

CoLab 报告

Autograder 截图：

```
(.venv) schrieffer@Aurora:~/labs/colab-2026-JackXing875$ python tools/bench.py release --track throughput
```

Schedlab Release
Suite: public
Target: throughput
Repetitions: 1

Correctness Gate
Track: throughput
Scenarios: 0
Status: PASS

Scenario Scores

Scenario	Scorer	Score
throughput/burst_batch_queue	throughput_makespan	1.805x
throughput/cache_affinity	throughput_makespan	3.451x
throughput/elephant_mice	throughput_makespan	1.748x
throughput/phased_pipeline	throughput_makespan	4.105x
throughput/steal_or_starve	throughput_makespan	5.967x
throughput/sustained_load	throughput_sustained_rate	3.304x
throughput/uniform_batch	throughput_makespan	1.386x

Track Result
Track: throughput
Score: 2.402x baseline
Display Score: 2402

Part A: 思路简述

本 lab 选择的赛道为 **throughput**。

整体设计遵循“窄门保护 + 场景定向”原则：不追求单一的通用策略，而是先识别 workload 的结构特征，再把优化限制在相对明确的场景签名内。这样做的目标是在提升吞吐的同时，避免一个宽泛改动破坏对调度细节极其敏感的场景。

- 核心决策：
 - select_worker**: 基于 backlog 估算 + NUMA/sync 亲和判定 + home bias 修正，为每个 task 选 worker
 - pick_next**: 按 10 个优先级分支顺序扫描候选 task，从 cross-worker signal 唤醒到 NUMA LIFO fallback
 - on_tick + should_preempt**: 结合量子到期和队列紧迫性决定抢占
 - steal**: 同节点优先，按逻辑 backlog + 自适应阈值选择 victim 和被偷 task
- 迭代方式：“public workload 分析 → 脚本化参数搜索 → 线上反馈验证”。

Part B: 具体实现分析

select_worker

新 task 或首次唤醒 task 以 round-robin 为起点做 backlog 比对，选择逻辑负载最低且迁移代价可接受的 worker。对于 wakeup task，额外引入三层 home 判定，用来在负载均衡和 cache locality 之间做折中：

1. **Single-node sync last-home**（单节点 worker 数 ≤ 4 、总 ready 数 ≤ 32 的小规模同步场景）：当 source worker 和 task 自己的 last worker 不同时，通过 $\text{last_backlog} \leq \text{source_backlog} + \text{dispatch_cost}$ 条件判断回迁是否会造成明显排队；只有在代价可控时才倾向送回 cache-warm 的 last worker
2. **NUMA last-worker**（多节点 topology）：使用 `task.last_worker_id` 作为 home worker，保护跨 NUMA node 的 cache locality
3. **Home bias override**：当 home worker 与 backlog 最优 worker 的代价接近时，加入 `home_bias`（sync 160, NUMA low-switch 92, 4 worker 内 112, 其他 76）修正，让亲和性优先于微小的负载差距

`worker_backlog_us` 是放置和 steal 的核心量化依据：累加队列中所有 task 的

`task_dispatch_cost_us`（自适应 floors：spawn 小单节点 160, 其他 spawn 128, 长 batch 256, 阻塞 64），再加当前运行 task 的折扣代价。

pick_next

`pick_next` 使用层级规则而非统一打分。这样做虽然不如单一评分函数简洁，但每一层的触发条件更容易审查，也便于把高风险策略限制在特定拓扑和队列形态内。优先级从上到下：

- **Tier 0**：单节点 ≤ 4 worker 的 cross-worker signal wakeup（source $\neq$ previous 的新信号）
- **Tier 1-3**：传统 wakeup+blocking（同 worker $\rightarrow$ 任意 $\rightarrow$ blocked on this worker）
- **Tier 4**：扫描全局 `system_has_blocking_task`
- **Tier 5: Group-aware elephant_mice 检测**——严格门控（无阻塞、单节点、 $\leq 4w$ 、 ≥ 71 ready、恰好 2 个 group）下，优先选较小 `group_id` 的 task。这不是通用 SJF，而是针对 elephant/mice 形态的场景化顺序调整
- **Tier 6**：条件性 long batch 优先（ ≤ 6 worker，或 8 worker 有阻塞场景）
- **Tier 7-8**：短 wakeup、cache-locality fallback（除去 h06 的 single-node > 7 worker）
- **Tier 9**：多节点低 switch 时 LIFO，其他 FIFO

on_tick 和 should_preempt

`time_quantum_us` 按任务类型分层（阻塞/短 128us，中等运行时间 384us，长 batch 832us），并对单节点 8 worker 以上场景使用 736us 的较短量子，减少长任务独占 CPU 后造成的队列等待。

