

# zoo-bench 性能报告

原始数据：results/zoo-framework-0.6.0/20261011T091515Z.json

## 运行环境

| 项                    | 值                                                                     |
|----------------------|-----------------------------------------------------------------------|
| CPU 型号               | AMD EPYC 7763 64-Core Processor                                       |
| 逻辑核数                 | 4                                                                     |
| 平台                   | Linux-x86_64                                                          |
| 操作系统                 | Linux 6.17.0-1022-azure (#22-Ubuntu SMP Mon Jul 27 17:24:03 UTC 2026) |
| Python               | CPython 3.13.16                                                       |
| 解释器                  | /opt/hostedtoolcache/Python/3.13.16/x64/bin/python                    |
| GIL                  | default                                                               |
| 被测框架（发行元数据）          | 0.6.0                                                                 |
| 被测框架（模块 __version__） | 0.1.1-beta（仅附注，不作版本判据）                                                |
| 被测框架的驱动面世代           | 上一代驱动面（workers 为可写属性，完成信号取自框架内部回调）                                    |
| harness              | zoo-bench 0.0.1 @ 9a42be24ac43f75e2e5d7b16434f91ba3d0717b6            |
| 复现命令                 | zoo-bench run --framework zoo-framework==0.6.0                        |

以 dist\_version（发行元数据）为版本真源；module\_version 仅作参考

## 负载

- 构成：单位工作 = JSON 序列化 + 反序列化 + 字符串拼接，重复若干次以达到目标耗时；档位由 matrix.yaml 的 body\_tiers\_us 声明
- 负载是替身，不是真实 trace：形状接近“取一帧数据 -> 序列化 -> 解析 -> 字符串处理”这类业务动作，但仓库内没有真实样本，结论的适用性以此为前提

## 测量口径

- 执行体耗时在同一次运行内埋点：由执行体在自身内部测量并回传，框架开销 = 端到端 - 它。不做跨运行减法——两次运行的状态不同（缓存、频率、调度噪声），相减引入的是系统性偏差。
- 预热不入统计：每个单元先跑 3 轮预热并丢弃，只统计随后的 30 轮正式采样；采样数等于「正式轮数 x 并发度」。
- 绝对耗时不可跨运行比较：它只用于看量级；跨版本/跨机器要看的是同一次运行内的相对量（开销占比、相对各对照方案的倍数），它们对整体快慢不敏感。

## 结论摘要

- 被测框架 zoo 的执行体时长达到约 10000 微秒 及以上时，其框架开销占端到端的比例降到 15% 以下；短于此档位，选用它的主要代价就是框架开销本身。
- 并发度 1 下，所测档位内 apscheduler 始终快于被测框架——这类档位正是本报告的公开不利数据。
- 并发度 4 下，所测档位内 apscheduler 始终快于被测框架——这类档位正是本报告的公开不利数据。
- 并发度 16 下，所测档位内 apscheduler 始终快于被测框架——这类档位正是本报告的公开不利数据。

- 并发度 64 下，所测档位内 apscheduler 始终快于被测框架——这类档位正是本报告的公开不利数据。
- 并发度 1 下，所测档位内 asyncio\_pool 始终快于被测框架——这类档位正是本报告的公开不利数据。
- 并发度 4 下，所测档位内 asyncio\_pool 始终快于被测框架——这类档位正是本报告的公开不利数据。
- 并发度 16 下，所测档位内 asyncio\_pool 始终快于被测框架——这类档位正是本报告的公开不利数据。
- 并发度 64 下，所测档位内 asyncio\_pool 始终快于被测框架——这类档位正是本报告的公开不利数据。
- 并发度 1 下，所测档位内 bare\_thread 始终快于被测框架——这类档位正是本报告的公开不利数据。
- 并发度 4 下，所测档位内 bare\_thread 始终快于被测框架——这类档位正是本报告的公开不利数据。
- 并发度 16 下，所测档位内 bare\_thread 始终快于被测框架——这类档位正是本报告的公开不利数据。
- 并发度 64 下，所测档位内 bare\_thread 始终快于被测框架——这类档位正是本报告的公开不利数据。
- 并发度 1 下，所测档位内 process\_pool 始终快于被测框架——这类档位正是本报告的公开不利数据。
- 并发度 4 下，所测档位内 process\_pool 始终快于被测框架——这类档位正是本报告的公开不利数据。
- 并发度 16 下，所测档位内 process\_pool 始终快于被测框架——这类档位正是本报告的公开不利数据。
- 并发度 64 下，所测档位内 process\_pool 始终快于被测框架——这类档位正是本报告的公开不利数据。
- 并发度 1 下，所测档位内 thread\_pool 始终快于被测框架——这类档位正是本报告的公开不利数据。
- 并发度 4 下，所测档位内 thread\_pool 始终快于被测框架——这类档位正是本报告的公开不利数据。
- 并发度 16 下，所测档位内 thread\_pool 始终快于被测框架——这类档位正是本报告的公开不利数据。
- 并发度 64 下，所测档位内 thread\_pool 始终快于被测框架——这类档位正是本报告的公开不利数据。

单点加速比没有选型含义；结论以“多大的执行体时长下选哪个方案”表述（同一份框架开销，在 40 微秒的任务上占 70%，在 10 ms 上只占百分之几）

## 开销阈值交叉点

| 方案          | 并发度 | 阈值     | 首个不高于阈值的档位（微秒） | 所测档位（微秒）             | 说明                                                                                                    |
|-------------|-----|--------|----------------|----------------------|-------------------------------------------------------------------------------------------------------|
| apscheduler | 1   | 15.00% | 2700           | 40, 300, 2700, 10000 | 该档位是所测档位中最小的满足者；真实交叉点落在它与前一档之间                                                                        |
| apscheduler | 4   | 15.00% | —              | 40, 300, 2700, 10000 | 该并发度超过这次运行所在机器的并行能力：执行体自报的耗时里含超订带来的调度等待，与「墙钟 / 并发度」不是同一件事，相减的结果没有意义（实测出现过负值）。端到端、吞吐与执行体自报值仍然有效，故照旧给出。 |

| 方案           | 并发度 | 阈值     | 首个不高于阈值的档位（微秒） | 所测档位（微秒）             | 说明                                                                                                    |
|--------------|-----|--------|----------------|----------------------|-------------------------------------------------------------------------------------------------------|
| apscheduler  | 16  | 15.00% | —              | 40, 300, 2700, 10000 | 该并发度超过这次运行所在机器的并行能力：执行体自报的耗时里含超订带来的调度等待，与「墙钟 / 并发度」不是同一件事，相减的结果没有意义（实测出现过负值）。端到端、吞吐与执行体自报值仍然有效，故照旧给出。 |
| apscheduler  | 64  | 15.00% | —              | 40, 300, 2700, 10000 | 该并发度超过这次运行所在机器的并行能力：执行体自报的耗时里含超订带来的调度等待，与「墙钟 / 并发度」不是同一件事，相减的结果没有意义（实测出现过负值）。端到端、吞吐与执行体自报值仍然有效，故照旧给出。 |
| asyncio_pool | 1   | 15.00% | 2700           | 40, 300, 2700, 10000 | 该档位是所测档位中最小的满足者；真实交叉点落在它与前一档之间                                                                        |
| asyncio_pool | 4   | 15.00% | —              | 40, 300, 2700, 10000 | 该并发度超过这次运行所在机器的并行能力：执行体自报的耗时里含超订带来的调度等待，与「墙钟 / 并发度」不是同一件事，相减的结果没有意义（实测出现过负值）。端到端、吞吐与执行体自报值仍然有效，故照旧给出。 |
| asyncio_pool | 16  | 15.00% | —              | 40, 300, 2700, 10000 | 该并发度超过这次运行所在机器的并行能力：执行体自报的耗时里含超订带来的调度等待，与「墙钟 / 并发度」不是同一件事，相减的结果没有意义（实测出现过负值）。端到端、吞吐与执行体自报值仍然有效，故照旧给出。 |

| 方案           | 并发度 | 阈值     | 首个不高于阈值的档位（微秒） | 所测档位（微秒）             | 说明                                                                                                    |
|--------------|-----|--------|----------------|----------------------|-------------------------------------------------------------------------------------------------------|
| asyncio_pool | 64  | 15.00% | —              | 40, 300, 2700, 10000 | 该并发度超过这次运行所在机器的并行能力：执行体自报的耗时里含超订带来的调度等待，与「墙钟 / 并发度」不是同一件事，相减的结果没有意义（实测出现过负值）。端到端、吞吐与执行体自报值仍然有效，故照旧给出。 |
| bare_thread  | 1   | 15.00% | 2700           | 40, 300, 2700, 10000 | 该档位是所测档位中最小的满足者；真实交叉点落在它与前一档之间                                                                        |
| bare_thread  | 4   | 15.00% | —              | 40, 300, 2700, 10000 | 该并发度超过这次运行所在机器的并行能力：执行体自报的耗时里含超订带来的调度等待，与「墙钟 / 并发度」不是同一件事，相减的结果没有意义（实测出现过负值）。端到端、吞吐与执行体自报值仍然有效，故照旧给出。 |
| bare_thread  | 16  | 15.00% | —              | 40, 300, 2700, 10000 | 该并发度超过这次运行所在机器的并行能力：执行体自报的耗时里含超订带来的调度等待，与「墙钟 / 并发度」不是同一件事，相减的结果没有意义（实测出现过负值）。端到端、吞吐与执行体自报值仍然有效，故照旧给出。 |
| bare_thread  | 64  | 15.00% | —              | 40, 300, 2700, 10000 | 该并发度超过这次运行所在机器的并行能力：执行体自报的耗时里含超订带来的调度等待，与「墙钟 / 并发度」不是同一件事，相减的结果没有意义（实测出现过负值）。端到端、吞吐与执行体自报值仍然有效，故照旧给出。 |

