHINA	28.50	51.66	7.62	7.08	7.02	22.73	7.62	7.79	6.70	22.12
UK@1000000	3.38	4.07	0.37	0.33	0.29	1.00	0.40	0.31	0.32	1.03
EU@2005	1.57	2.39	0.22	0.21	0.21	0.66	0.24	0.22	0.24	0.70
IN	1.25	2.23	0.19	0.18	0.16	0.54	0.20	0.18	0.19	0.58
BERKSTAN	0.97	1.51	0.11	0.10	0.10	0.31	0.12	0.11	0.09	0.32

Table 3: Comparison of compression ratios and compressed graph sizes of Ligra+, CompressGraph and Laconic.

Dataset	Original Size	Ligra+ Compressed Size	Ligra+ Compression Ratio	CompressGraph Compressed Size	CompressGraph Compression Ratio	Laconic Compressed Size	Laconic Compression Ratio
UK@2014	360.63GB	-	-	-	-	23.89GB	16.30
CLUE	324.49GB	-	-	-	-	38.99GB	8.32
UK	28.65GB	5.44GB	5.26	4.48GB	6.38	3.48GB	8.23
SK	14.90GB	5.04GB	2.95	2.62GB	5.68	1.85GB	8.05
GSH	13.94GB	5.93GB	2.35	6.47GB	2.15	4.76GB	2.93
IT	8.88GB	3.04GB	2.92	1.72GB	5.16	1.22GB	7.29
ARABIC	4.93GB	1.71GB	2.88	0.96GB	5.12	699MB	7.22
INDOCHINA	1.50GB	525MB	2.92	295MB	5.20	216MB	7.10
UK@1000000	322MB	106MB	3.03	47MB	6.76	32MB	9.88
EU@2005	153MB	61MB	2.50	43MB	3.52	32MB	4.69
IN	153MB	58MB	2.63	48MB	3.16	35MB	4.32
BERKSTAN	63MB	30MB	2.10	26MB	2.43	20MB	3.15

RUTGERS SAMSUNG igalia

OmniCache: Collaborative Caching for Near-storage Accelerators

Jian Zhang, Yujie Ren, Marie Nguyen, Changwoo Min, Sudarsun Kannan

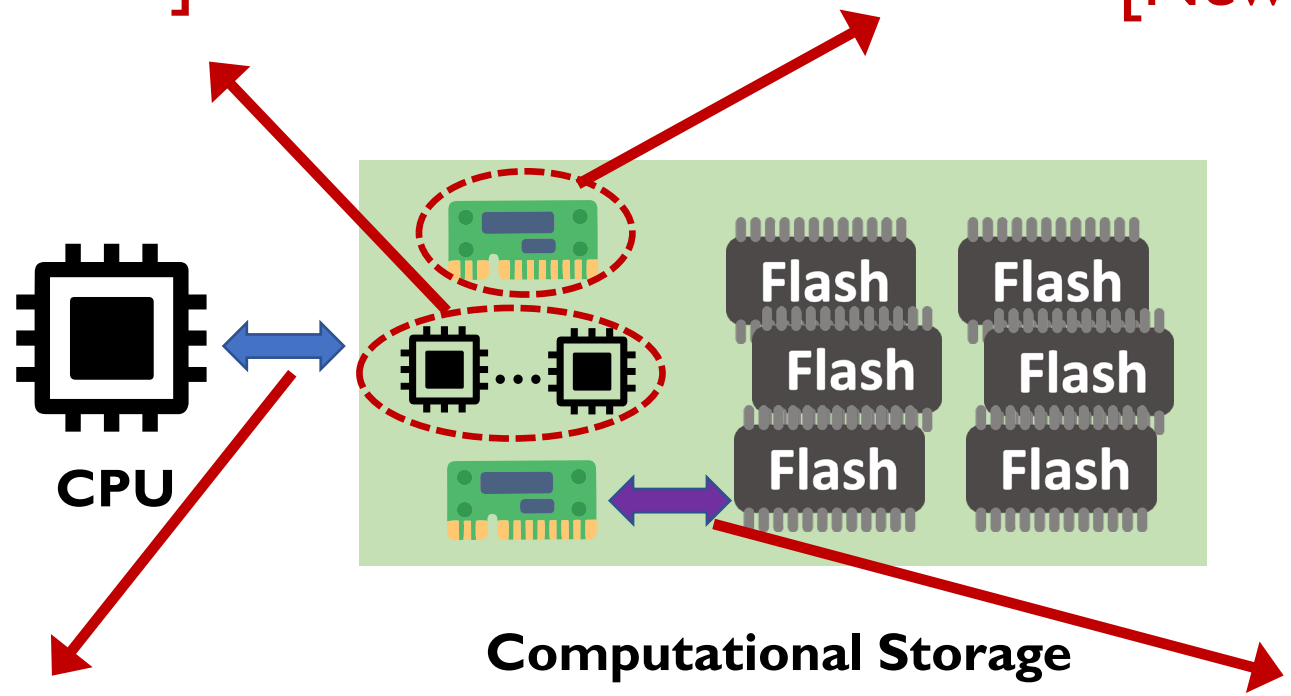
Hardware Trends

Multi-core compute (4-16 cores)

