CHINA LINUX KERNEL
中国Linux内核开发者大会

華中科技大學
网络空间安全学院
School of Cyber Science and Engineering, HUST

第19届中国Linux内核开发者大会

赞助单位

华中科技大学 | HUAWEI | 阿里云 | FUJITSU | 腾讯云
oppo | 火山引擎 | vivo | 龙芯中科 LOONGSON TECHNOLOGY | 蚂蚁集团 ANT GROUP

支持单位

清华大学 Tsinghua University | 迪捷软件 DIGIPROTO | OpenEuler | OpenAnolis 龙蜥社区 | OpenCloudOS

支持社区&媒体

CSDN | 云巅论剑 | sf 思否 | 阅码场 yamcode | InfoQ 极客传媒
51CTO | 开源江湖

2024年10月　湖北 • 武汉

華中科技大學

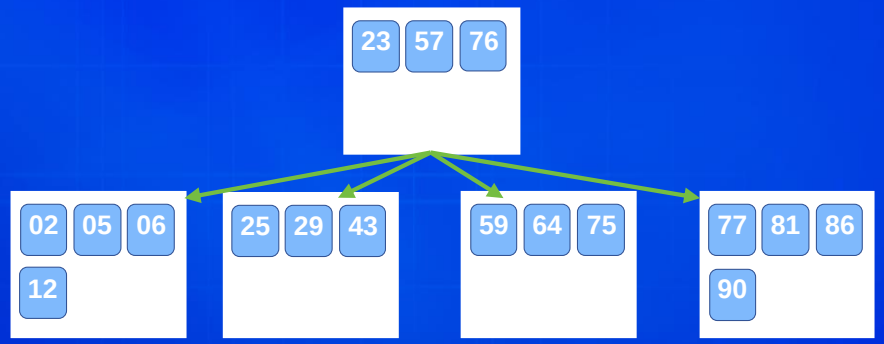
YALT: Yet Another Lockless Tree

余松平(yusongping@huawei.com)
Kuzin Artem(artem.kuzin@huawei.com)
华为中央软件院OS内核实验室

背景介绍—B树

■ 定义

- ✓ 一种有序的平衡多路查找树，存储Key-Value键值对
- ✓ 一棵M阶B树满足如下关键属性：
  - 每个节点最多M个子树；除根节点外，非叶子节点至少⌈m/2⌉个键值对等；
- ✓ B树的主要操作：
  - Insert(k,v)：插入k-v键值对，若超过最大阶数需要split；
  - Delete(k)：删除任意对(k, *)，若小于⌈阶数/2⌉需要merge；
  - PointQuery(k)：返回所有键值对(k, *)；
  - RangeQuery(k1, k2)：返回所有键值对(k，*)，k1<=k<=k2



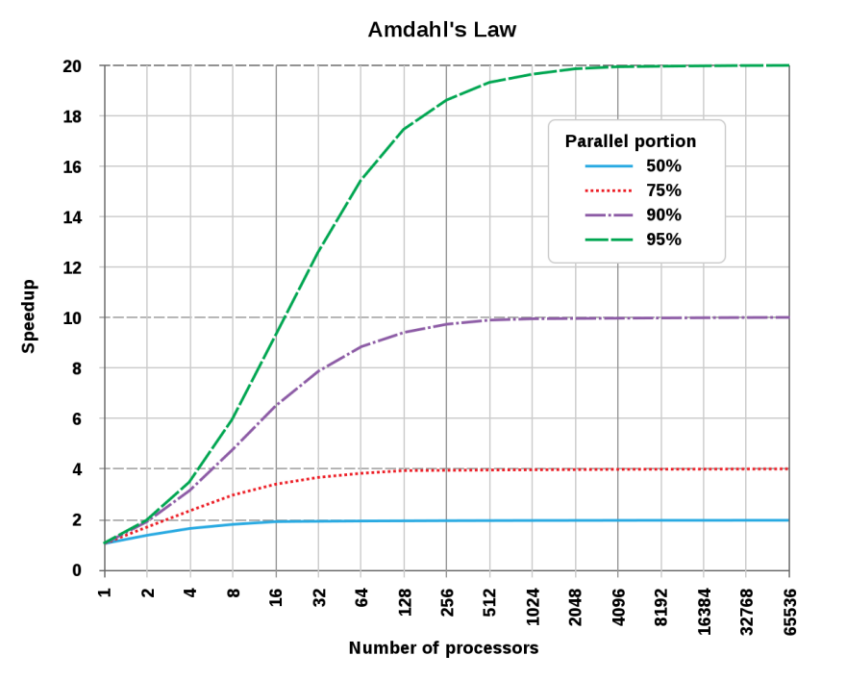
■ 应用举例

- **红黑树**用于管理Linux的CFS调度进程实体的vruntime
- **红黑树**广泛应用在C++的STL中，比如map和set，以及Java的TreeMap等基本数据结构
- **红黑树**管理Linux的虚拟内存，从Linux 6.1开始替换成B+树管理
- **B+树**适用于高效的磁盘IO存取和索引，用于文件系统
- **B+树**适用于范围查询，用于MySQL、PostgreSQL等关系型数据库的查询索引