| 方案           | 并发度 | 閾值     | 首个不高于閾值的<br>档位（微秒） | 所测档位（微秒）                | 说明                                                                                                    |
|--------------|-----|--------|--------------------|-------------------------|-------------------------------------------------------------------------------------------------------|
| process_pool | 1   | 15.00% | 2700               | 40, 300, 2700,<br>10000 | 该档位是所测档位中最小的满足者；真实交叉点落在它与前一档之间                                                                        |
| process_pool | 4   | 15.00% | —                  | 40, 300, 2700,<br>10000 | 该并发度超过这次运行所在机器的并行能力：执行体自报的耗时里含超订带来的调度等待，与「墙钟 / 并发度」不是同一件事，相减的结果没有意义（实测出现过负值）。端到端、吞吐与执行体自报值仍然有效，故照旧给出。 |
| process_pool | 16  | 15.00% | —                  | 40, 300, 2700,<br>10000 | 该并发度超过这次运行所在机器的并行能力：执行体自报的耗时里含超订带来的调度等待，与「墙钟 / 并发度」不是同一件事，相减的结果没有意义（实测出现过负值）。端到端、吞吐与执行体自报值仍然有效，故照旧给出。 |
| process_pool | 64  | 15.00% | —                  | 40, 300, 2700,<br>10000 | 该并发度超过这次运行所在机器的并行能力：执行体自报的耗时里含超订带来的调度等待，与「墙钟 / 并发度」不是同一件事，相减的结果没有意义（实测出现过负值）。端到端、吞吐与执行体自报值仍然有效，故照旧给出。 |
| thread_pool  | 1   | 15.00% | 300                | 40, 300, 2700,<br>10000 | 该档位是所测档位中最小的满足者；真实交叉点落在它与前一档之间                                                                        |

| 方案          | 并发度 | 阈值     | 首个不高于阈值的档位（微秒） | 所测档位（微秒）             | 说明                                                                                                    |
|-------------|-----|--------|----------------|----------------------|-------------------------------------------------------------------------------------------------------|
| thread_pool | 4   | 15.00% | —              | 40, 300, 2700, 10000 | 该并发度超过这次运行所在机器的并行能力：执行体自报的耗时里含超订带来的调度等待，与「墙钟 / 并发度」不是同一件事，相减的结果没有意义（实测出现过负值）。端到端、吞吐与执行体自报值仍然有效，故照旧给出。 |
| thread_pool | 16  | 15.00% | —              | 40, 300, 2700, 10000 | 该并发度超过这次运行所在机器的并行能力：执行体自报的耗时里含超订带来的调度等待，与「墙钟 / 并发度」不是同一件事，相减的结果没有意义（实测出现过负值）。端到端、吞吐与执行体自报值仍然有效，故照旧给出。 |
| thread_pool | 64  | 15.00% | —              | 40, 300, 2700, 10000 | 该并发度超过这次运行所在机器的并行能力：执行体自报的耗时里含超订带来的调度等待，与「墙钟 / 并发度」不是同一件事，相减的结果没有意义（实测出现过负值）。端到端、吞吐与执行体自报值仍然有效，故照旧给出。 |
| zoo         | 1   | 15.00% | 10000          | 40, 300, 2700, 10000 | 该档位是所测档位中最小的满足者；真实交叉点落在它与前一档之间                                                                        |
| zoo         | 4   | 15.00% | —              | 40, 300, 2700, 10000 | 该并发度超过这次运行所在机器的并行能力：执行体自报的耗时里含超订带来的调度等待，与「墙钟 / 并发度」不是同一件事，相减的结果没有意义（实测出现过负值）。端到端、吞吐与执行体自报值仍然有效，故照旧给出。 |

| 方案  | 并发度 | 阈值     | 首个不高于阈值的档位（微秒） | 所测档位（微秒）             | 说明                                                                                                    |
|-----|-----|--------|----------------|----------------------|-------------------------------------------------------------------------------------------------------|
| zoo | 16  | 15.00% | —              | 40, 300, 2700, 10000 | 该并发度超过这次运行所在机器的并行能力：执行体自报的耗时里含超订带来的调度等待，与「墙钟 / 并发度」不是同一件事，相减的结果没有意义（实测出现过负值）。端到端、吞吐与执行体自报值仍然有效，故照旧给出。 |
| zoo | 64  | 15.00% | —              | 40, 300, 2700, 10000 | 该并发度超过这次运行所在机器的并行能力：执行体自报的耗时里含超订带来的调度等待，与「墙钟 / 并发度」不是同一件事，相减的结果没有意义（实测出现过负值）。端到端、吞吐与执行体自报值仍然有效，故照旧给出。 |

### 相对转折点

| 对照方案         | 并发度 | 该方案不再更快的档位（微秒） | 说明                      |
|--------------|-----|----------------|-------------------------|
| apscheduler  | 1   | —              | 未观测到：所测档位内该对照方案始终快于被测框架 |
| apscheduler  | 4   | —              | 未观测到：所测档位内该对照方案始终快于被测框架 |
| apscheduler  | 16  | —              | 未观测到：所测档位内该对照方案始终快于被测框架 |
| apscheduler  | 64  | —              | 未观测到：所测档位内该对照方案始终快于被测框架 |
| asyncio_pool | 1   | —              | 未观测到：所测档位内该对照方案始终快于被测框架 |
| asyncio_pool | 4   | —              | 未观测到：所测档位内该对照方案始终快于被测框架 |
| asyncio_pool | 16  | —              | 未观测到：所测档位内该对照方案始终快于被测框架 |
| asyncio_pool | 64  | —              | 未观测到：所测档位内该对照方案始终快于被测框架 |
| bare_thread  | 1   | —              | 未观测到：所测档位内该对照方案始终快于被测框架 |
| bare_thread  | 4   | —              | 未观测到：所测档位内该对照方案始终快于被测框架 |
| bare_thread  | 16  | —              | 未观测到：所测档位内该对照方案始终快于被测框架 |

| 对照方案         | 并发度 | 该方案不再更快的档位（微秒） | 说明                      |
|--------------|-----|----------------|-------------------------|
| bare_thread  | 64  | —              | 未观测到：所测档位内该对照方案始终快于被测框架 |
| process_pool | 1   | —              | 未观测到：所测档位内该对照方案始终快于被测框架 |
| process_pool | 4   | —              | 未观测到：所测档位内该对照方案始终快于被测框架 |
| process_pool | 16  | —              | 未观测到：所测档位内该对照方案始终快于被测框架 |
| process_pool | 64  | —              | 未观测到：所测档位内该对照方案始终快于被测框架 |
| thread_pool  | 1   | —              | 未观测到：所测档位内该对照方案始终快于被测框架 |
| thread_pool  | 4   | —              | 未观测到：所测档位内该对照方案始终快于被测框架 |
| thread_pool  | 16  | —              | 未观测到：所测档位内该对照方案始终快于被测框架 |
| thread_pool  | 64  | —              | 未观测到：所测档位内该对照方案始终快于被测框架 |

图表

[缺图：figures/zh-overhead\_ratio.svg]

仅并发度 4 的档位；比值仅在同一次运行内成立。并发度 16, 64 的开销数字已撤下（超出该机器并行能力，见口径章节），故不在图上

[缺图：figures/zh-throughput.svg]

仅执行体 40 微秒 档；吞吐仅在同一次运行内成立

[缺图：figures/zh-relative\_multiple.svg]