[ARM CSDs]

DRAM size (4-16GB)

[Newport]



Fast remote memory access with CXL

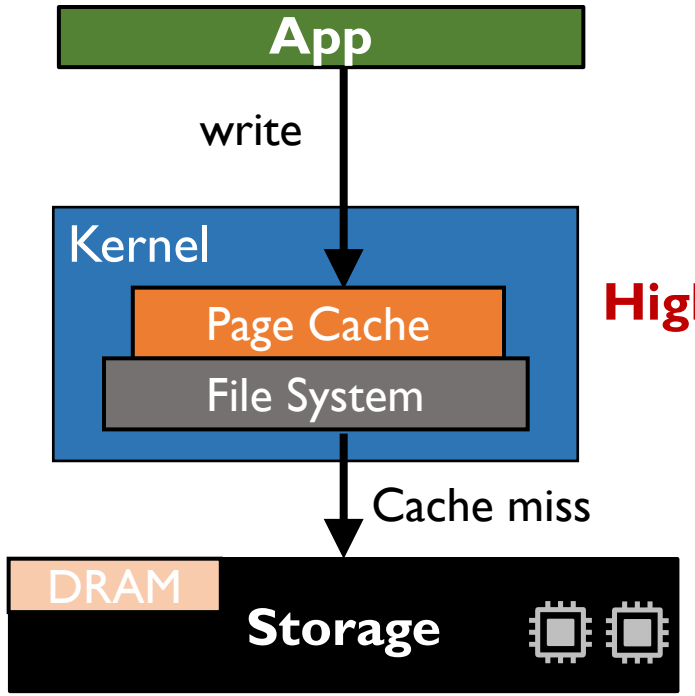
[Samsung Memory-Semantic SSD]

High speed interconnect

[ScaleFlux]

Failure to Exploit Near-storage Memory

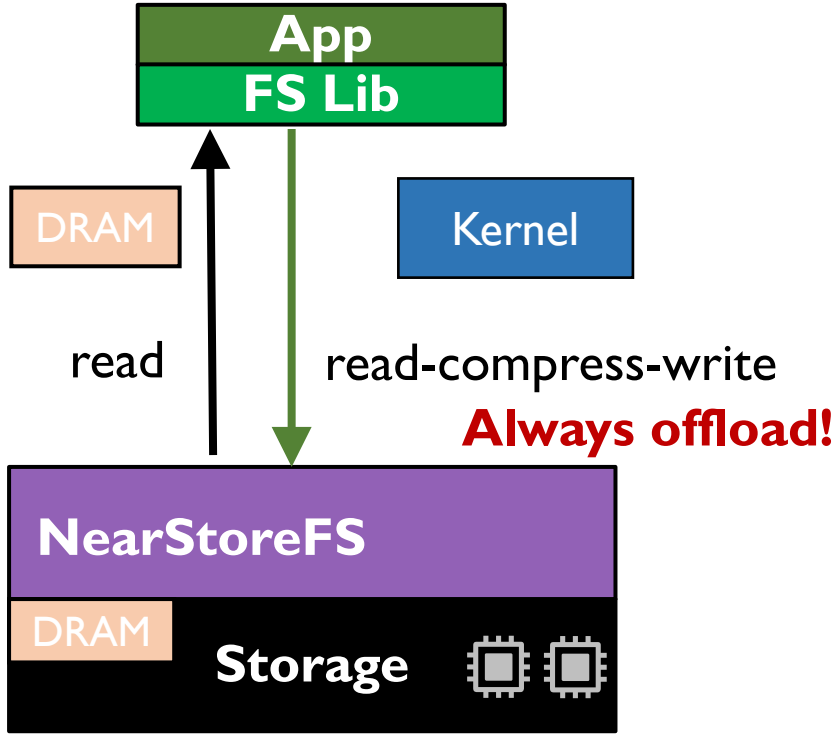
Near-storage Device with Host FS
(PolarDB [FAST '20], λ-IO [FAST '23], etc.)



