



华中科技大学

HUAZHONG UNIVERSITY OF SCIENCE AND TECHNOLOGY

大容量SSD数据管理算法设计与实现

AIMS-FTL：一种高性能DFTL架构设计

队名：AIMS小分队

汇报人：崔乘上

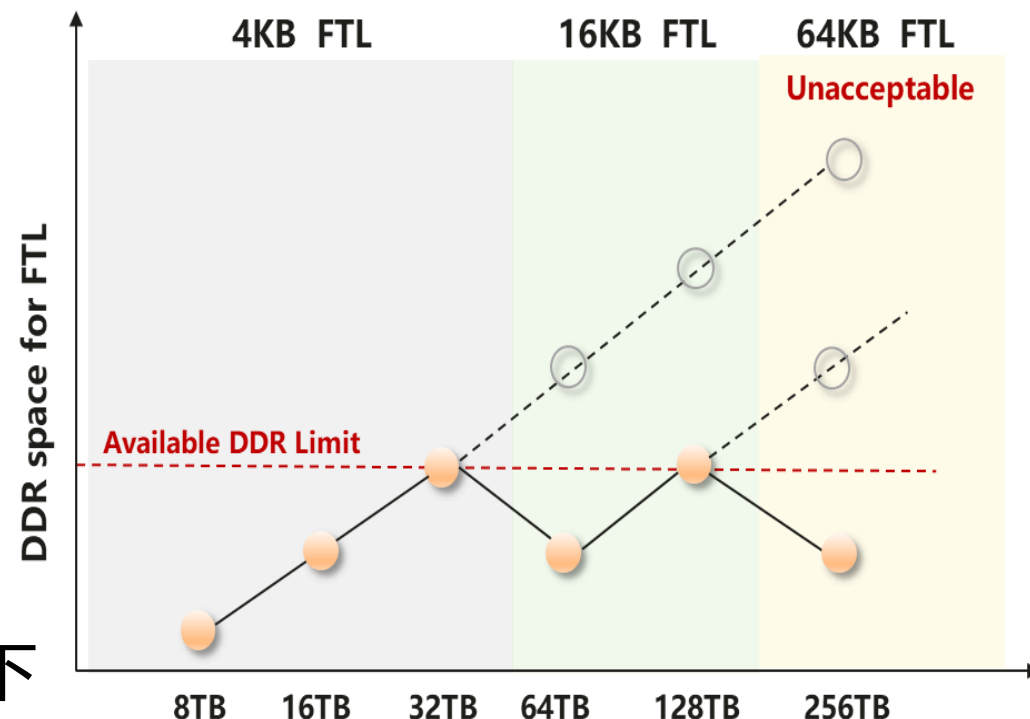
队员：张纳百川 黄可喻 崔乘上 徐裕钧

指导老师：童薇

2026年1月10日

| SSD容量 | FTL粒度 | FTL表项个数 | 内存空间占用 |
|-------|-------|---------|---------------|
| 256TB | 4KB | 64G | 512GB @ 64bit |
| | 16KB | 16G | 128GB @ 64bit |
| | 32KB | 8G | 64GB @ 64bit |
| | 64KB | 4G | 16GB @ 32bit |