背景介绍—并发同步演进

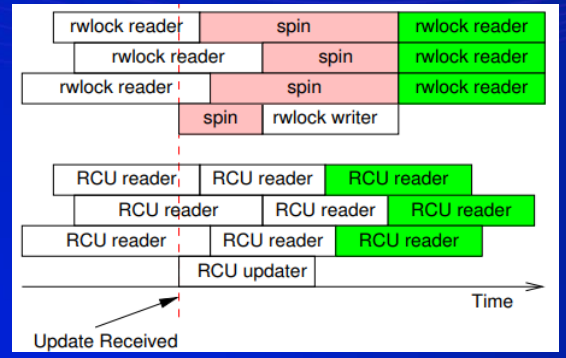
- 阿姆达尔定律：并行计算的加速比;**S表示串行部分比例(↓)**，N表示核的数量

$$\text{Speedup} \le \frac{1}{S+\frac{(1-S)}{N}}$$



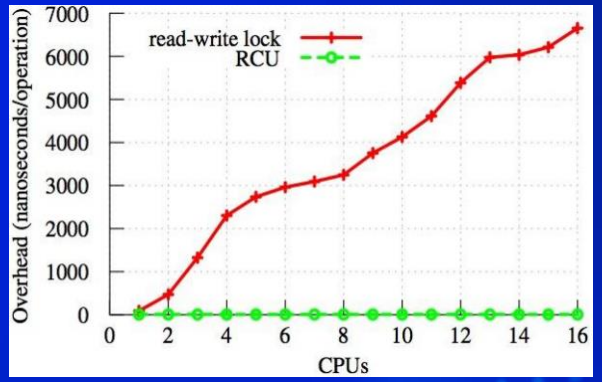
来源:维基百科

- Relax同步语义:锁同步到无锁同步



McKenney P E. Is parallel programming hard, and, if so, what can you do about it?(v2017. 01.02 a)[J]. arXiv preprint arXiv:1701.00854, 2017.

- 早期(1995)应用在Sequent Computer System公司的UNIX系统DYNIX/ptx
- 1999年IBM收购Sequent
- 2000年IBM 开始赞助开源OSDL组织开发linux
- 2001年Paul E. McKenney在OLS(Ottawa Linux Symposium) 首次介绍RCU在linux(2.4.x)中的应用; 同年, BKL开始移除
- 2002年RCU正式进入内核2.5.43; 同年, Maged M. Michael提出Hazard pointer机制
- 2003年RCU加速文件系统Dcache正式进入内核2.5.62
- ……
- *2024年内核中使用RCU API的数量超过* ***21,000+***



多核读写锁获取/释放锁与RCU进入/退出关键区操作开销对比

背景介绍—面向RCU的并发树挑战

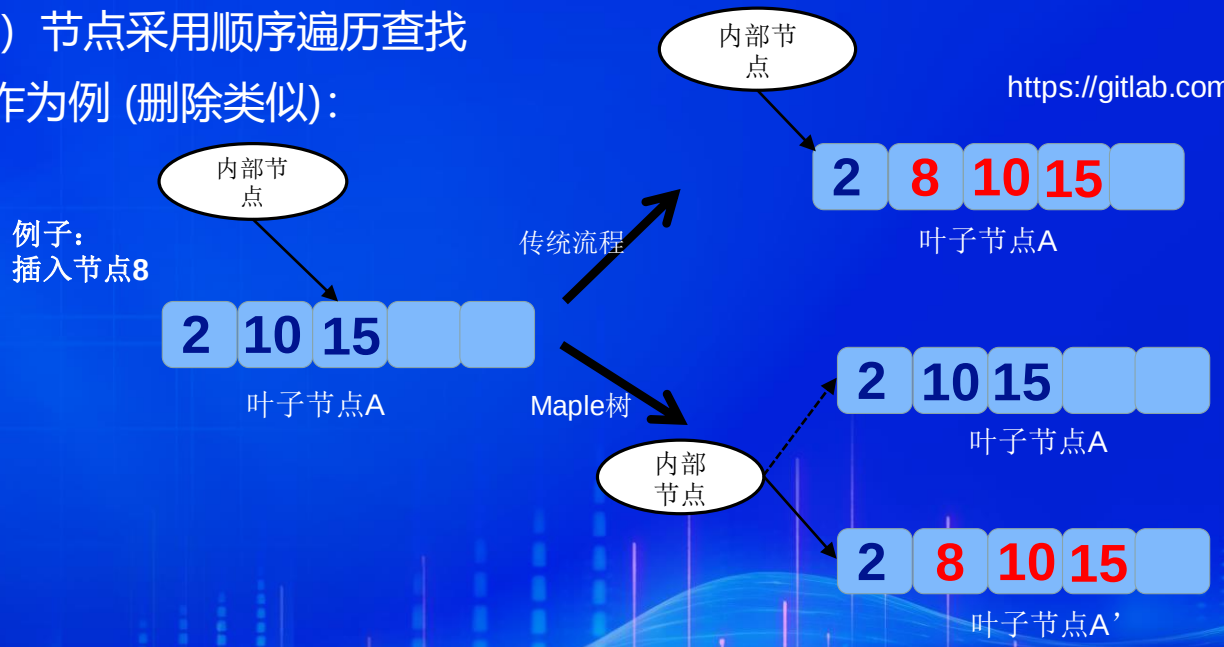
- 保证复杂数据结构B树的**读写并发安全性**和**高效性**(**读写性能和内存开销**)是主要挑战
- 当前RCU-safe的并发树，更新侧涉及的静/动态内存开销大，影响读性能

Name	Bonasi tree	Relativistic rb-tree	Maple tree
Tree Type	binary tree	binary tree	B+-tree
Node Allocation	on-write	on-balance	on-write
Copy Overhead	meduim high	medium	high
Balancing Cost	meduim high	medium	high
Linux support	2.6.37(experimental)	unknown	6.1 and above