High kernel overhead

Lack of support to use device memory

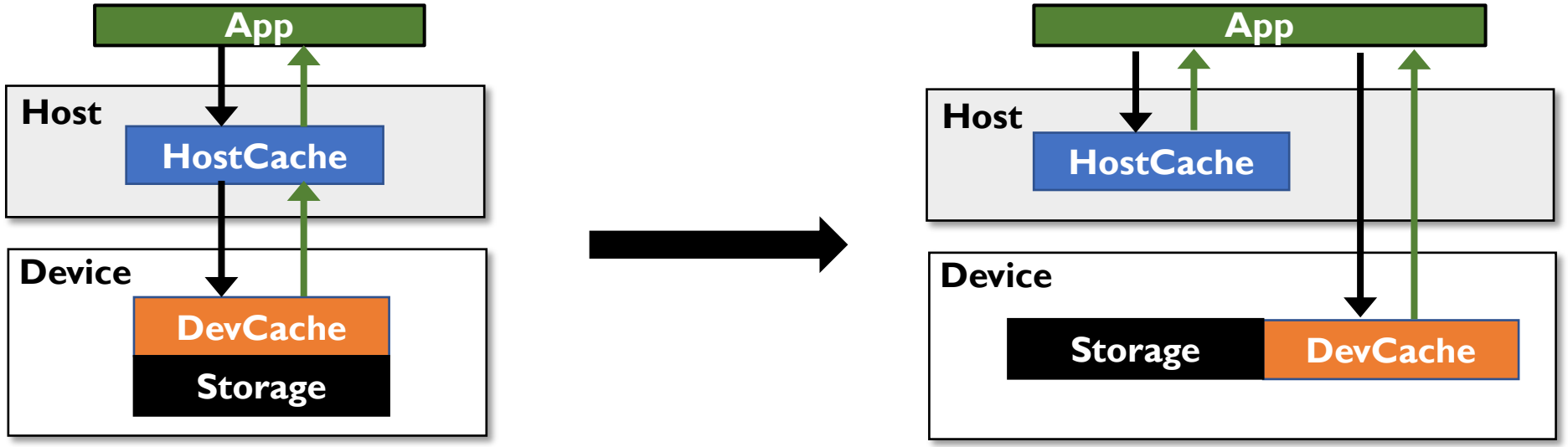
Device FS
(DevFS [FAST '18], FusionFS [FAST '22])



Failure to utilize host and device memory

Our Solution: OmniCache

A **horizontal caching** design to exploit the **combined capabilities** of near-storage, host compute, and their memory resources to accelerate I/O and data processing



Vertical Caching **Horizontal Caching**

More details in the paper!

PolarDB-SCC: A Cloud-Native Database Ensuring Low Latency for Strongly Consistent Reads

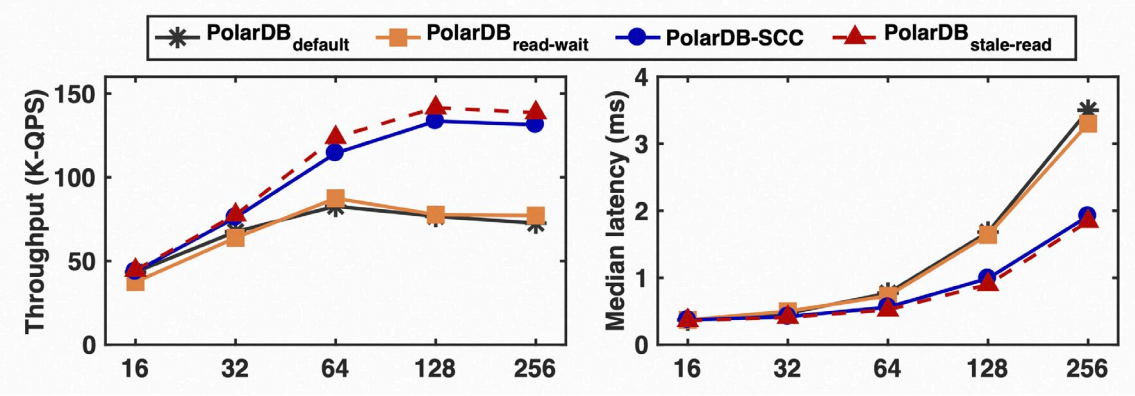
Xinjun Yang, Yingqiang Zhang, ***Hao Chen***,
Chuan Sun, Feifei Li, Wenchao Zhou

Alibaba Group

Strongly Consistent Read on Secondary Nodes

- **Problem:** eventual consistency on secondary nodes
  - Typically applying log on secondary nodes is asynchronous
  - Secondary nodes may return stale data (***eventual consistency***)

- **Solution:** PolarDB Strong Consistency Cluster (PolarDB-SCC)
  - Track primary node's modification timestamp at three progressively fine-grained levels
  - Design linear Lamport timestamp to reduce timestamp fetching operation
  - Highly co-design with one-sided RDMA interface



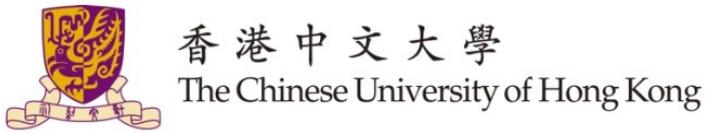
- PolarDB-SCC has ***nearly identical performance with stale-read*** policy
- PolarDB-SCC is the ***first*** cloud-native database that supports strongly consistent read with negligible overhead.