- SSD容量扩张，但内存容量有限
- 大粒度IO与小粒度FTL不适配，导致访问性能低下



如何在有限内存容量下，保证小粒度FTL的访问性能？

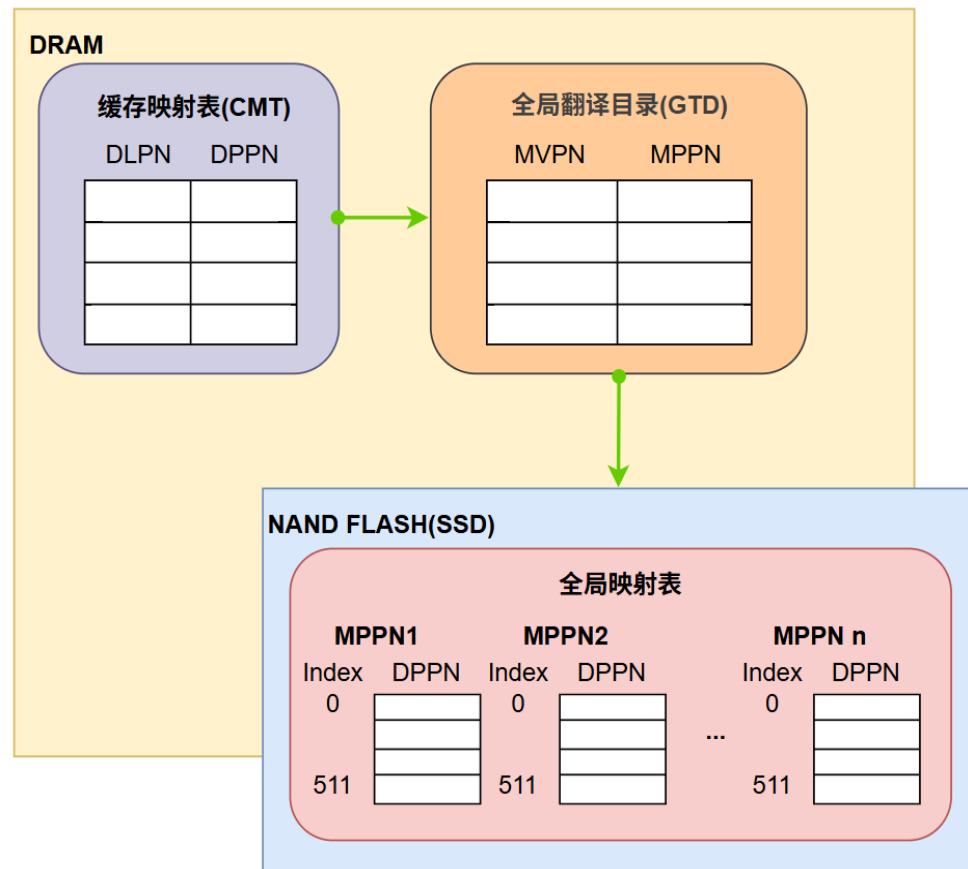
映射粒度

- 块级映射写放大严重，**不适配小粒度FTL需求**
- 混合级映射存在日志块的合并操作，性能抖动剧烈，**不适配高访问性能需求**
- 可变长度映射元数据管理开销复杂，内存需求难以估计，**不适配有限内存容量需求**
- **以DFTL为典型代表的页级映射在内存容量、映射粒度、访问性能三者间达成较好平衡**

| 映射粒度 | 特点 | 代表性方案 |
|---------------|------------------|--------------|
| 页级映射 | 逻辑页映射到物理页 | DFTL |
| 块级映射 | 逻辑块映射到物理块 | NFTL |
| 混合级映射 | 页块级映射粒度混合 | BAST |
| 可变长度映射 | 动态分配映射粒度 | Variable FTL |

DFTL

- 核心思想
 - 闪存存储全局映射表
 - DRAM使用缓存映射表（CMT）存储访问频率高的映射条目，CMT未命中时通过全局翻译目录（GTD）查找映射条目在全局映射表的位置
- 存在的瓶颈
 - **访问时延瓶颈**：CMT对数据局部性支持不足，面对大规模顺序读写场景性能不佳
 - **内存占用瓶颈**：标准 DFTL 的 GTD 需要存储完整的物理页号
 - **延迟抖动与阻塞瓶颈**：DFTL使用传统同步架构，高并发场景下QoS无法保障，Miss时的阻塞会导致吞吐率剧烈波动