每个所测并发度各一个面板（共 4 个）；倍数仅在同一次运行内成立

维度：延迟分位数与抖动

端到端 / 并发度；分位数与相对离散度同表给出，只给中位数的度量不合格

| 并发度 | 执行体档位（微秒） | zoo       | apscheduler | asyncio_pool | bare_thread | process_pool | thread_pool |
|-----|-----------|-----------|-------------|--------------|-------------|--------------|-------------|
| 1   | 40        | 606.25 微秒 | 293.74 微秒   | 145.49 微秒    | 143.75 微秒   | 336.43 微秒    | 74.84 微秒    |
| 1   | 300       | 867.95 微秒 | 559.04 微秒   | 389.99 微秒    | 399.35 微秒   | 606.27 微秒    | 338.26 微秒   |
| 1   | 2700      | 3.263 毫秒  | 2.989 毫秒    | 2.617 毫秒     | 2.690 毫秒    | 3.015 毫秒     | 2.669 毫秒    |
| 1   | 10000     | 10.505 毫秒 | 9.889 毫秒    | 9.708 毫秒     | 9.552 毫秒    | 9.702 毫秒     | 9.577 毫秒    |
| 4   | 40        | 611.17 微秒 | 267.20 微秒   | 153.05 微秒    | 158.85 微秒   | 192.86 微秒    | 95.11 微秒    |
| 4   | 300       | 866.98 微秒 | 526.01 微秒   | 428.07 微秒    | 415.41 微秒   | 283.82 微秒    | 361.48 微秒   |
| 4   | 2700      | 3.307 毫秒  | 3.003 毫秒    | 2.800 毫秒     | 2.693 毫秒    | 1.390 毫秒     | 2.676 毫秒    |
| 4   | 10000     | 10.700 毫秒 | 10.132 毫秒   | 9.695 毫秒     | 9.872 毫秒    | 4.608 毫秒     | 9.867 毫秒    |
| 16  | 40        | 610.25 微秒 | 226.27 微秒   | 138.37 微秒    | 171.89 微秒   | 161.53 微秒    | 92.31 微秒    |
| 16  | 300       | 878.71 微秒 | 484.41 微秒   | 395.42 微秒    | 423.77 微秒   | 278.33 微秒    | 353.51 微秒   |
| 16  | 2700      | 3.306 毫秒  | 2.977 毫秒    | 2.744 毫秒     | 2.704 毫秒    | 1.410 毫秒     | 2.790 毫秒    |
| 16  | 10000     | 10.596 毫秒 | 10.332 毫秒   | 9.964 毫秒     | 9.972 毫秒    | 4.636 毫秒     | 10.146 毫秒   |

| 并发度 | 执行体档位<br>(微秒) | zoo       | apscheduler | asyncio_pool | bare_thread | process_pool | thread_pool |
|-----|---------------|-----------|-------------|--------------|-------------|--------------|-------------|
| 64  | 40            | 565.92 微秒 | 223.95 微秒   | 120.73 微秒    | 170.05 微秒   | 196.54 微秒    | 89.40 微秒    |
| 64  | 300           | 842.59 微秒 | 507.24 微秒   | 371.91 微秒    | 425.03 微秒   | 300.97 微秒    | 355.32 微秒   |
| 64  | 2700          | 3.323 毫秒  | 2.949 毫秒    | 2.832 毫秒     | 2.764 毫秒    | 1.354 毫秒     | 2.788 毫秒    |
| 64  | 10000         | 10.636 毫秒 | 10.585 毫秒   | 10.189 毫秒    | 9.526 毫秒    | 4.629 毫秒     | 10.289 毫秒   |

逐单元数值（分位数、离散度、各轮原始样本）见文末「附录：全部数值」。

## 维度：框架自身开销占比

开销 = 端到端/并发度 - 同一次运行内实测的执行体耗时（不做跨运行减法）。带 — 的格：该并发度超出这台机器的并行能力，执行体自报耗时含超订的调度等待，与该行的每任务端到端不可比，故不给开销数字（端到端与吞吐见各自的维度）

| 并发度 | 执行体档位<br>(微秒) | zoo    | apscheduler | asyncio_pool | bare_thread | process_pool | thread_pool |
|-----|---------------|--------|-------------|--------------|-------------|--------------|-------------|
| 1   | 40            | 88.13% | 80.50%      | 66.60%       | 64.52%      | 83.87%       | 40.99%      |
| 1   | 300           | 62.15% | 46.08%      | 25.72%       | 26.40%      | 49.32%       | 12.38%      |
| 1   | 2700          | 17.04% | 8.91%       | 4.87%        | 4.18%       | 11.36%       | 1.76%       |
| 1   | 10000         | 5.28%  | 3.41%       | 1.51%        | 1.31%       | 3.30%        | 0.64%       |
| 4   | 40            | 89.54% | 79.29%      | 64.75%       | 67.82%      | 51.69%       | 48.45%      |
| 4   | 300           | 62.78% | 39.72%      | 24.92%       | 29.04%      | —            | 14.55%      |
| 4   | 2700          | 19.63% | 10.01%      | 4.93%        | 4.60%       | —            | 2.23%       |
| 4   | 10000         | —      | —           | —            | —           | —            | —           |
| 16  | 40            | —      | —           | —            | —           | —            | —           |
| 16  | 300           | —      | —           | —            | —           | —            | —           |
| 16  | 2700          | —      | —           | —            | —           | —            | —           |
| 16  | 10000         | —      | —           | —            | —           | —            | —           |
| 64  | 40            | —      | —           | —            | —           | —            | —           |
| 64  | 300           | —      | —           | —            | —           | —            | —           |
| 64  | 2700          | —      | —           | —            | —           | —            | —           |
| 64  | 10000         | —      | —           | —            | —           | —            | —           |

逐单元数值（分位数、离散度、各轮原始样本）见文末「附录：全部数值」。

## 维度：吞吐与并发伸缩

吞吐 = 并发度 / 端到端中位数；随并发度是否继续上升即为伸缩性

| 并发度 | 执行体档位<br>(微秒) | zoo    | apscheduler | asyncio_pool | bare_thread | process_pool | thread_pool |
|-----|---------------|--------|-------------|--------------|-------------|--------------|-------------|
| 1   | 40            | 1649.5 | 3404.3      | 6873.2       | 6956.3      | 2972.4       | 13362.6     |
| 1   | 300           | 1152.1 | 1788.8      | 2564.2       | 2504.1      | 1649.4       | 2956.3      |
| 1   | 2700          | 306.5  | 334.5       | 382.1        | 371.7       | 331.7        | 374.6       |
| 1   | 10000         | 95.2   | 101.1       | 103.0        | 104.7       | 103.1        | 104.4       |
| 4   | 40            | 6544.9 | 14970.0     | 26136.1      | 25180.5     | 20740.7      | 42055.6     |
| 4   | 300           | 4613.7 | 7604.4      | 9344.3       | 9629.0      | 14093.4      | 11065.7     |
| 4   | 2700          | 1209.4 | 1331.9      | 1428.5       | 1485.5      | 2878.6       | 1494.9      |
| 4   | 10000         | 373.8  | 394.8       | 412.6        | 405.2       | 868.1        | 405.4       |

| 并发度 | 执行体档位<br>(微秒) | zoo      | apscheduler | asyncio_pool | bare_thread | process_pool | thread_pool |
|-----|---------------|----------|-------------|--------------|-------------|--------------|-------------|
| 16  | 40            | 26218.8  | 70711.5     | 115629.2     | 93082.7     | 99052.0      | 173329.2    |
| 16  | 300           | 18208.4  | 33030.1     | 40462.9      | 37756.1     | 57486.6      | 45260.1     |
| 16  | 2700          | 4839.3   | 5375.2      | 5831.5       | 5917.6      | 11345.1      | 5735.0      |
| 16  | 10000         | 1510.0   | 1548.6      | 1605.9       | 1604.5      | 3451.0       | 1577.0      |
| 64  | 40            | 113090.9 | 285777.7    | 530094.2     | 376362.9    | 325635.1     | 715882.7    |
| 64  | 300           | 75956.6  | 126173.2    | 172085.1     | 150577.9    | 212645.4     | 180117.2    |
| 64  | 2700          | 19258.5  | 21702.3     | 22599.7      | 23154.0     | 47269.4      | 22956.6     |
| 64  | 10000         | 6017.5   | 6046.2      | 6281.4       | 6718.2      | 13826.1      | 6220.2      |

逐单元数值（分位数、离散度、各轮原始样本）见文末「附录：全部数值」。

## 维度：开销归因

口径：把每任务端到端拆成提交侧 / 手交 / 执行体 / 回程四段，并在同一档位与并发度下与各对照方案逐段对照，回答「超出对照的部分落在哪一段」。它是诊断而非主证据：抽样一小批档位，且不与对照方案比总开销。手交为负不是错误：那表示提交调用还没返回、执行体就已经开跑——把建线程一类工作算进提交侧的方案就会这样（实测 bare\_thread 手交 -343