OSDI 23 ChinaSys

SMART: A High-Performance Adaptive Radix Tree for Disaggregated Memory

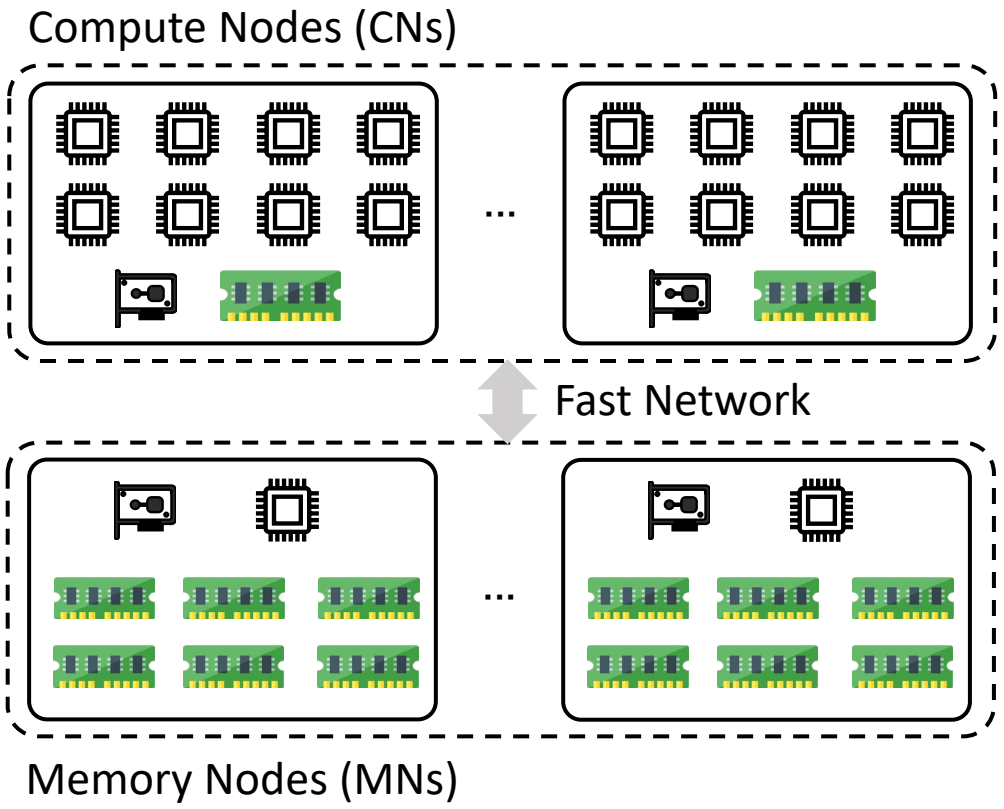
Xuchuan Luo[1], Pengfei Zuo[2], Jiacheng Shen[3], Jiazhen Gu[3],
Xin Wang[1,4], Michael R. Lyu[3], and Yangfan Zhou[1,4]

[1]School of Computer Science, Fudan University
[2]Huawei Cloud *[3]The Chinese University of Hong Kong*
[4]Shanghai Key Laboratory of Intelligent Information Processing, Shanghai, China

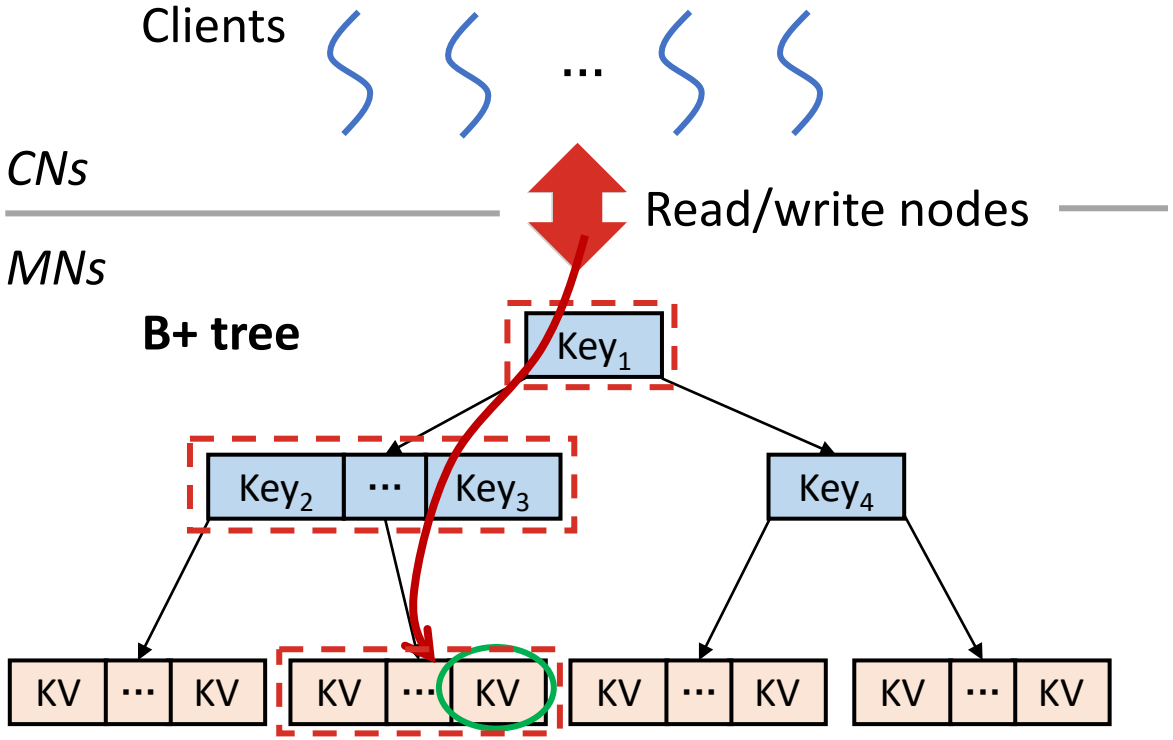


Motivation: B+ Trees are bandwidth-bound on disaggregated memory

Disaggregated Memory (DM)