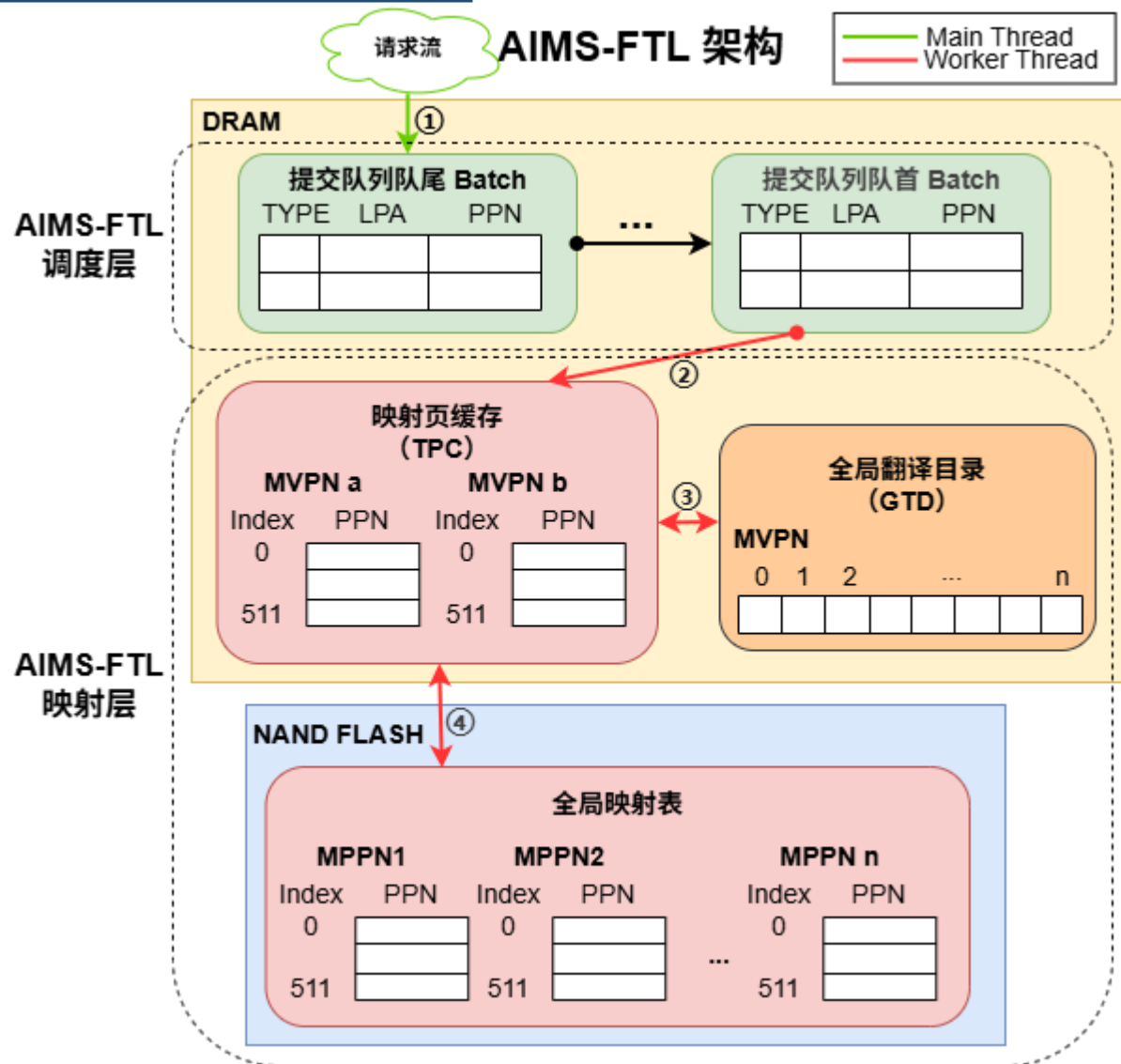


新型 FTL 架构 —— AIMS-FTL

- 传统DFTL设计：
 - 缓存映射表 (CMT)
 - 传统GTD
 - 无调度策略, 传统同步架构



- AIMS-FTL 架构设计:**
 - 映射页缓存 (TPC)
 - 位图压缩GTD
 - SQ队列和提交批处理

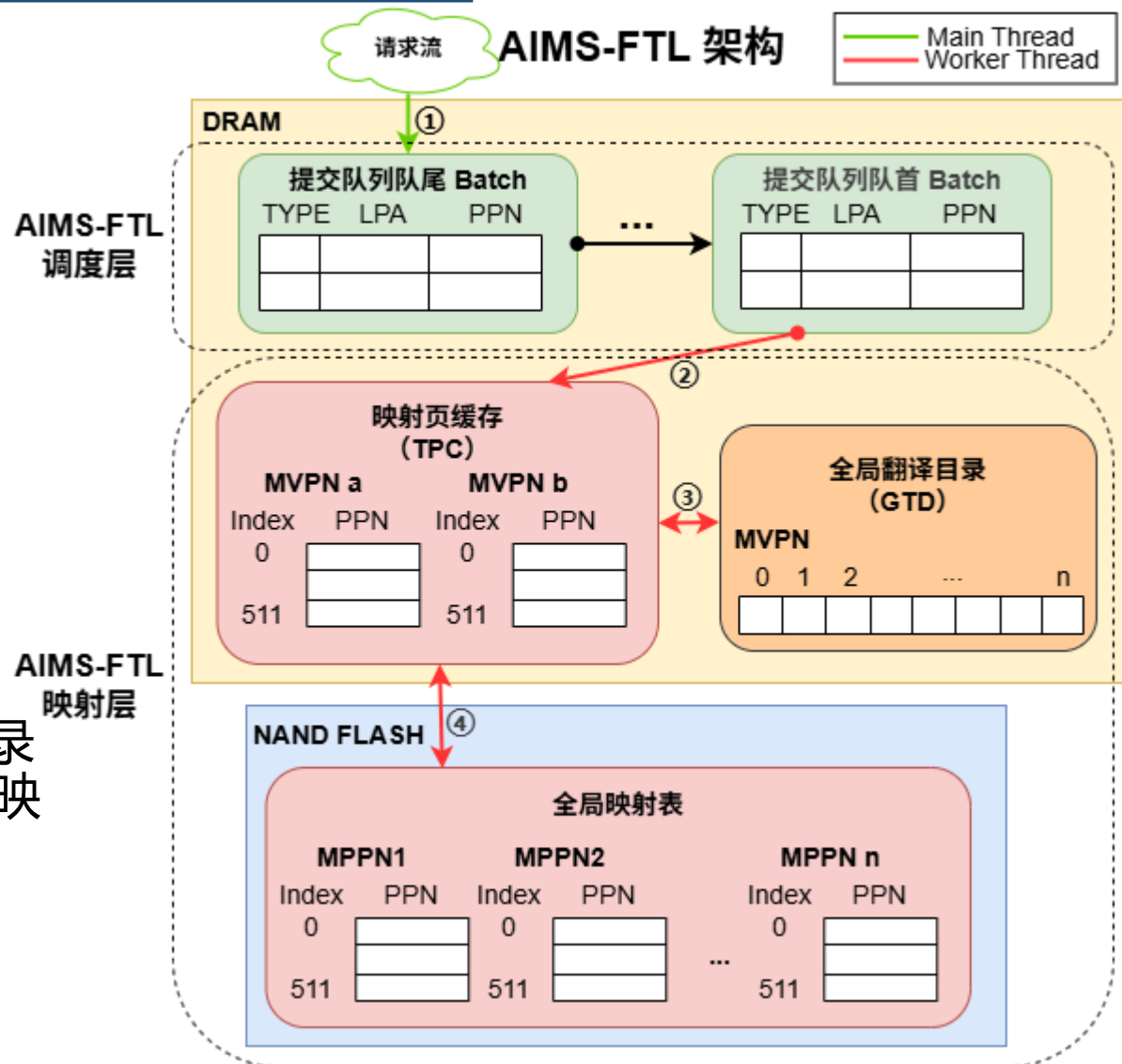


新型 FTL 架构 —— AIMS-FTL

- 面向大容量、低时延需求