背景介绍—Linux Maple tree

- Maple Tree是一种基于B⁺树的范围查找树，支持RCU的安全并发，用于减少mmap_lock的锁竞争
- 主要特征：
  1）完全移除红黑树和双向链表的结构，采用B⁺树去组织管理vma_area_struct[]；
  2）节点默认256字节(4个CacheLine)，相比于rb-tree(24字节)Cache对齐、Cache更友好；
  3）内部节点最大分支因子为16个(augmented是14个)，树的高度大大下降，遍历效率提升；
  4）叶子节点存储value值并形成有序链表，方便范围查询，中间节点仅构建查询路径、不包含数据；
  5）节点采用顺序遍历查找
- 插入操作为例 (删除类似)：

https://gitlab.com/linux-kernel/linux/-/blob/master/lib/maple_tree.c

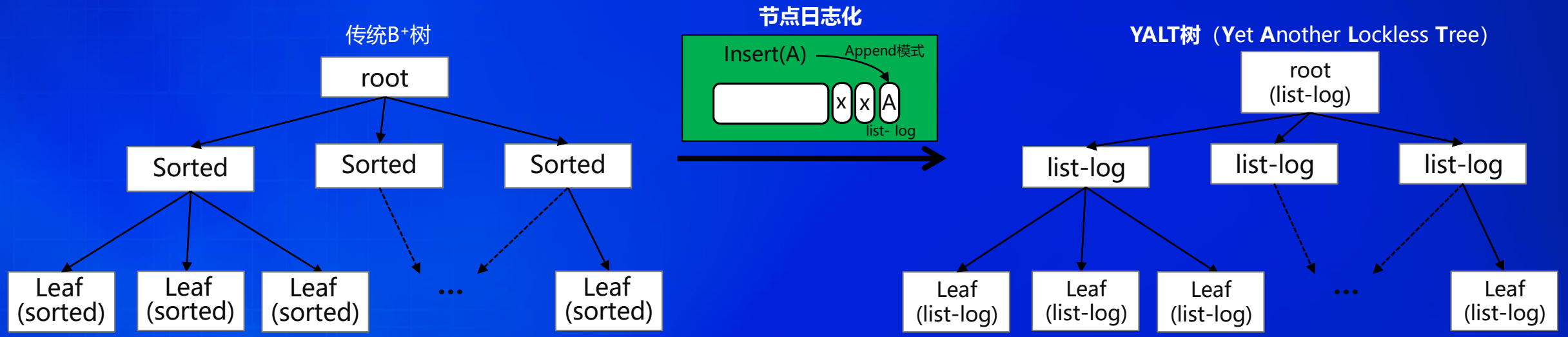


传统流程仅需保证节点值的有序性，
移动10与15完成一次插入排序

Maple树流程分为三步(Copy-Update)：
1）额外分配新的叶子节点，拷贝数据并完成8的插入排序； —— memcpy()
2）发布，原子性修改指针完成新节点加入；—— mas_put_in_tree()
3）synchronize读者，释放旧叶子节点；—— mas_free()

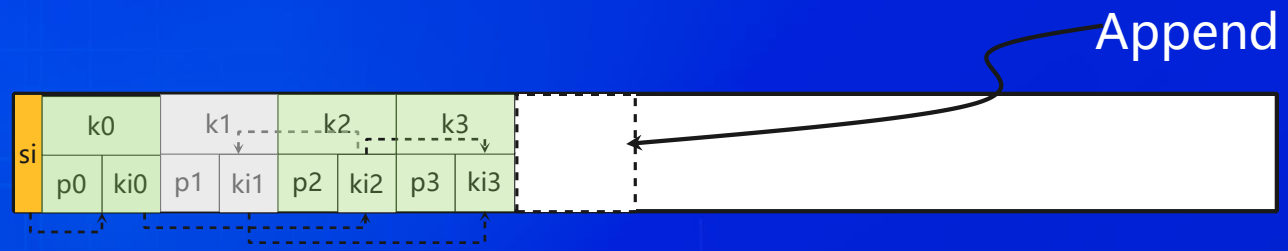
YALT：一种新型高效无锁并发B+树

- 核心设计: 树节点采用链式日志数组（list-log）的新结构组织，支持原子性的节点修改，从而减少额外内存分配和拷贝开销



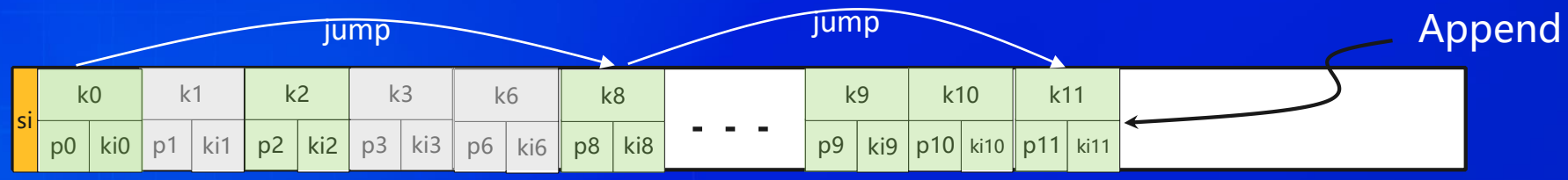
YALT节点结构

- 每个节点包含固定数量的entry，每个entry包含KV对和一个额外的位置索引k-index用于指向其后驱entry，包含一个起始索引(si)用于遍历整个节点
- 插入操作：通过追加在list-log的尾部，通过修改k-index加入到链表中
- 删除操作：通过修改k-index从链表中移除，相应的空间还是占用的状态