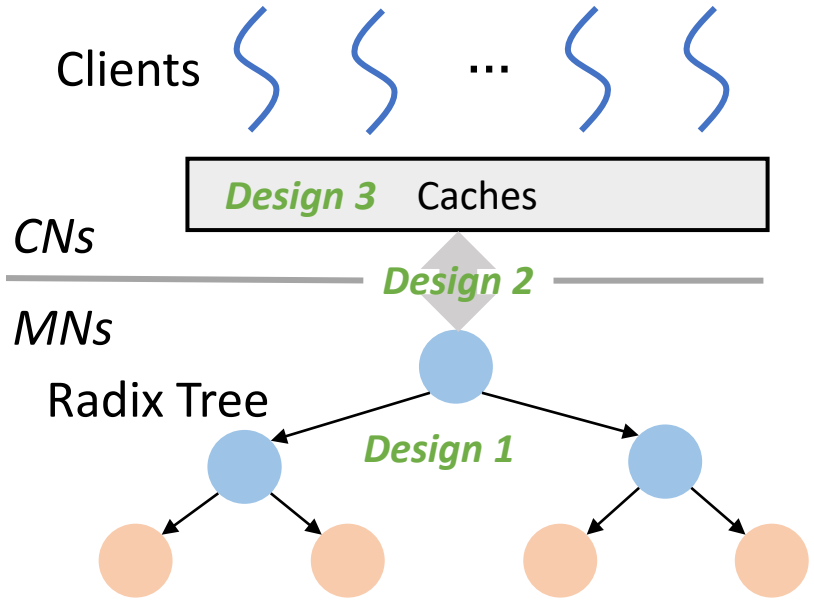
Existing Tree Indexes on DM



☹ **Problem:** Read and write amplifications of B+ trees

➔ Exacerbate the network bandwidth bottleneck of DM

Idea: Using radix trees to build high-performance tree indexes on DM



Challenges and Solutions

☹ **Expensive concurrency control**

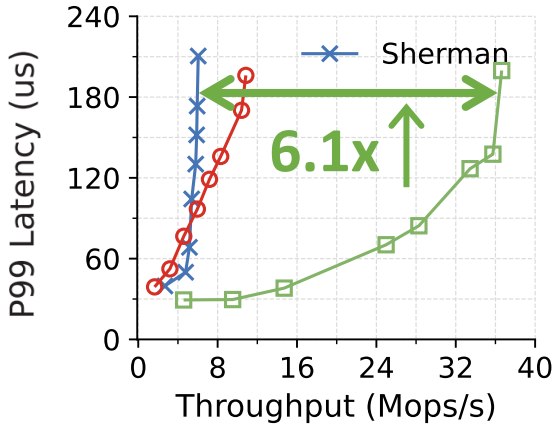
➢ *Design 1: Hybrid concurrency control*

☹ **Bounded memory-side IOPS**

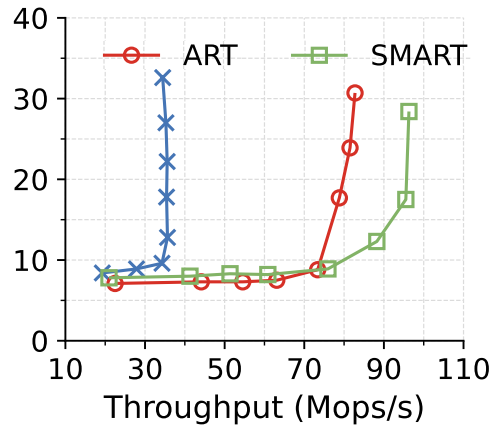
➢ *Design 2: Read delegation and write combining*

☹ **Complicated cache validation**

➢ *Design 3: Reverse check mechanism*



Write-intensive



Read-intensive

☺ **Results:** SMART outperforms the state-of-the-art B+ tree on DM by up to **6.1x**

https://github.com/dmemsys/SMART

xcluo23@m.fudan.edu.cn

Thank you!