微秒）。回程含完成信号的等待（排空返回前的最后一段），它与主测量是同一口径

状态：ok

本维度是诊断不是主证据：抽样的档位与并发度固定为一小批，不覆盖矩阵里的全部单元。并发度上界取机器的并行能力——超过它时量到的是争用；

抽样档位（微秒）：40、10000；并发度：[1, 4]；该机器并行能力：4

| 方案           | 并发度 | 档位（微秒） | 提交侧       | 手交          | 执行体       | 回程        | 状态                                                  |
|--------------|-----|--------|-----------|-------------|-----------|-----------|-----------------------------------------------------|
| apscheduler  | 1   | 40     | 107.44 微秒 | 91.43 微秒    | 50.14 微秒  | 39.97 微秒  | ok                                                  |
| apscheduler  | 4   | 40     | 110.35 微秒 | 524.42 微秒   | 55.76 微秒  | 229.78 微秒 | ok                                                  |
| apscheduler  | 1   | 10000  | 138.62 微秒 | 110.21 微秒   | 9.664 毫秒  | 49.29 微秒  | ok                                                  |
| apscheduler  | 4   | 10000  | 120.24 微秒 | 12.806 毫秒   | 14.493 毫秒 | 9.873 毫秒  | ok                                                  |
| asyncio_pool | 1   | 40     | 17.47 微秒  | 26.05 微秒    | 47.28 微秒  | 58.04 微秒  | ok                                                  |
| asyncio_pool | 4   | 40     | 18.91 微秒  | 250.49 微秒   | 53.92 微秒  | 283.87 微秒 | ok                                                  |
| asyncio_pool | 1   | 10000  | 23.03 微秒  | 35.74 微秒    | 9.574 毫秒  | 84.68 微秒  | ok                                                  |
| asyncio_pool | 4   | 10000  | 22.74 微秒  | 12.394 毫秒   | 10.117 毫秒 | 10.093 毫秒 | ok                                                  |
| bare_thread  | 1   | 40     | 143.44 微秒 | -75.55 微秒   | 51.49 微秒  | 30.78 微秒  | ok                                                  |
| bare_thread  | 4   | 40     | 152.07 微秒 | -74.15 微秒   | 50.96 微秒  | 304.52 微秒 | ok                                                  |
| bare_thread  | 1   | 10000  | 5.171 毫秒  | -5093.61 微秒 | 9.397 毫秒  | 37.25 微秒  | ok                                                  |
| bare_thread  | 4   | 10000  | 5.180 毫秒  | -5085.51 微秒 | 14.704 毫秒 | 9.875 毫秒  | ok                                                  |
| process_pool | 1   | 40     | —         | —           | —         | —         | not measurable：执行体的起止时刻读不到——它没有在本进程里运行（跨进程档位无法这样分段） |

| 方案           | 并发度 | 档位（微秒） | 提交侧      | 手交        | 执行体       | 回程        | 状态                                                  |
|--------------|-----|--------|----------|-----------|-----------|-----------|-----------------------------------------------------|
| process_pool | 4   | 40     | —        | —         | —         | —         | not measurable：执行体的起止时刻读不到——它没有在本进程里运行（跨进程档位无法这样分段） |
| process_pool | 1   | 10000  | —        | —         | —         | —         | not measurable：执行体的起止时刻读不到——它没有在本进程里运行（跨进程档位无法这样分段） |
| process_pool | 4   | 10000  | —        | —         | —         | —         | not measurable：执行体的起止时刻读不到——它没有在本进程里运行（跨进程档位无法这样分段） |
| thread_pool  | 1   | 40     | 8.13 微秒  | 10.33 微秒  | 46.76 微秒  | 15.15 微秒  | ok                                                  |
| thread_pool  | 4   | 40     | 11.19 微秒 | 173.04 微秒 | 49.03 微秒  | 153.53 微秒 | ok                                                  |
| thread_pool  | 1   | 10000  | 12.12 微秒 | 16.27 微秒  | 9.498 毫秒  | 27.49 微秒  | ok                                                  |
| thread_pool  | 4   | 10000  | 13.25 微秒 | 9.992 毫秒  | 14.876 毫秒 | 8.754 毫秒  | ok                                                  |
| zoo          | 1   | 40     | 23.37 微秒 | 366.57 微秒 | 71.55 微秒  | 131.62 微秒 | ok                                                  |
| zoo          | 4   | 40     | 16.35 微秒 | 1.144 毫秒  | 67.00 微秒  | 1.218 毫秒  | ok                                                  |
| zoo          | 1   | 10000  | 36.26 微秒 | 392.84 微秒 | 9.674 毫秒  | 145.56 微秒 | ok                                                  |
| zoo          | 4   | 10000  | 21.46 微秒 | 10.784 毫秒 | 14.929 毫秒 | 11.213 毫秒 | ok                                                  |

### 超出各对照方案的部分落在哪一段

| 对照方案         | 并发度 | 档位（微秒） | 每任务超出     | 主要落在 | 该段之差      |
|--------------|-----|--------|-----------|------|-----------|
| apscheduler  | 1   | 40     | 324.22 微秒 | 手交   | 275.14 微秒 |
| asyncio_pool | 1   | 40     | 467.68 微秒 | 手交   | 340.52 微秒 |
| bare_thread  | 1   | 40     | 465.54 微秒 | 手交   | 442.12 微秒 |
| thread_pool  | 1   | 40     | 536.55 微秒 | 手交   | 356.24 微秒 |
| apscheduler  | 4   | 40     | 1.548 毫秒  | 回程   | 987.97 微秒 |
| asyncio_pool | 4   | 40     | 1.870 毫秒  | 回程   | 933.88 微秒 |
| bare_thread  | 4   | 40     | 2.048 毫秒  | 手交   | 1.219 毫秒  |
| thread_pool  | 4   | 40     | 2.092 毫秒  | 回程   | 1.064 毫秒  |
| apscheduler  | 1   | 10000  | 313.01 微秒 | 手交   | 282.63 微秒 |
| asyncio_pool | 1   | 10000  | 568.72 微秒 | 手交   | 357.10 微秒 |
| bare_thread  | 1   | 10000  | 766.30 微秒 | 手交   | 5.486 毫秒  |
| thread_pool  | 1   | 10000  | 708.50 微秒 | 手交   | 376.57 微秒 |
| apscheduler  | 4   | 10000  | 2.600 毫秒  | 回程   | 1.340 毫秒  |

| 对照方案         | 并发度 | 档位（微秒） | 每任务超出     | 主要落在 | 该段之差      |
|--------------|-----|--------|-----------|------|-----------|
| asyncio_pool | 4   | 10000  | 2.688 毫秒  | 执行体  | 4.812 毫秒  |
| bare_thread  | 4   | 10000  | 10.806 毫秒 | 手交   | 15.869 毫秒 |
| thread_pool  | 4   | 10000  | 3.498 毫秒  | 回程   | 2.459 毫秒  |

用到的插桩接缝（在 harness 侧临时包装，不改框架代码）：BaseWaiter.\_dispatch\_worker（把 worker 交给调度模型）；BaseWaiter.execute\_service（走一次调度轮）

本次未能量到的接缝（被测框架的这一代没有对应入口，故相关细分项不在场）：WorkerDispatchCore.resolve\_period / resolve\_phase / resolve\_run\_timeout（每轮策略查询）

插桩自身的成本（被测框架那一侧，由同一子进程里插桩前后两轮配对得到，可如实折价）：zoo 40 微秒 / 并发 1：不插桩 624.17 微秒 对插桩后 616.59 微秒，差 7.58

微秒（插桩后反而更快：这轮里机器漂移比插桩成本还大）；zoo 40 微秒 / 并发 4：不插桩 2.464 毫秒 对插桩后 2.477 毫秒，差 13.43 微秒；zoo 10000 微秒 / 并发 1：不插桩 10.256 毫秒 对插桩后 10.283 毫秒，差 26.40 微秒；zoo 10000 微秒 / 并发 4：不插桩 42.327 毫秒 对插桩后 42.213 毫秒，差 114.21 微秒（插桩后反而更快：这轮里机器漂移比插桩成本还大）

## 被测框架的提交侧由什么构成

- 并发度 1、档位 40 微秒：调度轮其余（判定、锁与调度列表维护） 3.82 微秒；派发（交给调度模型，含池的入队） 13.62 微秒；每轮策略查询（周期/相位/超时各查一次配置） 0.00 微秒；提交侧其他（适配器自己的记账与完成信号状态） 5.84 微秒
- 并发度 4、档位 40 微秒：调度轮其余（判定、锁与调度列表维护） 3.20 微秒；派发（交给调度模型，含池的入队） 10.58 微秒；每轮策略查询（周期/相位/超时各查一次配置） 0.00 微秒；提交侧其他（适配器自己的记账与完成信号状态） 4.34 微秒
- 并发度 1、档位 10000 微秒：调度轮其余（判定、锁与调度列表维护） 5.62 微秒；派发（交给调度模型，含池的入队） 19.72 微秒；每轮策略查询（周期/相位/超时各查一次配置） 0.00 微秒；提交侧其他（适配器自己的记账与完成信号状态） 9.41 微秒
- 并发度 4、档位 10000 微秒：调度轮其余（判定、锁与调度列表维护） 4.23 微秒；派发（交给调度模型，含池的入队） 15.24 微秒；每轮策略查询（周期/相位/超时各查一次配置） 0.00 微秒；提交侧其他（适配器自己的记账与完成信号状态） 5.66 微秒

## 归因结论

- 并发度 1、执行体 40 微秒 下，被测框架相对 apscheduler 每任务多花 324 微秒，其中主要落在「手交」（+275 微秒）。
- 注意：该组逐段差额并非同号——「提交侧」为 -84 微秒，读的时候要一并看。
- 并发度 1、执行体 40 微秒 下，被测框架相对 asyncio\_pool 每任务多花 468 微秒，其中主要落在「手交」（+341 微秒）。
- 并发度 1、执行体 40 微秒 下，被测框架相对 bare\_thread 每任务多花 466 微秒，其中主要落在「手交」（+442 微秒）。
- 注意：该组逐段差额并非同号——「提交侧」为 -120 微秒，读的时候要一并看。
- 并发度 1、执行体 40 微秒 下，被测框架相对 thread\_pool 每任务多花 537 微秒，其中主要落在「手交」（+356 微秒）。
- 被测框架在该组的提交侧由以下部分构成（微秒）：调度轮其余（判定、锁与调度列表维护） 4、派发（交给调度模型，含池的入队） 14、每轮策略查询（周期/相位/超时各查一次配置） 0、提交侧其他（适配器自己的记账与完成信号状态） 6。