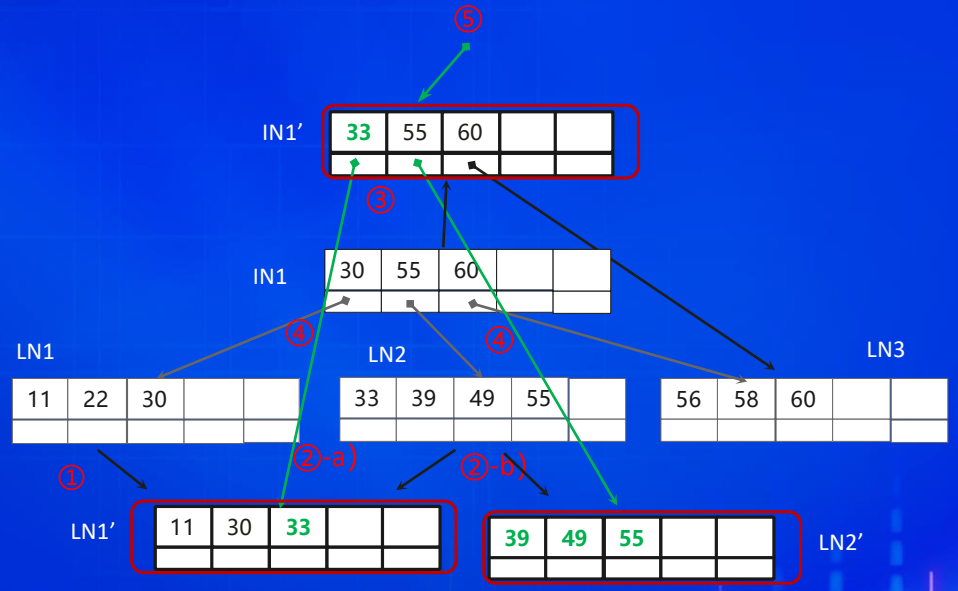
List-log查找

- 目标节点大小(<512B), 通过k-index线性遍历节点内的entry
  - 元数据k-index占用空间影响节点容量
  - entry随机分布影响查找效率
- 优化方法
  - k-index压缩至相应Entry的指针LSBs：保证内部节容量不变
  - Jump Search快速跳跃(offset)到目标Entry范围: 日志结构组织,目标entry在尾部的概率较大



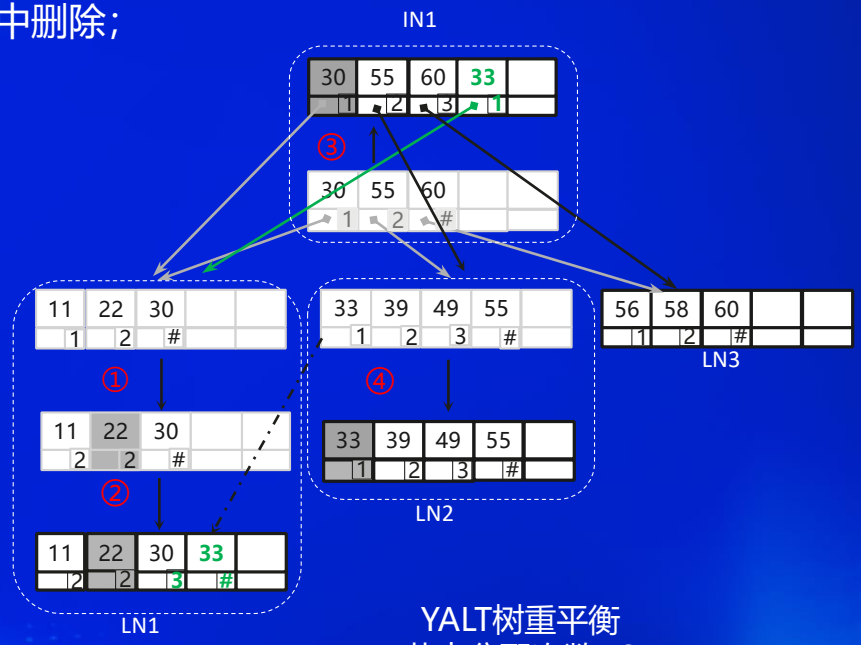
面型轻量级内存分配的负载均衡

- 步骤①：删除叶子节点LN1中键值22，**分配新的叶子节点LN1'**，并将剩余键值移动到LN1' 叶子节点；
- 步骤②：将叶子节点LN2中的较小键值33移动到LN1' 中，**同步分配新的叶子节点LN2'**，并将其他键值对移动到LN2' 中；
- 步骤③：**生成新的内部节点IN1'**，插入键值33；
- 步骤④：关联新分配的叶子节点LN1' 和LN2' 到内部节点IN1' ；
- 步骤⑤： 发布新的内部节点IN1' ；