SODA: A Set of Fast Oblivious Algorithms in Distributed Secure Data Analytics

Xiang Li[1], Nuozhou Sun[1], Yunqian Luo[1], Mingyu Gao[1,2]

1 Tsinghua University

2 Shanghai Qi Zhi Institute

清華大學 交叉信息研究院
Institute for Interdisciplinary Information Sciences, Tsinghua University

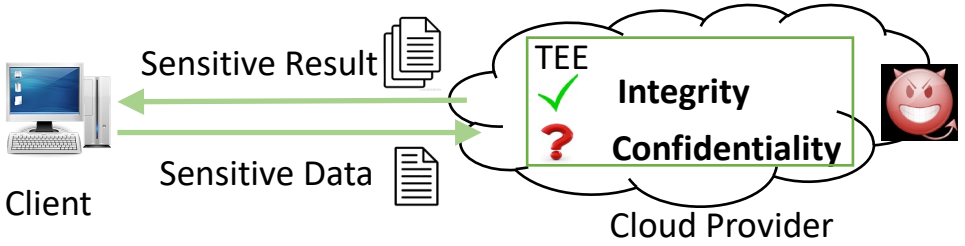
上海期智研究院
SHANGHAI QI ZHI INSTITUTE

清華大學
Tsinghua University

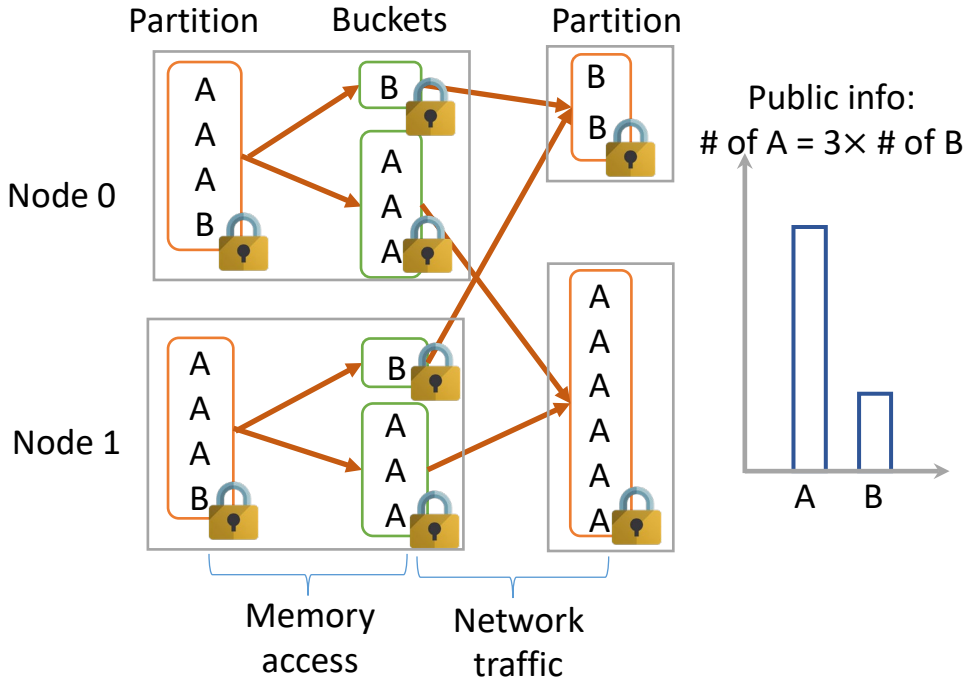
Published at VLDB 2023

TEE Side Channels in Distributed Setting

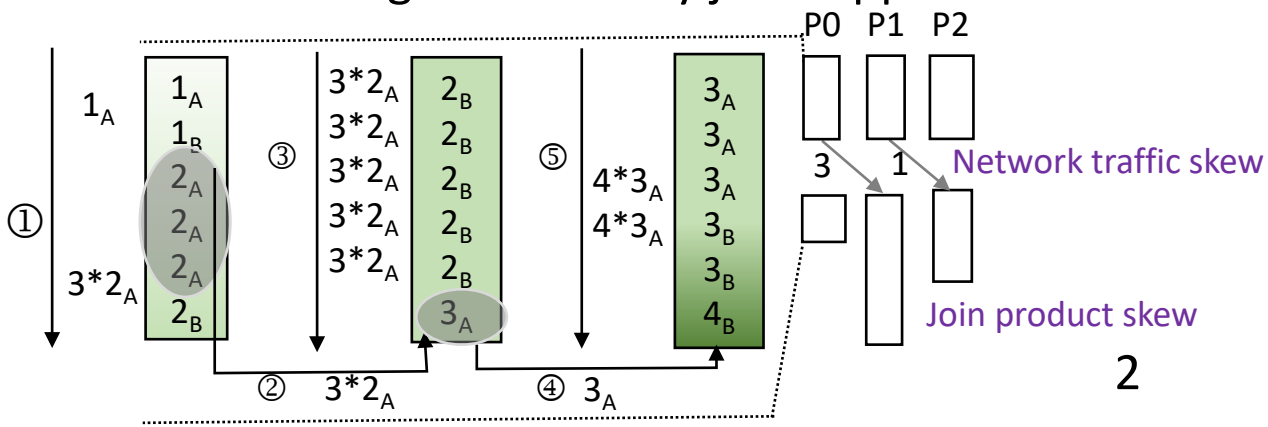
- Trusted execution environment



- Access pattern side channels in distributed setting

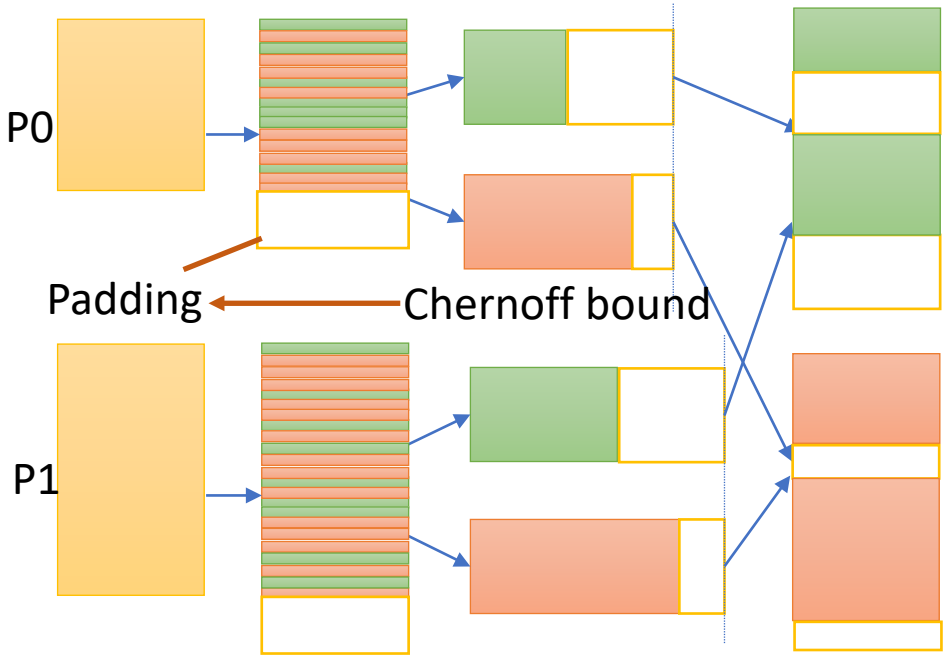


- Oblivious algorithms
  - Access pattern independent of content
  - Allowing input/output size leakage
- SOTA: Opaque, NSDI'17
  - Ops: filter, aggregate, join
  - Key step: oblivious global sort
  - Problems
    - Perf overhead
    - Lack of general binary join support



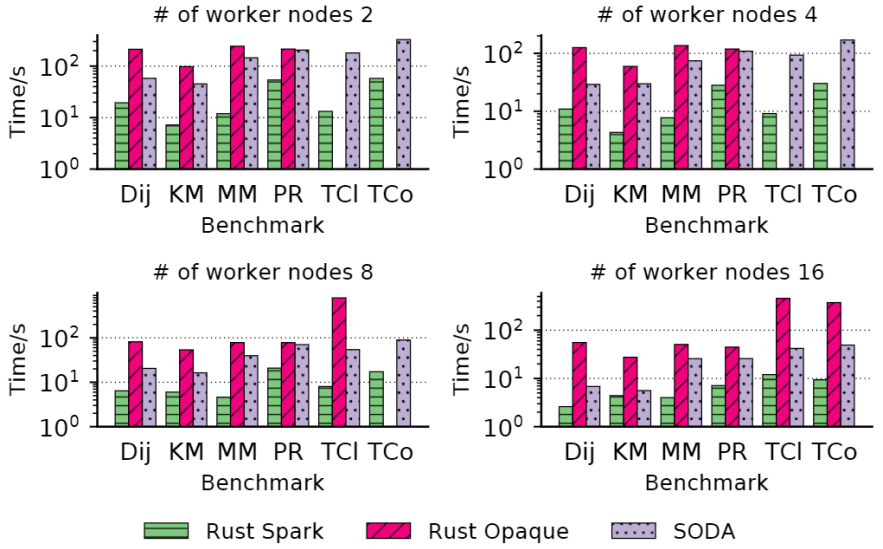
A Set of Distributed Oblivious Algorithms

- SODA design
  - Pseudo-random communication
    - Observation: oblivious network traffic if each node sends the same amount of data to all receiving nodes



    - Two-level assignment for join
      - Determine assignment before data shuffle

- Impl & eval
  - Based on Flare, VLDB'23
  - Individual operators (vs Opaque)
    - Aggregate: 2.6x
    - Filter: 1.8x
    - Binary equi-join: 1.5x-3.7x
  - Evaluation: macro-benchmarks
    - 1.1x-14.6x



46th IEEE/ACM International Conference on Software Engineering

Understanding Transaction Bugs in Database Systems

Ziyu Cui, Wensheng Dou, Yu Gao, Dong Wang, Jiansen Song, Yingying Zheng, Tao Wang, Rui Yang, Kang Xu, Yixin Hu, Jun Wei, Tao Huang

Institute of Software, Chinese Academy of Sciences

University of Chinese Academy of Sciences

Sun Yat-sen University

Transaction Bug (TXBug) Study Methodology

Collect 140 TXBugs from 6 DBMSs

Time range	Keyword	Transaction feature
2018-2022	transaction, isolation level, commit, abort, ...	TXBug contains • at least one explicit transaction • or at least two transactions