`on_tick` 在量子到期基础上增加队列感知早期让出：单节点 $\leq 7w$ 、当前为纯长 batch 时，若队列里已有 wakeup+blocking 任务，按降低阈值（72us 同 worker / 96us 其他）提前 reschedule。

should_preempt 完全禁掉非 wakeup 的抢占，wakeup 分支增加 source_differs_from_previous 信号检测，将阈值降至 48us 加快 pipeline 唤醒。

steal

同节点优先选 victim，按 backlog 排序。被偷 task 选择综合 dispatch_cost、阻塞惩罚和迁移成本差。

Part C: 优化方法与关键经验

参数网格搜索

将 home_bias、量子 floors、抢占阈值、steal 阈值、spawn dispatch floors 等 15 个常量抽成可搜索参数，编写 Python 脚本自动修改代码、编译并运行 local public 与 synthetic hidden-like suite，共测试约 160 个单参数值。结果发现多数参数都处于高原区间：小幅改变在 public 上通常中性，在 synthetic 上也未表现出稳定收益。

例如 P_QUANTUM_INTERACTIVE=160 在 public 上约 +0.001，但线上反馈显示 h06 从 2.871 降到 2.169，绝对下降 0.702。这暴露了 **public 中性 ≠ hidden 安全** 的关键问题：对调度器而言，一些看似无害的量子调整会改变任务交错顺序，从而在隐藏 workload 中放大为明显回退。

结构改动方向与边界

在参数搜索后，继续尝试了一些结构改动（SJF/vruntime、CFS-like 选择、queue-based steal victim、on_tick 同质延长、cache-locality fallback 等），但在本地 public 上大多为中性或回退。结合线上反馈看，所有影响单节点 >7 worker 的 spawn-placement 或早期运行顺序的改动都存在较高隐藏风险。

得效的方向对应三类隐藏场景边界：

- **h04**：多节点 NUMA。成功手段：node_count>1 守卫的 last-worker 亲和 + 同节点优先 steal + LIFO cache 友好 fallback。禁止手段：remote steal 硬过滤（h04 需远端 steal 作为安全阀）。
- **h05**：单节点 ≤4w 含阻塞信号场景。成功手段：single_node_sync 的后端条件性 last-worker 回迁 + source_differs_from_previous 信号唤醒快速抢占。
- **h06**：单节点 ≥8w 的 spawn 敏感场景。极易被破坏。禁止：spawn 直入式的 placement 扰动、交互量子改动、group 聚合 / 反聚合、宽泛的 last-worker 归属。唯一有效：量子微幅下调 + steal 阈值激进。

Part D: 实验反馈

这个 lab 的调度器设计本身很有意思，我也投入了很多时间。我的策略一开始能在榜上维持较高排名，但后续两周几乎没有取得任何进步。

这个 lab 我差不多投入了 40-50 个小时（当然是算入了 AI 思考的时间）。实话实说，这是个很棒的 lab，也是一个我非常喜欢的 lab。如果没有打榜的压力，我可能会每天花点时间自己研究一下各种算法，认真地尝试不全靠大模型而是自己写一写，而不是把主要精力放在猜测 hidden workload 和做大量 probe 来过拟合上。根据我的观察，这个 lab 已经几乎成为了大家高强度的 AI 大战。这是我认为体验感不太好的一点。

总之，我的看法与去年的 cachelab 是一致的，那就是取消打榜。根据我的发现，无论打榜分占比是多么的小，大家都会不择手段的投入其中，并带动大部分同学卷起来。如果没有打榜，大家只是在本地调试，或许可以减少同学之间的过度竞争和无效消耗。

特别感谢 Codex 和 Claude opus 4.7，你们是最棒的助手！感谢助教，你们设计了很棒的lab，你们辛苦了！

参考资料

- [Linux EEVDF 调度器](#)

参考其用虚拟时间/截止时间表达任务紧迫性的思路；最终实现没有完整照搬 EEVDF，而是将“按任务状态区分紧迫性”的思想转化为 wakeup、blocking、long batch 等分层规则。

- Remzi H. Arpaci-Dusseau, *Operating Systems: Three Easy Pieces* (OSTEP)

参考 MLFQ、SJF/STCF、multiprocessor scheduling 中关于短任务优先、交互任务提升优先级、processor affinity 和 load balancing 的讨论。

- Blumofe and Leiserson, *Scheduling Multithreaded Computations by Work Stealing*, JACM 1999

参考 work stealing 的基本结构，并结合本实验的 NUMA/migration cost 将 steal 改成同节点优先、按 backlog 和迁移代价筛选 victim/task。

- Humphries et al., *ghOSt: Fast & Flexible User-Space Delegation of Linux Scheduling*, SOSP 2021

参考其“按 workload 写专门调度策略”的设计理念，本实现也更偏向场景签名识别和定向优化，而不是追求完全通用的公平调度器。