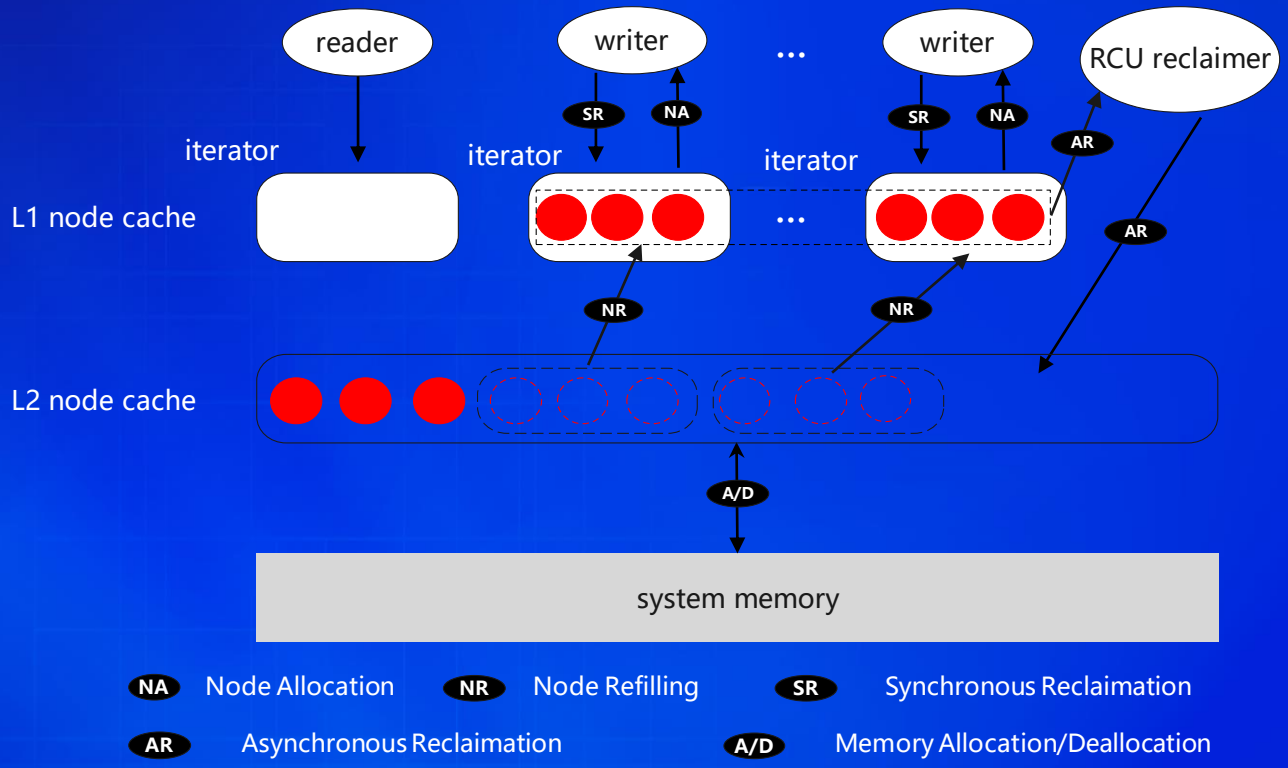
Maple树重平衡
节点分配次数: 3

- 步骤①：修改键值11的k-index为2，,从而将叶子节点LN1中键值22从日志链表中删除；
- 步骤②：将叶子节点LN2中的较小键值33添加到LN1的末尾, 并修改键值30的k-index为3；
- 步骤③：键值33添加到内部节点IN1的日志链表尾部，并修改其k-index为1；
- 步骤④：修改叶子节点LN2的起始k-index索引为2，从而将叶子节点LN2中键值33从日志链表中删除；



YALT树重平衡
节点分配次数≤2

层次结构两级节点缓存

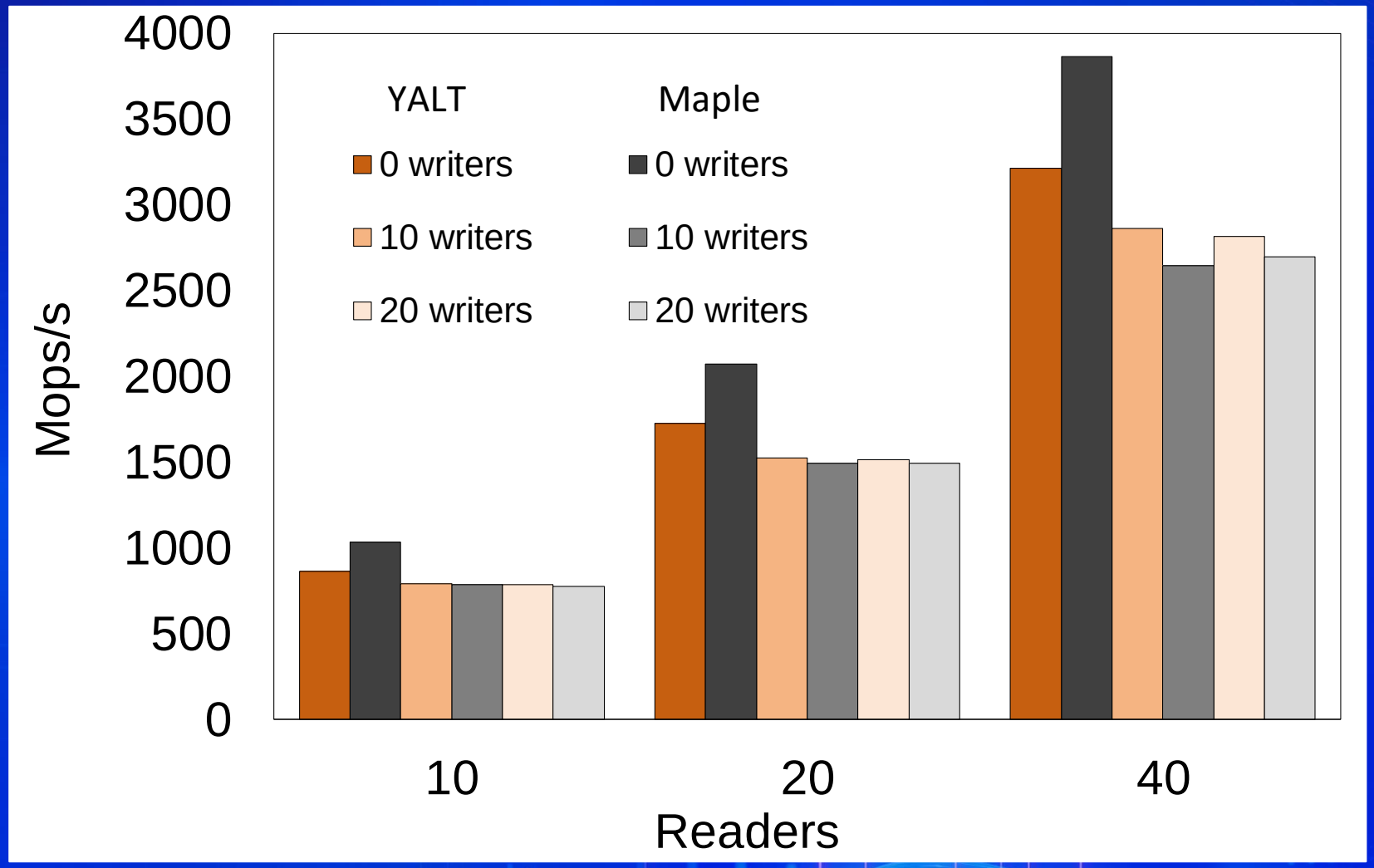


- 现有的节点内存缓存依赖分配器的内置缓存，节点复用不可控：节点真正释放时间与树的访问行为特征不匹配
- Two-level node caching：一个全局node缓存和多个per-request节点缓存
  - 通过RCU释放的节点必须经过L2节点缓存层
  - Per-request节点缓存放在节点的迭代器中，大小不超过4个节点
  - 全局缓存最大为树的高度的两倍