- 大规模顺序读写请求

采用 **调度层 + 映射层** 设计：

- 调度层 (Schedule Layer)：包含 提交队列 (Submission Queue, SQ) 和 提交批处理
- 映射层 (Translation Layer)：以 全局映射目录 (GTD) 和 映射页缓存 (TPC) 为核心维护页级映射



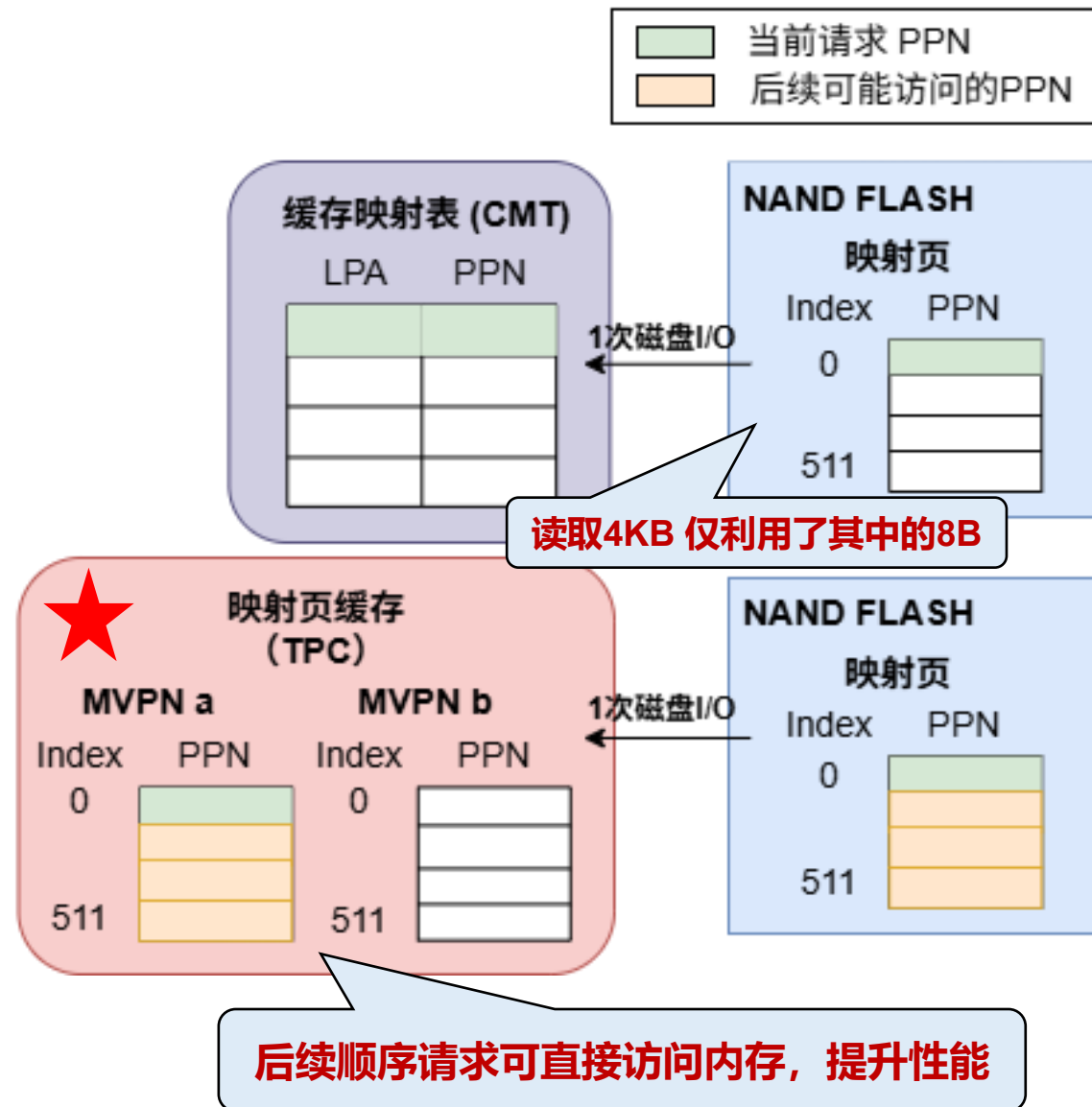
映射页缓存 (TPC)

❌ CMT 架构存在访问局部性 (Locality) 的浪费

○ 引入新设计 —— 映射页缓存 (TPC)