- 并发度 4、执行体 40 微秒 下，被测框架相对 `apscheduler` 每任务多花 1548 微秒，其中主要落在「回程」（+988 微秒）。
- 注意：该组逐段差额并非同号——「提交侧」为 -94 微秒，读的时候要一并看。
- 并发度 4、执行体 40 微秒 下，被测框架相对 `asyncio_pool` 每任务多花 1870 微秒，其中主要落在「回程」（+934 微秒）。
- 注意：该组逐段差额并非同号——「提交侧」为 -3 微秒，读的时候要一并看。
- 并发度 4、执行体 40 微秒 下，被测框架相对 `bare_thread` 每任务多花 2048 微秒，其中主要落在「手交」（+1219 微秒）。
- 注意：该组逐段差额并非同号——「提交侧」为 -136 微秒，读的时候要一并看。
- 并发度 4、执行体 40 微秒 下，被测框架相对 `thread_pool` 每任务多花 2092 微秒，其中主要落在「回程」（+1064 微秒）。
- 被测框架在该组的提交侧由以下部分构成（微秒）：调度轮其余（判定、锁与调度列表维护）3、派发（交给调度模型，含池的入队）11、每轮策略查询（周期/相位/超时各查一次配置）0、提交侧其他（适配器自己的记账与完成信号状态）4。
- 并发度 1、执行体 10000 微秒 下，被测框架相对 `apscheduler` 每任务多花 313 微秒，其中主要落在「手交」（+283 微秒）。
- 注意：该组逐段差额并非同号——「提交侧」为 -102 微秒，读的时候要一并看。
- 并发度 1、执行体 10000 微秒 下，被测框架相对 `asyncio_pool` 每任务多花 569 微秒，其中主要落在「手交」（+357 微秒）。
- 并发度 1、执行体 10000 微秒 下，被测框架相对 `bare_thread` 每任务多花 766 微秒，其中主要落在「手交」（+5486 微秒）。
- 注意：该组逐段差额并非同号——「提交侧」为 -5135 微秒，读的时候要一并看。
- 并发度 1、执行体 10000 微秒 下，被测框架相对 `thread_pool` 每任务多花 709 微秒，其中主要落在「手交」（+377 微秒）。
- 被测框架在该组的提交侧由以下部分构成（微秒）：调度轮其余（判定、锁与调度列表维护）6、派发（交给调度模型，含池的入队）20、每轮策略查询（周期/相位/超时各查一次配置）0、提交侧其他（适配器自己的记账与完成信号状态）9。
- 并发度 4、执行体 10000 微秒 下，被测框架相对 `apscheduler` 每任务多花 2600 微秒，其中主要落在「回程」（+1340 微秒）。
- 注意：该组逐段差额并非同号——「手交」为 -2022 微秒，读的时候要一并看。
- 并发度 4、执行体 10000 微秒 下，被测框架相对 `asyncio_pool` 每任务多花 2688 微秒，其中主要落在「执行体」（+4812 微秒）。
- 注意：该组逐段差额并非同号——「手交」为 -1610 微秒，读的时候要一并看。
- 并发度 4、执行体 10000 微秒 下，被测框架相对 `bare_thread` 每任务多花 10806 微秒，其中主要落在「手交」（+15869 微秒）。
- 注意：该组逐段差额并非同号——「提交侧」为 -5158 微秒，读的时候要一并看。
- 并发度 4、执行体 10000 微秒 下，被测框架相对 `thread_pool` 每任务多花 3498 微秒，其中主要落在「回程」（+2459 微秒）。
- 被测框架在该组的提交侧由以下部分构成（微秒）：调度轮其余（判定、锁与调度列表维护）4、派发（交给调度模型，含池的入队）15、每轮策略查询（周期/相位/超时各查一次配置）0、提交侧其他（适配器自己的记账与完成信号状态）6。

## 维度：调度语义的代价

口径：本维度采用内部开关对照（同一 workload 下语义全关 vs 逐个开启），不与对照方案横向比较——裸写法没有优先级/超时/重试的对应物，故只能回答“本框架的语义值多少钱”，不能回答“比裸写法贵多少”

状态：not\_measured（结论在该次测量装着的「previous」代驱动面上得出）

在当前被测版本上，枚举出的语义项没有一项可测：要么其执行动作未实现，要么所在层无法执行反应器。按 design D7 的逃生口，此类项被剔除而非用近似状态代替

| 语义项 | 层级  | 探查方式                    | 可测 | 证据 / 原因                                                                                                                                                                                                                                                                                 |
|-----|-----|-------------------------|----|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 超时  | 派发层 | 行为探查（按调度轮反复驱动，量在飞表摘除时刻） | 否  | 判定存在但执行动作未实现：本探查按调度轮驱动、探查 worker 也声明了循环，调度轮确实跑过，故未摘除只能是判定本身没有执行动作——开启它只多算一次减法，量到的不是语义的代价                                                                                                                                                                                                |
| 优先级 | 事件层 | 结构探查（检查执行所需的符号是否存在）     | 否  | 事件层执行反应器要先有 perform：EventWorker.execute 调 reactor.perform (workers/event_worker.py)，而 EventReactor 只定义了 execute (reactor/event_reactor.py) ——反应器一旦匹配即抛 AttributeError；EventPriorityCalculator 存在 (fifo/node/event_fifo_node.py) 但未接入 drain 循环——workers/event_worker.py 里“根据优先级排序”只有一句注释 |
| 重试  | 事件层 | 结构探查（检查执行所需的符号是否存在）     | 否  | 事件层执行反应器要先有 perform：EventWorker.execute 调 reactor.perform (workers/event_worker.py)，而 EventReactor 只定义了 execute (reactor/event_reactor.py) ——反应器一旦匹配即抛 AttributeError；retry_times 的默认 0 是干净的关闭态，但触发条件是“没有匹配到反应器”，而反应器路径本身不可执行                                                           |

本结论按被测版本得出，坐标只对该版本成立；往 matrix.yaml 追加版本后须重跑本探查

维度：原生任务线：同一被测对象的三种执行形态

口径：该线独立测量：它不进入主矩阵的透视表与交叉点，其数值也不可与其他表里的对照方案横向比较

该线结论：not\_applicable

被测框架不具备该能力（找不到这些符号：adaptive\_params, dual\_arm\_worker, native\_params, native\_task\_worker）——该线对此版本不适用，这不是失败

等价性门禁：未跑到（前置探查没走到这一步）

所测任务 modbus\_rtu.parse\_response——Modbus RTU 响应帧解析：输入是一条裸 RTU ADU 响应帧（不含任何传输层封装），其长度即该档位的帧字节

规模档位 R1、R8、R125（帧字节 7、21、255）；并发度 1、4、16、64；每格预热 100 轮 + 采样 100 轮，每个并发度一个子进程（共 4 个）

被测框架在哪些档位更快

被测框架处于优势的档位（两侧差异大于该档位的带宽）。它与「公开的不利数据」出自同一份同运行内的相对比，只是一个取更快的、一个取更慢的——两者的覆盖面相同，故谁都挑不了档位。本节的缺席不构成不合格：分不出胜负或处处更慢时它就应当是空的

所测档位内未出现被测框架处于优势的情形。

公开的不利数据

被测框架处于劣势的档位（两侧差异大于该档位的带宽）。这部分缺失的报告不合格——一份只展示自己赢的报告，读者有理由认为它不可信。差异小于带宽的档位不在本节，而在「分不出胜负」那一节：那是判据的结果，不是把它们藏起来

| 对照方案         | 并发度 | 执行体档位（微秒） | 差距                         | 被测框架中位数   |
|--------------|-----|-----------|----------------------------|-----------|
| apscheduler  | 1   | 40        | apscheduler 比 zoo 快 2.06x  | 606.25 微秒 |
| asyncio_pool | 1   | 40        | asyncio_pool 比 zoo 快 4.17x | 606.25 微秒 |
| bare_thread  | 1   | 40        | bare_thread 比 zoo 快 4.22x  | 606.25 微秒 |
| process_pool | 1   | 40        | process_pool 比 zoo 快 1.80x | 606.25 微秒 |
| thread_pool  | 1   | 40        | thread_pool 比 zoo 快 8.10x  | 606.25 微秒 |
| apscheduler  | 1   | 300       | apscheduler 比 zoo 快 1.55x  | 867.95 微秒 |
| asyncio_pool | 1   | 300       | asyncio_pool 比 zoo 快 2.23x | 867.95 微秒 |
| bare_thread  | 1   | 300       | bare_thread 比 zoo 快 2.17x  | 867.95 微秒 |
| process_pool | 1   | 300       | process_pool 比 zoo 快 1.43x | 867.95 微秒 |
| thread_pool  | 1   | 300       | thread_pool 比 zoo 快 2.57x  | 867.95 微秒 |
| apscheduler  | 1   | 2700      | apscheduler 比 zoo 快 1.09x  | 3.263 毫秒  |
| asyncio_pool | 1   | 2700      | asyncio_pool 比 zoo 快 1.25x | 3.263 毫秒  |
| bare_thread  | 1   | 2700      | bare_thread 比 zoo 快 1.21x  | 3.263 毫秒  |
| thread_pool  | 1   | 2700      | thread_pool 比 zoo 快 1.22x  | 3.263 毫秒  |