YALT评估

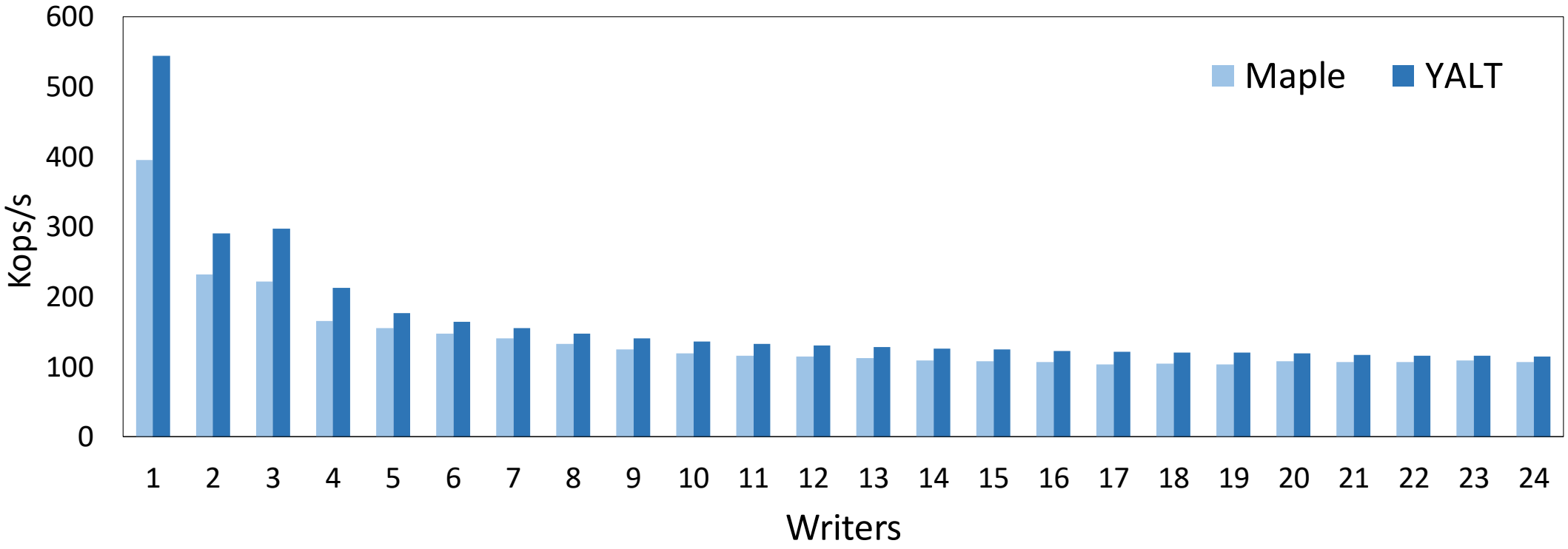
- 物理环境： 16核(2.3GHz)的x86服务器
- 对比数据结构： Linux Maple tree, 初始化 1M entries
- Microbenchmark： 用户态多线程评测读写性能和动静态内存开销
- 集成至Linux6.7 VMM: 测试mmap和pagefault的性能和内存开销

YALT—Lookup性能



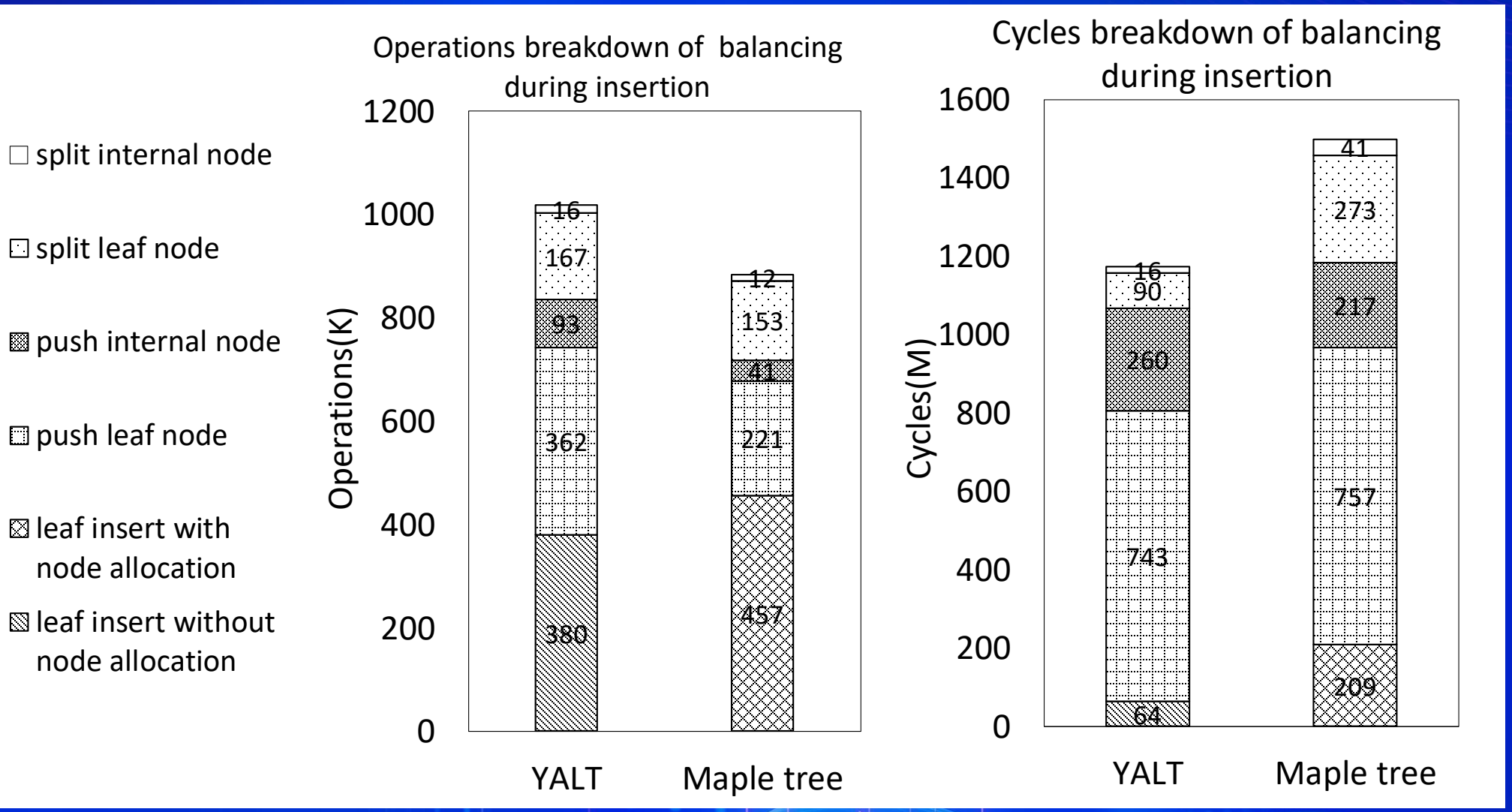
- 在混合读写负载下lookup性能匹配Maple tree
  - ✓Read-only：YALT读性能慢~15%
  - ✓Read-Write：YALT的lookup性能提升1%~6%