- 在TPC中缓存映射页信息，其中包含512个LBA-PPN条目
 - 访问LBA 0时，可以通过一次磁盘 I/O 将 LBA 0-511 全部存入缓存；写回磁盘时也可以一次写回512个条目
 - 很好地利用数据局部性

**顺序写性能提升 10x 以上
(14.6s vs 197.5s)**



GTD位图压缩

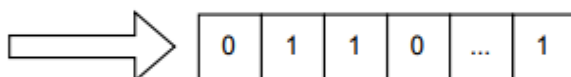
引入新设计 —— GTD 压缩

- 将 GTD 压缩为位图(Bitmap), 每个比特位仅用于标记一个映射页(MVPN)是否已被分配

传统GTD

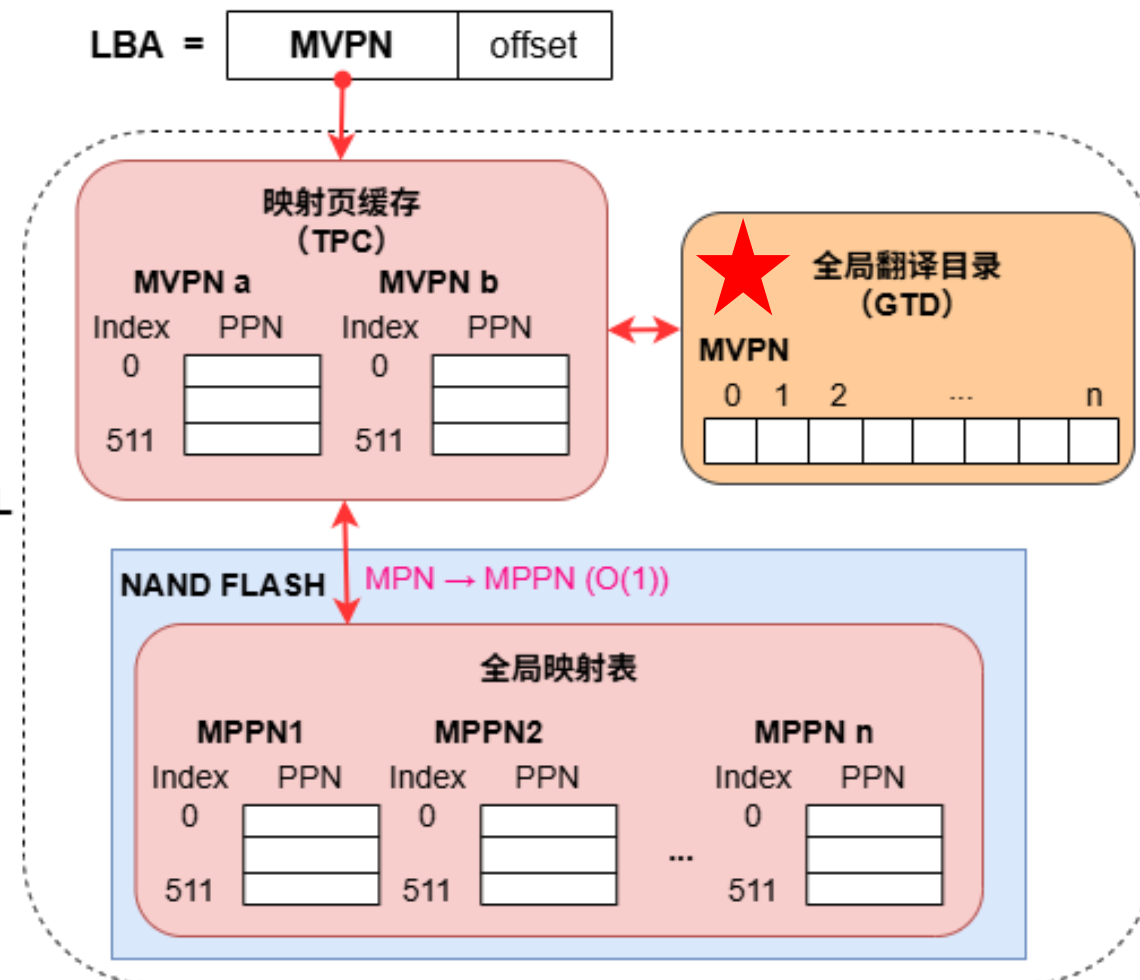
| |
|-------|
| PPN A |
| PPN B |
| PPN C |
| ... |
| PPN X |