MySQL, PostgreSQL, MariaDB Foundation, SQLite, CockroachDB, TiDB

Analyze TXBugs

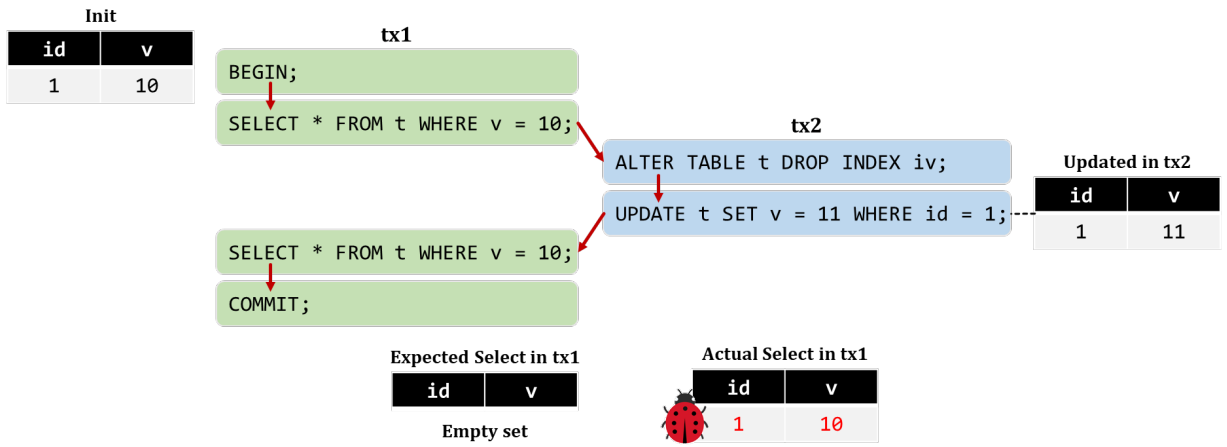
- Issue descriptions
- Embedded test cases
- Developer discussions
- Available fixing patches

- **Bug manifestation**
- **Root cause**
- **Bug impact**
- **Detection capability of existing approaches**

Transaction Bug (TXBug) Study Findings

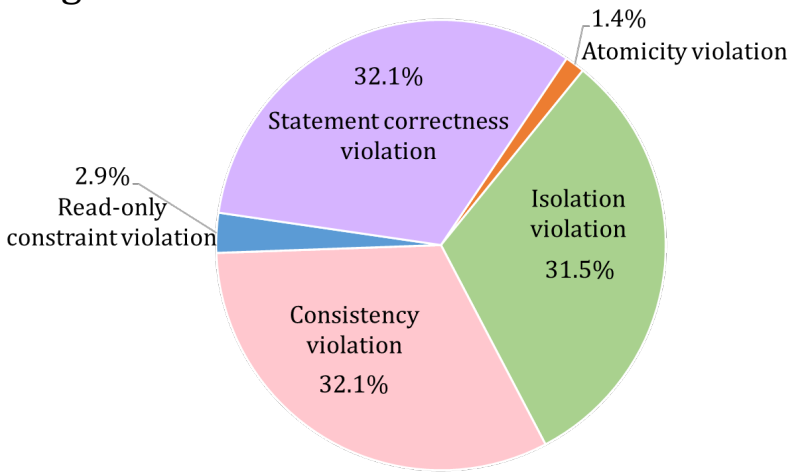
Bug Manifestation

- 94.3% of TXBugs can be triggered deterministically



Root Cause

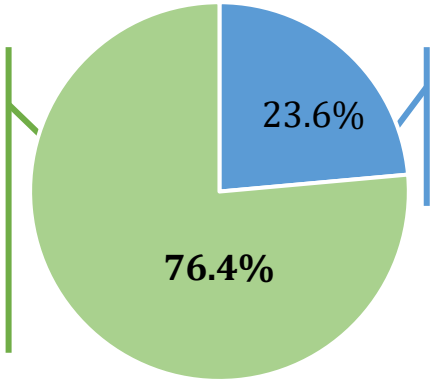
- TXBugs violate five kinds of transaction semantics



Bug Impact

- 76.4% of TXBugs only lead to silent failures

Silent failures
- Incorrect DBMS states
- Incorrect database states
- Incorrect query results
- Performance degradation
- Missing blocking
- Incorrect error reporting



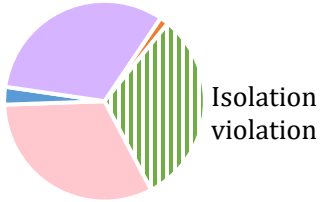
Explicit failures
- DBMS errors
- DBMS unavailability

Detection Capability of Existing Approaches

- Transaction verification approaches[1][2] cannot detect 97.1% of TXBugs

key	value
1	a
2	b

Key-value structure



Root cause

[1] K. Kingsbury et al., Elle: Inferring Isolation Anomalies from Experimental Observations. (VLDB Endow 2020).
[2] C. Tan et al., Cobra: Making Transactional Key-Value Stores Verifiably Serializable. (OSDI 2020).