YALT—写性能



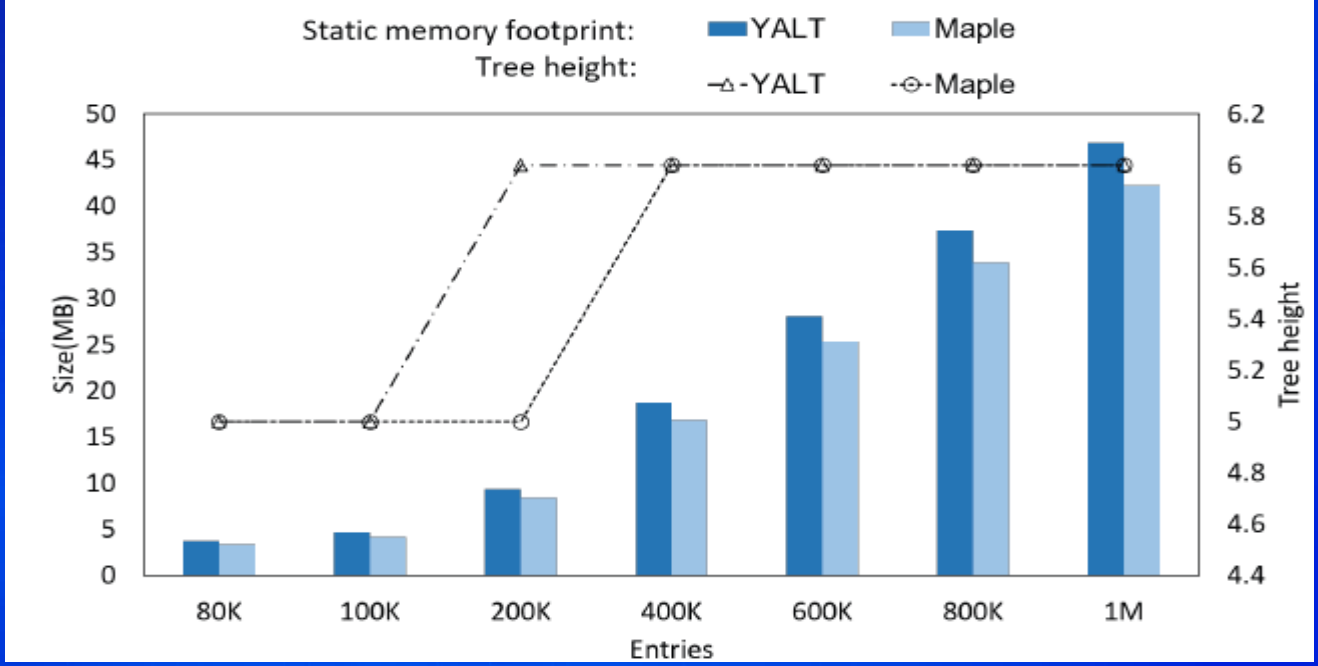
写性能优于Maple tree: 7%~38%

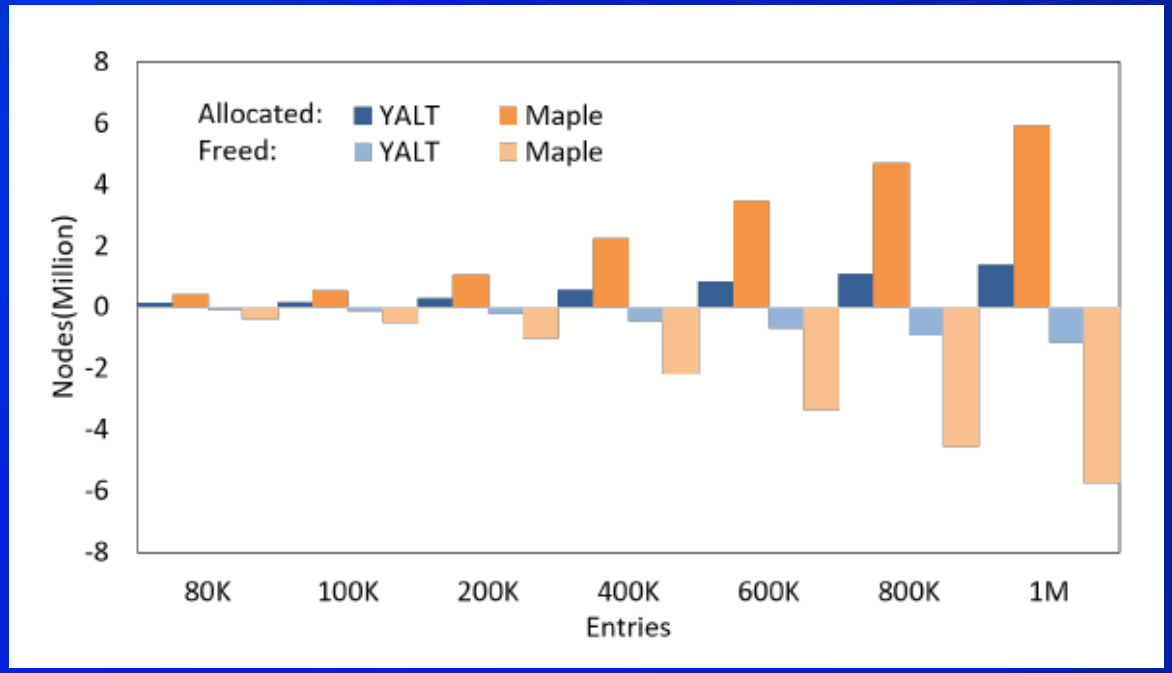
YALT-负载均衡



- ✓ 涉及内存分配的操作减少~37%
- ✓ 均衡的开销下降~27%

YALT—内存开销





- 静态内存仅增加4MB，动态内存减少3.38倍(1400MB→320MB)