位图 GTD



原理：利用大规模顺序写场景下 MVPN和MPPN的线性关系

元数据内存占用降低 64 倍



请求调度策略

引入新设计 —— SQ队列和提交批处理

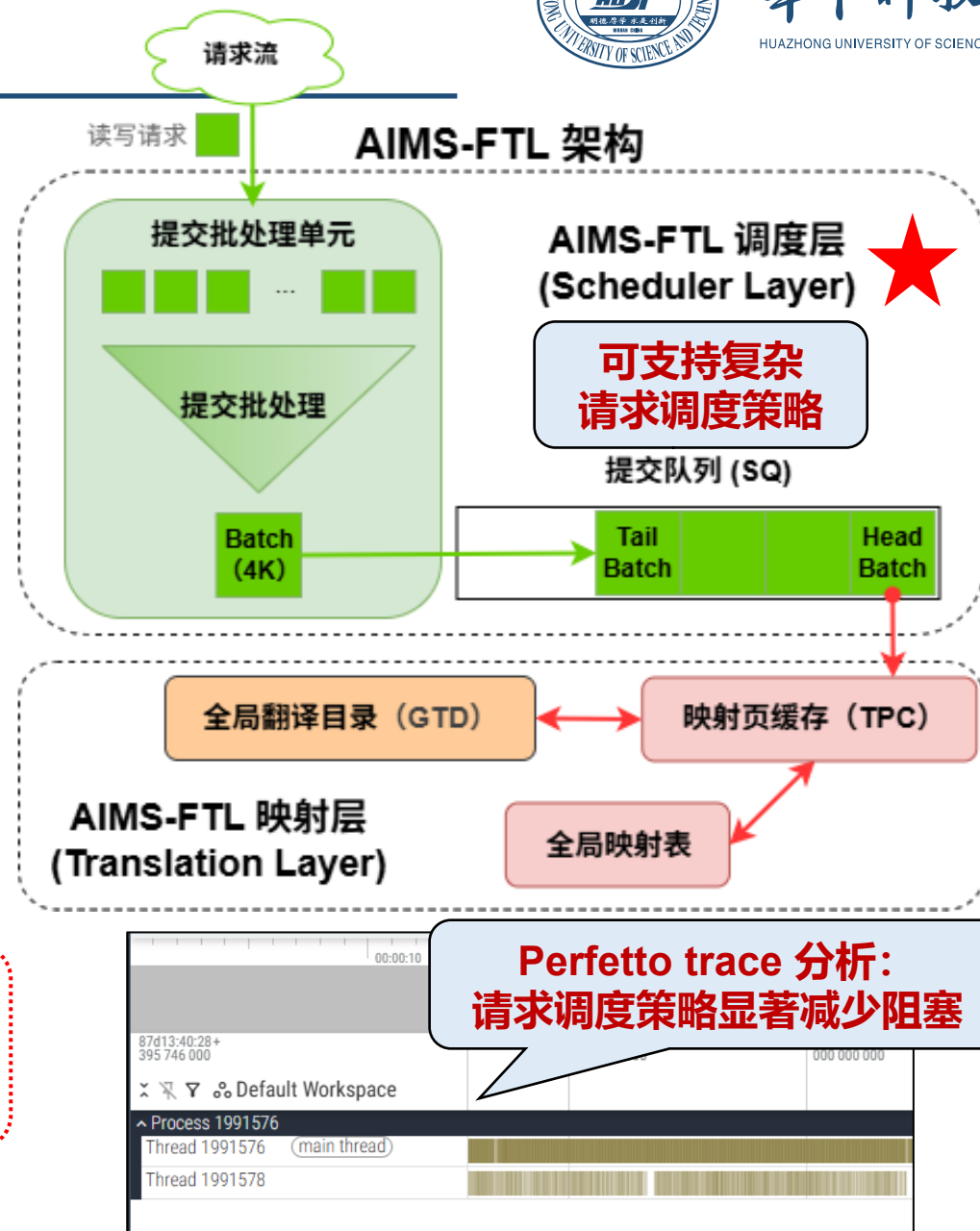
① SQ队列:

- 为应对延迟抖动、减少阻塞，将最近请求以队列形式维护，将FTL的控制面重构为异步模型，支持更复杂的请求调度策略

② 提交批处理:

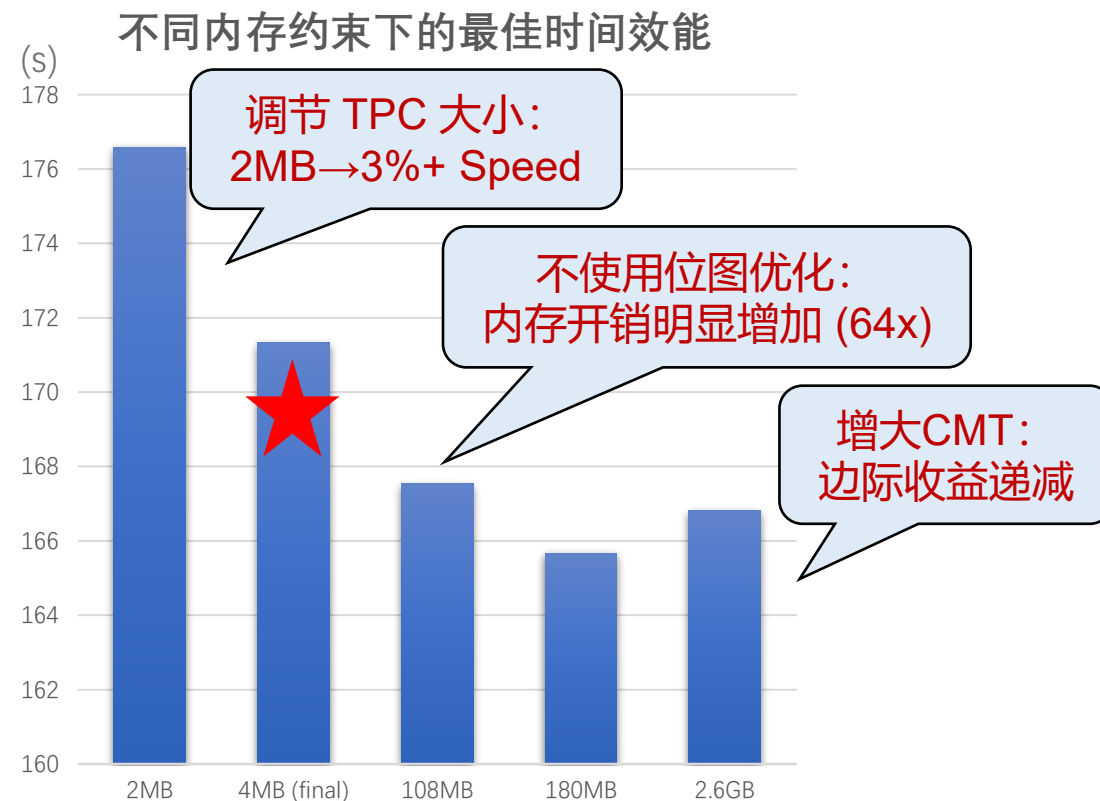
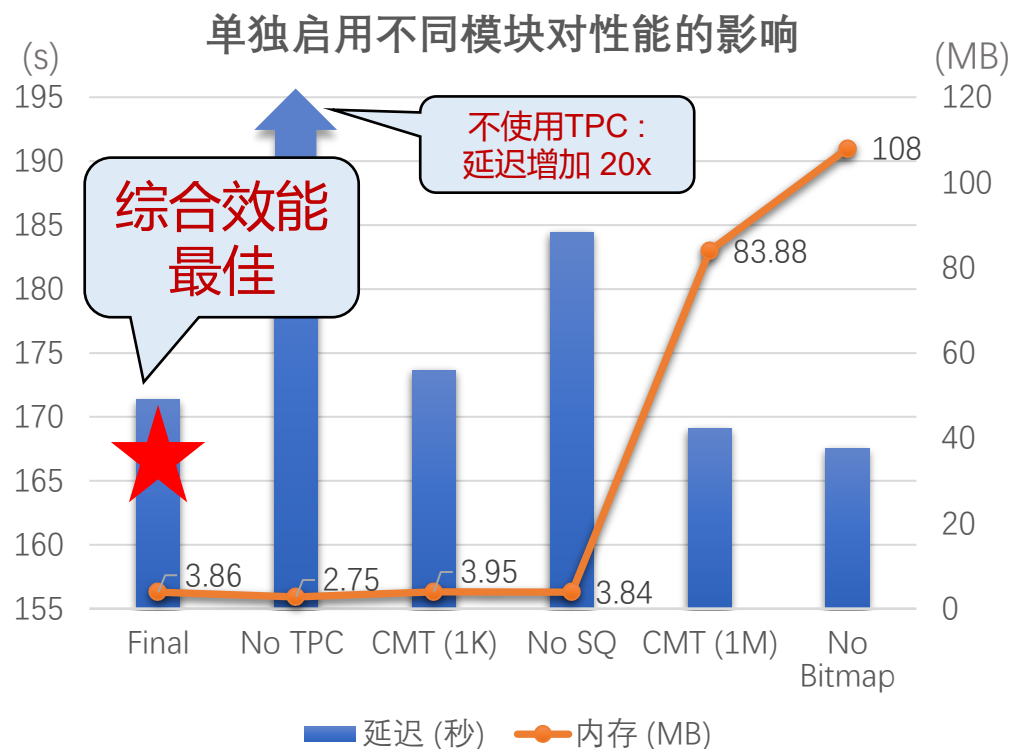
- 为降低锁竞争和上下文切换开销、提升吞吐量，将单次请求的提交转化为批量提交