| 对照方案         | 并发度 | 执行体档位（微秒） | 差距                         | 被测框架中位数   |
|--------------|-----|-----------|----------------------------|-----------|
| apscheduler  | 1   | 10000     | apscheduler 比 zoo 快 1.06x  | 10.505 毫秒 |
| asyncio_pool | 1   | 10000     | asyncio_pool 比 zoo 快 1.08x | 10.505 毫秒 |
| bare_thread  | 1   | 10000     | bare_thread 比 zoo 快 1.10x  | 10.505 毫秒 |
| process_pool | 1   | 10000     | process_pool 比 zoo 快 1.08x | 10.505 毫秒 |
| thread_pool  | 1   | 10000     | thread_pool 比 zoo 快 1.10x  | 10.505 毫秒 |
| apscheduler  | 4   | 40        | apscheduler 比 zoo 快 2.29x  | 611.17 微秒 |
| asyncio_pool | 4   | 40        | asyncio_pool 比 zoo 快 3.99x | 611.17 微秒 |
| bare_thread  | 4   | 40        | bare_thread 比 zoo 快 3.85x  | 611.17 微秒 |
| process_pool | 4   | 40        | process_pool 比 zoo 快 3.17x | 611.17 微秒 |
| thread_pool  | 4   | 40        | thread_pool 比 zoo 快 6.43x  | 611.17 微秒 |
| apscheduler  | 4   | 300       | apscheduler 比 zoo 快 1.65x  | 866.98 微秒 |
| asyncio_pool | 4   | 300       | asyncio_pool 比 zoo 快 2.03x | 866.98 微秒 |
| bare_thread  | 4   | 300       | bare_thread 比 zoo 快 2.09x  | 866.98 微秒 |
| process_pool | 4   | 300       | process_pool 比 zoo 快 3.05x | 866.98 微秒 |
| thread_pool  | 4   | 300       | thread_pool 比 zoo 快 2.40x  | 866.98 微秒 |
| asyncio_pool | 4   | 2700      | asyncio_pool 比 zoo 快 1.18x | 3.307 毫秒  |
| bare_thread  | 4   | 2700      | bare_thread 比 zoo 快 1.23x  | 3.307 毫秒  |
| process_pool | 4   | 2700      | process_pool 比 zoo 快 2.38x | 3.307 毫秒  |
| thread_pool  | 4   | 2700      | thread_pool 比 zoo 快 1.24x  | 3.307 毫秒  |
| apscheduler  | 4   | 10000     | apscheduler 比 zoo 快 1.06x  | 10.700 毫秒 |
| asyncio_pool | 4   | 10000     | asyncio_pool 比 zoo 快 1.10x | 10.700 毫秒 |
| bare_thread  | 4   | 10000     | bare_thread 比 zoo 快 1.08x  | 10.700 毫秒 |
| process_pool | 4   | 10000     | process_pool 比 zoo 快 2.32x | 10.700 毫秒 |
| apscheduler  | 16  | 40        | apscheduler 比 zoo 快 2.70x  | 610.25 微秒 |
| asyncio_pool | 16  | 40        | asyncio_pool 比 zoo 快 4.41x | 610.25 微秒 |

| 对照方案         | 并发度 | 执行体档位（微秒） | 差距                         | 被测框架中位数   |
|--------------|-----|-----------|----------------------------|-----------|
| bare_thread  | 16  | 40        | bare_thread 比 zoo 快 3.55x  | 610.25 微秒 |
| process_pool | 16  | 40        | process_pool 比 zoo 快 3.78x | 610.25 微秒 |
| thread_pool  | 16  | 40        | thread_pool 比 zoo 快 6.61x  | 610.25 微秒 |
| apscheduler  | 16  | 300       | apscheduler 比 zoo 快 1.81x  | 878.71 微秒 |
| asyncio_pool | 16  | 300       | asyncio_pool 比 zoo 快 2.22x | 878.71 微秒 |
| bare_thread  | 16  | 300       | bare_thread 比 zoo 快 2.07x  | 878.71 微秒 |
| process_pool | 16  | 300       | process_pool 比 zoo 快 3.16x | 878.71 微秒 |
| thread_pool  | 16  | 300       | thread_pool 比 zoo 快 2.49x  | 878.71 微秒 |
| apscheduler  | 16  | 2700      | apscheduler 比 zoo 快 1.11x  | 3.306 毫秒  |
| asyncio_pool | 16  | 2700      | asyncio_pool 比 zoo 快 1.21x | 3.306 毫秒  |
| bare_thread  | 16  | 2700      | bare_thread 比 zoo 快 1.22x  | 3.306 毫秒  |
| process_pool | 16  | 2700      | process_pool 比 zoo 快 2.34x | 3.306 毫秒  |
| thread_pool  | 16  | 2700      | thread_pool 比 zoo 快 1.19x  | 3.306 毫秒  |
| asyncio_pool | 16  | 10000     | asyncio_pool 比 zoo 快 1.06x | 10.596 毫秒 |
| bare_thread  | 16  | 10000     | bare_thread 比 zoo 快 1.06x  | 10.596 毫秒 |
| process_pool | 16  | 10000     | process_pool 比 zoo 快 2.29x | 10.596 毫秒 |
| thread_pool  | 16  | 10000     | thread_pool 比 zoo 快 1.04x  | 10.596 毫秒 |
| apscheduler  | 64  | 40        | apscheduler 比 zoo 快 2.53x  | 565.92 微秒 |
| asyncio_pool | 64  | 40        | asyncio_pool 比 zoo 快 4.69x | 565.92 微秒 |
| bare_thread  | 64  | 40        | bare_thread 比 zoo 快 3.33x  | 565.92 微秒 |
| process_pool | 64  | 40        | process_pool 比 zoo 快 2.88x | 565.92 微秒 |
| thread_pool  | 64  | 40        | thread_pool 比 zoo 快 6.33x  | 565.92 微秒 |
| apscheduler  | 64  | 300       | apscheduler 比 zoo 快 1.66x  | 842.59 微秒 |
| asyncio_pool | 64  | 300       | asyncio_pool 比 zoo 快 2.27x | 842.59 微秒 |
| bare_thread  | 64  | 300       | bare_thread 比 zoo 快 1.98x  | 842.59 微秒 |

| 对照方案         | 并发度 | 执行体档位（微秒） | 差距                         | 被测框架中位数   |
|--------------|-----|-----------|----------------------------|-----------|
| process_pool | 64  | 300       | process_pool 比 zoo 快 2.80x | 842.59 微秒 |
| thread_pool  | 64  | 300       | thread_pool 比 zoo 快 2.37x  | 842.59 微秒 |
| apscheduler  | 64  | 2700      | apscheduler 比 zoo 快 1.13x  | 3.323 毫秒  |
| asyncio_pool | 64  | 2700      | asyncio_pool 比 zoo 快 1.17x | 3.323 毫秒  |
| bare_thread  | 64  | 2700      | bare_thread 比 zoo 快 1.20x  | 3.323 毫秒  |
| process_pool | 64  | 2700      | process_pool 比 zoo 快 2.45x | 3.323 毫秒  |
| thread_pool  | 64  | 2700      | thread_pool 比 zoo 快 1.19x  | 3.323 毫秒  |
| asyncio_pool | 64  | 10000     | asyncio_pool 比 zoo 快 1.04x | 10.636 毫秒 |
| bare_thread  | 64  | 10000     | bare_thread 比 zoo 快 1.12x  | 10.636 毫秒 |
| process_pool | 64  | 10000     | process_pool 比 zoo 快 2.30x | 10.636 毫秒 |
| thread_pool  | 64  | 10000     | thread_pool 比 zoo 快 1.03x  | 10.636 毫秒 |

## 分不出胜负的档位

两侧分不出胜负的档位：差异小于该档位的带宽。带宽取自同一次运行内两侧各自的跨轮离散度（每任务中位数的标准差 / 中位数）之和，故它随样本变化，读者可用留档里的逐轮样本自行复核。它同样覆盖全部所测档位、逐项写明原因——本节不是藏东西的地方：把噪声级差异报成「快 1.00x」才是