基于YALT的Linux 6.7 VMM性能

RCU-enhanced VMM														
Testcase/Number		1	2	4	6	8	10	12	14	16	18	20	22	24	Average
Multi-processes	MMAP1	81.03%	60.81%	21.37%	29.83%	27.26%	44.11%	117.60%	118.39%	128.17%	92.74%	128.98%	120.81%	89.78%	81.61%
	MMAP2	75.14%	27.31%	42.44%	22.38%	62.07%	101.98%	77.47%	112.55%	120.00%	121.64%	111.31%	97.95%	78.49%	80.83%
	PAGE_FAULT1	1.51%	2.79%	2.38%	1.96%	1.68%	1.18%	1.61%	2.13%	1.17%	1.43%	1.16%	1.90%	0.84%	1.67%
	PAGE_FAULT2	1.42%	0.53%	1.16%	0.97%	0.87%	1.34%	1.38%	1.26%	0.94%	1.14%	0.93%	1.33%	1.49%	1.13%
	PAGE_FAULT3	-0.52%	-0.08%	0.02%	0.93%	1.20%	2.02%	1.96%	1.61%	1.79%	0.32%	2.21%	2.29%	2.53%	1.25%
Multi-threaded Process	MMAP1	-4.72%	-6.15%	-10.70%	-10.21%	-21.20%	-19.50%	-8.29%	-4.10%	-9.47%	0.58%	2.55%	-14.43%	-15.54%	-9.32%
	MMAP2	-3.80%	-5.17%	-10.60%	-6.19%	-11.39%	-6.32%	-19.66%	-10.02%	7.87%	-15.96%	-9.51%	6.16%	-10.31%	-7.30%
	PAGE_FAULT1	1.84%	2.93%	4.68%	2.06%	7.31%	6.54%	3.96%	-0.45%	2.39%	8.41%	4.33%	6.68%	6.29%	4.38%
	PAGE_FAULT2	0.66%	0.62%	1.38%	1.37%	6.01%	3.34%	3.60%	3.19%	0.54%	3.54%	3.83%	2.27%	4.29%	2.66%
	PAGE_FAULT3	-0.48%	0.94%	1.80%	3.02%	3.59%	3.69%	2.71%	2.74%	8.54%	7.70%	9.11%	12.61%	10.51%	5.11%

lock-based VMM														
Testcase/Number		1	2	4	6	8	10	12	14	16	18	20	22	24	Average
Multi-processes	MMAP1	59.40%	58.56%	56.89%	58.67%	56.49%	58.00%	57.67%	57.37%	57.80%	58.61%	58.01%	56.90%	57.64%	57.85%
	MMAP2	54.39%	55.31%	54.69%	55.33%	54.68%	54.33%	54.22%	54.46%	53.60%	55.20%	54.45%	54.81%	54.83%	54.64%
	PAGE_FAULT1	0.64%	-0.38%	-0.03%	0.61%	0.97%	0.39%	0.72%	0.49%	0.25%	2.53%	1.73%	2.10%	1.27%	0.87%
	PAGE_FAULT2	1.31%	0.76%	0.46%	0.59%	0.12%	0.56%	0.40%	0.04%	0.71%	0.35%	0.65%	0.15%	0.03%	0.47%
	PAGE_FAULT3	0.95%	0.77%	0.86%	1.10%	0.75%	3.20%	1.36%	1.17%	0.72%	0.25%	1.06%	1.65%	0.28%	1.09%
Multi-threaded Process	MMAP1	-13.83%	-4.94%	-16.52%	9.99%	-11.43%	-14.18%	-10.04%	-7.74%	-7.49%	-1.77%	1.66%	-5.47%	-14.39%	-7.40%
	MMAP2	-12.77%	-13.35%	-1.49%	3.22%	-9.30%	-4.94%	-11.60%	-9.63%	-9.30%	-8.24%	-14.06%	-15.74%	-3.72%	-8.53%
	PAGE_FAULT1	0.35%	-1.94%	6.60%	4.99%	2.91%	3.68%	4.76%	4.30%	3.78%	0.52%	2.58%	-1.10%	-4.33%	2.09%
	PAGE_FAULT2	-1.08%	-0.04%	5.37%	3.80%	2.55%	3.54%	1.12%	4.48%	2.15%	2.39%	4.59%	1.67%	0.61%	2.40%
	PAGE_FAULT3	0.73%	2.54%	8.73%	9.85%	8.81%	8.11%	7.31%	5.45%	7.48%	5.29%	3.61%	5.84%	6.91%	6.21%

总结和未来工作

- 提出一种高效并发无锁树YALT，支持单点和范围操作，读性能匹配Maple tree，具备较好的写性能和内存占用优势；
- 基于YALT可开展的下一步研究工作：

  1) 结合跳跃查找优化读性能；2) 通过节点访问特征(热点路径等)提升缓存策略效率；3) 面向差异化大小的节点设计(调节内部节点和叶子节点的大小)

- 当前Linux的集成：在stress-ng压力测试情况下表现稳定，后续发布至openEuler和Linux社区

欢迎感兴趣的同学加入群聊，共同探索OS技术的真谛

谢谢聆听！