**延迟降低 7.1% 以上
(171.3s vs 184.4s)**



消融实验和敏感性分析

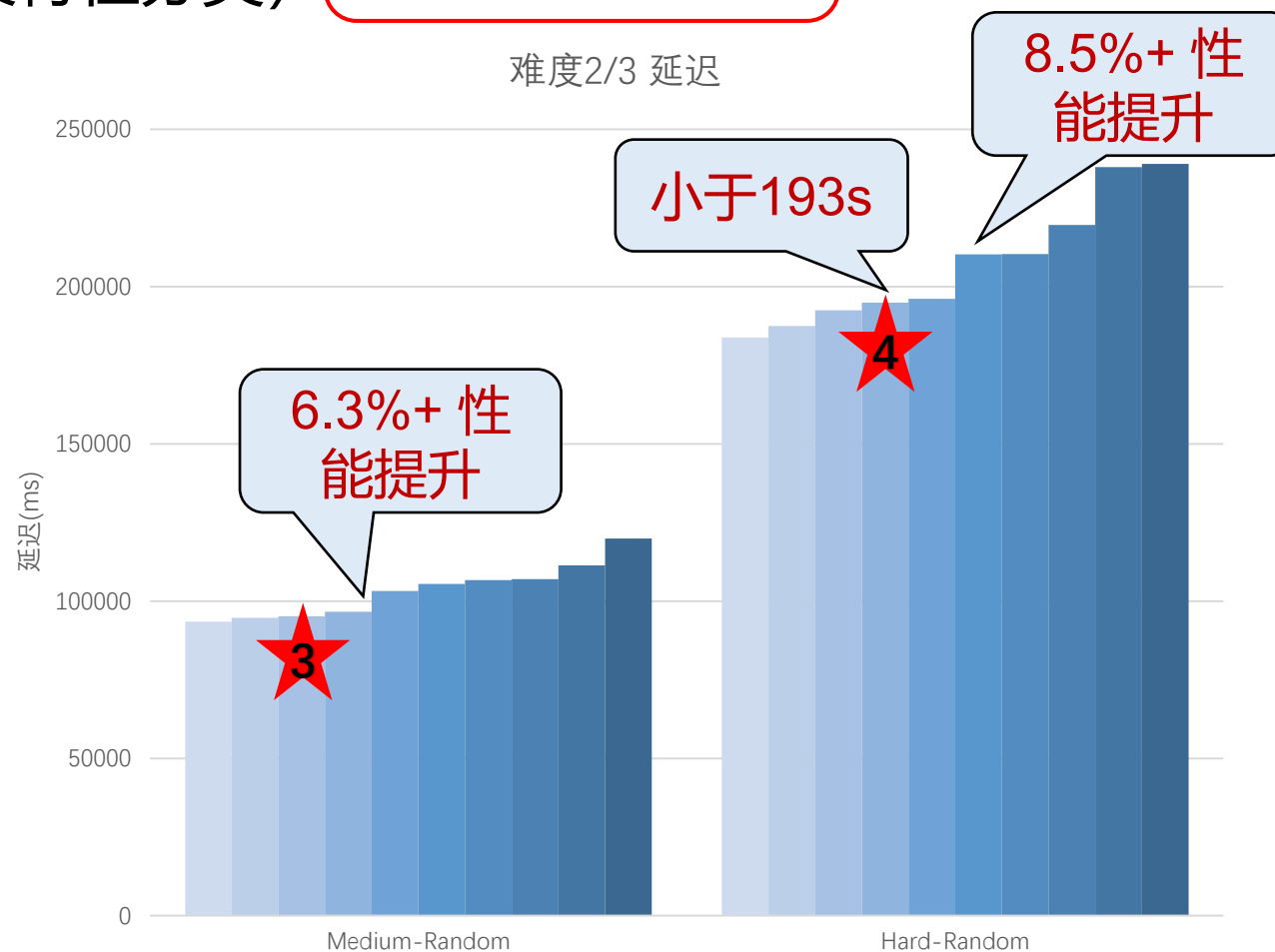
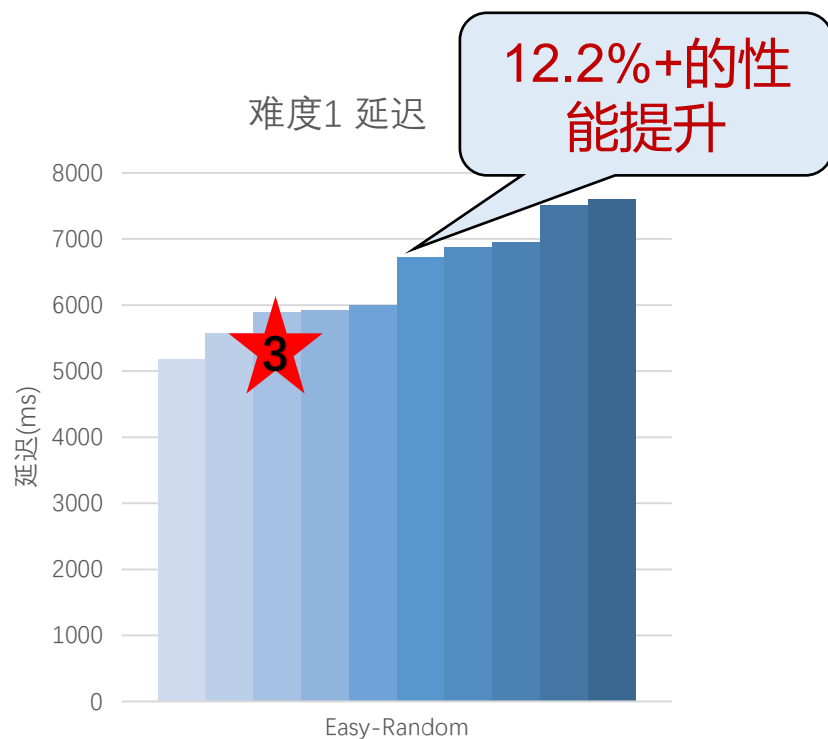
- 通过实验分析确定最终的提交方案



评测结果分析

- 前 Rank10 测试结果（按数据集特征分类）
- 延时：≈ Rank3
- 平均内存开销：< 4KB

综合排名：5



恳请各位老师批评指正！