判据：带宽 = 被测框架侧与对照侧各自的跨轮相对离散度之和；单侧的跨轮相对离散度 = 该单元逐轮的「每任务中位数」的标准差 / 中位数

本期带宽：1.6% 到 41.9%（随各档位的离散度不同而不同）。

| 对照方案         | 并发度 | 执行体档位（微秒） | 比值    | 原因                               |
|--------------|-----|-----------|-------|----------------------------------|
| process_pool | 1   | 2700      | 0.92x | 两侧差异 7.6%<br>小于带宽<br>21.2%，分不出胜负 |
| apscheduler  | 4   | 2700      | 0.91x | 两侧差异 9.2%<br>小于带宽<br>9.6%，分不出胜负  |
| thread_pool  | 4   | 10000     | 0.92x | 两侧差异 7.8%<br>小于带宽<br>9.2%，分不出胜负  |
| apscheduler  | 16  | 10000     | 0.98x | 两侧差异 2.5%<br>小于带宽<br>5.1%，分不出胜负  |
| apscheduler  | 64  | 10000     | 1.00x | 两侧差异 0.5%<br>小于带宽<br>4.1%，分不出胜负  |

## 口径局限与偏差来源

- 被测层级：调度派发层：直接驱动 `BaseWaiter.execute_service()`（线程池模式），不走 Master 的完整运行循环——后者的服务循环每秒一次，会把延迟量级从微秒抬到秒。每提交一个任务走一次调度轮，故该轮的全部簿记都计在这一个任务头上；Master 的用法是每轮服务多个 worker，那份开销本会摊到多个任务上
- 口径说明（apscheduler）：APScheduler 的定时作业模型，含作业登记与触发判定成本；该库会吞掉作业内异常，本档已在 drain 时重新抛出。口径说明：该库不提供作业完成通知，故完成信号由本档的 Condition 给出，每任务一次通知的开销计入其端到端
- 口径说明（asyncio\_pool）：asyncio 事件循环调度同步执行体（等价于 `loop.run_in_executor`），非原生协程并发；与线程池的差别是事件循环自身开销
- 口径说明（bare\_thread）：每任务派生一个 `threading.Thread`，无池化、无复用；线程创建成本计入端到端。收集执行体自报耗时用无锁的 `list.append`（GIL 下本身原子）——插桩不得给某一档加别档没有的成本，否则对照就成了插桩的对照
- 口径说明（process\_pool）：ProcessPoolExecutor：执行体与返回值跨进程序列化的成本计入端到端，与多线程档位不可直接等价比较
- 口径说明（thread\_pool）：concurrent.futures.ThreadPoolExecutor，池大小 = 并发度；线程复用、无任务队列语义
- 口径说明（zoo）：端到端终点为「worker 完成并在飞表注销」；框架其后的结果投递按世代分流：上一代的投递路径是坏的（dispatch 按 reactor\_name 而非主题查响应器，每次调用必抛），完成信号只能取自框架内部回调；当前代该处已修好，完成信号取自框架自己的结果响应器。口径说明：完成信号由本档的 Condition 给出，每任务一次通知的开销计入其端到端；该方向的偏差使被测框架看起来更慢，即对被测框架不利而非有利
- 不给出开销数字的组：以下（适配器，并发度）组的并发度超过这次运行所在机器的并行能力：执行体自报的耗时里含超订带来的调度等待，与「墙钟 / 并发度」不是同一件事，相减的结果没有意义（实测出现过负值，并衍生出无意义的变化倍数）。故本报告对它们不给框架开销、也不给交叉点；端到端、吞吐与执行体自报值仍然有效，原始留档里的字段也一个没删。判据取自环境自述的逻辑核数，另有一条证据性兜底：算出来的开销为负同样判为不可读——框架只会加时间不会减时间
- 原生任务线：未出结论：原生任务线结论为 not\_applicable（被测框架不具备该能力（找不到这些符号：adaptive\_params, dual\_arm\_worker, native\_params, native\_task\_worker）——该线对此版本不适用，这不是失败）、未产出任何数字。它记在这里而不是并入自检：一条线的失败不得拖住其余版本的数据与主矩阵
- 未参与判定的自检项：这些单元的执行体档位偏差未参与判定（并发度高于最低档，或档位短到墙钟受调度颗粒度支配）。原始值仍在场，但那两处的墙钟不足以断定校准是否正确——读这些档位的数字时要把它算进去

绝对耗时：仅在同一次运行内部有效，不可跨运行比较

## 加入你自己的对照

本 harness 的对照方案是开放的：实现 `zoo_bench.adapters.BaseAdapter` 的四个方法（`setup` / `submit` / `drain` / `teardown`），用 `zoo_bench.adapters.registry.register` 登记，或在 `matrix.yaml` 的 `extra_modules` 里列出你的模块路径——无需改动本仓库任何文件。你的实现会通过同一套等价性验证（提交 N 个执行体、断言全部执行且各执行一次），不通过的不会被采信。我们主动把这条路径写进报告：一份由被测框架维护者撰写、测量被测框架的报告，读者有理由怀疑对照被写慢或口径被挑选；让任何人能提交自己的对照，是这份报告可被质疑、也可被证伪的前提。

## 自检

- 测量自检整体：通过

- 进程隔离：{'parent\_pid': 2921, 'child\_pids': [3381, 3384, 3387, 3391, 3394, 3400, 3406, 3412, 3418, 3436, 3454, 3472, 3490, 3546, 3603, 3661, 3729, 3731, 3733, 3735, 3737, 3742, 3747, 3752, 3758, 3775, 3792, 3809, 3828, 3887, 3947, 4010, 4079, 4113, 4147, 4181, 4215, 4348, 4482, 4615, 4748, 5277, 5806, 6335, 6865, 8979, 11092, 13205, 15321, 15325, 15329, 15333, 15337, 15344, 15351, 15358, 15365, 15385, 15404, 15423, 15442, 15509, 15576, 15643, 15712, 15714, 15716, 15718, 15720, 15725, 15730, 15735, 15744, 15761, 15778, 15796, 15813, 15869, 15926, 16009, 16084, 16086, 16088, 16090, 16092, 16097, 16102, 16107, 16112, 16129, 16146, 16163, 16182, 16247, 16312, 16378], 'all\_distinct': True, 'none\_equals\_parent': True}

附录：全部数值

正文为可读性只给透视表；这里逐单元给出改动前的全部字段（分位数、离散度、执行体实测等）。数值一个不少：明细放在报告主体内，而不是外链、折叠或附件——「可查」一旦要另外去找，就等于不可查

附录：延迟分位数与抖动

| 方案           | 并发度 | 执行体档位（微秒） | 中位数       | p95       | p99       | 相对离散度 |
|--------------|-----|-----------|-----------|-----------|-----------|-------|
| apscheduler  | 1   | 40        | 293.74 微秒 | 325.53 微秒 | 342.96 微秒 | 0.056 |
| apscheduler  | 1   | 300       | 559.04 微秒 | 588.66 微秒 | 590.43 微秒 | 0.031 |
| apscheduler  | 1   | 2700      | 2.989 毫秒  | 3.092 毫秒  | 3.173 毫秒  | 0.021 |
| apscheduler  | 1   | 10000     | 9.889 毫秒  | 10.123 毫秒 | 10.143 毫秒 | 0.011 |
| apscheduler  | 4   | 40        | 267.20 微秒 | 277.92 微秒 | 308.26 微秒 | 0.044 |
| apscheduler  | 4   | 300       | 526.01 微秒 | 708.75 微秒 | 823.48 微秒 | 0.142 |
| apscheduler  | 4   | 2700      | 3.003 毫秒  | 3.546 毫秒  | 3.733 毫秒  | 0.066 |
| apscheduler  | 4   | 10000     | 10.132 毫秒 | 10.472 毫秒 | 10.635 毫秒 | 0.015 |
| apscheduler  | 16  | 40        | 226.27 微秒 | 262.75 微秒 | 266.37 微秒 | 0.061 |
| apscheduler  | 16  | 300       | 484.41 微秒 | 511.45 微秒 | 538.63 微秒 | 0.032 |
| apscheduler  | 16  | 2700      | 2.977 毫秒  | 3.027 毫秒  | 3.029 毫秒  | 0.012 |
| apscheduler  | 16  | 10000     | 10.332 毫秒 | 10.686 毫秒 | 11.247 毫秒 | 0.022 |
| apscheduler  | 64  | 40        | 223.95 微秒 | 265.82 微秒 | 274.59 微秒 | 0.074 |
| apscheduler  | 64  | 300       | 507.24 微秒 | 538.33 微秒 | 538.99 微秒 | 0.029 |
| apscheduler  | 64  | 2700      | 2.949 毫秒  | 2.989 毫秒  | 3.029 毫秒  | 0.011 |
| apscheduler  | 64  | 10000     | 10.585 毫秒 | 10.720 毫秒 | 10.771 毫秒 | 0.027 |
| asyncio_pool | 1   | 40        | 145.49 微秒 | 177.59 微秒 | 207.40 微秒 | 0.114 |
| asyncio_pool | 1   | 300       | 389.99 微秒 | 421.09 微秒 | 427.24 微秒 | 0.035 |
| asyncio_pool | 1   | 2700      | 2.617 毫秒  | 2.638 毫秒  | 2.662 毫秒  | 0.006 |
| asyncio_pool | 1   | 10000     | 9.708 毫秒  | 9.924 毫秒  | 10.234 毫秒 | 0.015 |
| asyncio_pool | 4   | 40        | 153.05 微秒 | 172.10 微秒 | 184.08 微秒 | 0.097 |
| asyncio_pool | 4   | 300       | 428.07 微秒 | 462.16 微秒 | 472.08 微秒 | 0.040 |
| asyncio_pool | 4   | 2700      | 2.800 毫秒  | 2.932 毫秒  | 2.945 毫秒  | 0.020 |
| asyncio_pool | 4   | 10000     | 9.695 毫秒  | 9.778 毫秒  | 9.863 毫秒  | 0.007 |
| asyncio_pool | 16  | 40        | 138.37 微秒 | 147.94 微秒 | 148.85 微秒 | 0.054 |
| asyncio_pool | 16  | 300       | 395.42 微秒 | 411.17 微秒 | 413.02 微秒 | 0.021 |
| asyncio_pool | 16  | 2700      | 2.744 毫秒  | 2.800 毫秒  | 2.828 毫秒  | 0.012 |
| asyncio_pool | 16  | 10000     | 9.964 毫秒  | 10.041 毫秒 | 10.050 毫秒 | 0.004 |
| asyncio_pool | 64  | 40        | 120.73 微秒 | 126.95 微秒 | 128.50 微秒 | 0.023 |

| 方案           | 并发度 | 执行体档位 (微秒) | 中位数       | p95       | p99       | 相对离散度 |
|--------------|-----|------------|-----------|-----------|-----------|-------|
| asyncio_pool | 64  | 300        | 371.91 微秒 | 375.19 微秒 | 376.71 微秒 | 0.007 |
| asyncio_pool | 64  | 2700       | 2.832 毫秒  | 2.882 毫秒  | 2.895 毫秒  | 0.010 |
| asyncio_pool | 64  | 10000      | 10.189 毫秒 | 10.383 毫秒 | 10.530 毫秒 | 0.019 |
| bare_thread  | 1   | 40         | 143.75 微秒 | 165.27 微秒 | 169.11 微秒 | 0.067 |
| bare_thread  | 1   | 300        | 399.35 微秒 | 468.24 微秒 | 483.16 微秒 | 0.070 |
| bare_thread  | 1   | 2700       | 2.690 毫秒  | 2.770 毫秒  | 2.841 毫秒  | 0.017 |
| bare_thread  | 1   | 10000      | 9.552 毫秒  | 9.644 毫秒  | 9.659 毫秒  | 0.005 |
| bare_thread  | 4   | 40         | 158.85 微秒 | 178.14 微秒 | 191.10 微秒 | 0.060 |
| bare_thread  | 4   | 300        | 415.41 微秒 | 440.37 微秒 | 445.94 微秒 | 0.029 |
| bare_thread  | 4   | 2700       | 2.693 毫秒  | 2.741 毫秒  | 2.756 毫秒  | 0.008 |
| bare_thread  | 4   | 10000      | 9.872 毫秒  | 9.967 毫秒  | 10.315 毫秒 | 0.012 |
| bare_thread  | 16  | 40         | 171.89 微秒 | 174.70 微秒 | 176.22 微秒 | 0.016 |
| bare_thread  | 16  | 300        | 423.77 微秒 | 440.14 微秒 | 444.99 微秒 | 0.018 |
| bare_thread  | 16  | 2700       | 2.704 毫秒  | 2.732 毫秒  | 2.740 毫秒  | 0.005 |
| bare_thread  | 16  | 10000      | 9.972 毫秒  | 10.392 毫秒 | 10.652 毫秒 | 0.017 |
| bare_thread  | 64  | 40         | 170.05 微秒 | 175.16 微秒 | 176.83 微秒 | 0.016 |
| bare_thread  | 64  | 300        | 425.03 微秒 | 431.55 微秒 | 432.96 微秒 | 0.008 |
| bare_thread  | 64  | 2700       | 2.764 毫秒  | 2.788 毫秒  | 2.808 毫秒  | 0.006 |
| bare_thread  | 64  | 10000      | 9.526 毫秒  | 9.558 毫秒  | 9.578 毫秒  | 0.002 |
| process_pool | 1   | 40         | 336.43 微秒 | 360.56 微秒 | 368.37 微秒 | 0.047 |
| process_pool | 1   | 300        | 606.27 微秒 | 631.93 微秒 | 655.66 微秒 | 0.033 |
| process_pool | 1   | 2700       | 3.015 毫秒  | 3.702 毫秒  | 5.552 毫秒  | 0.198 |
| process_pool | 1   | 10000      | 9.702 毫秒  | 9.785 毫秒  | 9.839 毫秒  | 0.005 |
| process_pool | 4   | 40         | 192.86 微秒 | 251.09 微秒 | 278.11 微秒 | 0.172 |
| process_pool | 4   | 300        | 283.82 微秒 | 328.17 微秒 | 371.71 微秒 | 0.083 |
| process_pool | 4   | 2700       | 1.390 毫秒  | 1.547 毫秒  | 1.670 毫秒  | 0.054 |
| process_pool | 4   | 10000      | 4.608 毫秒  | 4.892 毫秒  | 5.049 毫秒  | 0.025 |
| process_pool | 16  | 40         | 161.53 微秒 | 206.04 微秒 | 231.93 微秒 | 0.146 |
| process_pool | 16  | 300        | 278.33 微秒 | 329.96 微秒 | 357.83 微秒 | 0.089 |
| process_pool | 16  | 2700       | 1.410 毫秒  | 1.508 毫秒  | 1.542 毫秒  | 0.035 |
| process_pool | 16  | 10000      | 4.636 毫秒  | 5.021 毫秒  | 5.550 毫秒  | 0.049 |
| process_pool | 64  | 40         | 196.54 微秒 | 237.84 微秒 | 243.91 微秒 | 0.097 |
| process_pool | 64  | 300        | 300.97 微秒 | 327.19 微秒 | 328.16 微秒 | 0.047 |
| process_pool | 64  | 2700       | 1.354 毫秒  | 1.384 毫秒  | 1.460 毫秒  | 0.023 |
| process_pool | 64  | 10000      | 4.629 毫秒  | 4.673 毫秒  | 4.729 毫秒  | 0.006 |
| thread_pool  | 1   | 40         | 74.84 微秒  | 101.46 微秒 | 112.12 微秒 | 0.149 |
| thread_pool  | 1   | 300        | 338.26 微秒 | 362.96 微秒 | 376.81 微秒 | 0.039 |
| thread_pool  | 1   | 2700       | 2.669 毫秒  | 2.698 毫秒  | 2.734 毫秒  | 0.007 |
| thread_pool  | 1   | 10000      | 9.577 毫秒  | 9.658 毫秒  | 9.749 毫秒  | 0.007 |
| thread_pool  | 4   | 40         | 95.11 微秒  | 103.23 微秒 | 106.77 微秒 | 0.053 |
| thread_pool  | 4   | 300        | 361.48 微秒 | 375.03 微秒 | 376.09 微秒 | 0.018 |
| thread_pool  | 4   | 2700       | 2.676 毫秒  | 2.763 毫秒  | 2.777 毫秒  | 0.016 |
| thread_pool  | 4   | 10000      | 9.867 毫秒  | 11.864 毫秒 | 12.865 毫秒 | 0.081 |

| 方案          | 并发度 | 执行体档位（微秒） | 中位数       | p95       | p99       | 相对离散度 |
|-------------|-----|-----------|-----------|-----------|-----------|-------|
| thread_pool | 16  | 40        | 92.31 微秒  | 97.34 微秒  | 98.25 微秒  | 0.039 |
| thread_pool | 16  | 300       | 353.51 微秒 | 362.47 微秒 | 375.59 微秒 | 0.023 |
| thread_pool | 16  | 2700      | 2.790 毫秒  | 2.832 毫秒  | 2.868 毫秒  | 0.010 |
| thread_pool | 16  | 10000     | 10.146 毫秒 | 10.195 毫秒 | 10.211 毫秒 | 0.005 |
| thread_pool | 64  | 40        | 89.40 微秒  | 91.59 微秒  | 92.21 微秒  | 0.019 |
| thread_pool | 64  | 300       | 355.32 微秒 | 389.22 微秒 | 457.84 微秒 | 0.069 |
| thread_pool | 64  | 2700      | 2.788 毫秒  | 2.843 毫秒  | 2.858 毫秒  | 0.012 |
| thread_pool | 64  | 10000     | 10.289 毫秒 | 10.354 毫秒 | 10.413 毫秒 | 0.012 |
| zoo         | 1   | 40        | 606.25 微秒 | 961.39 微秒 | 991.89 微秒 | 0.270 |
| zoo         | 1   | 300       | 867.95 微秒 | 1.228 毫秒  | 1.250 毫秒  | 0.252 |
| zoo         | 1   | 2700      | 3.263 毫秒  | 3.328 毫秒  | 3.388 毫秒  | 0.014 |
| zoo         | 1   | 10000     | 10.505 毫秒 | 10.893 毫秒 | 10.970 毫秒 | 0.021 |
| zoo         | 4   | 40        | 611.17 微秒 | 713.60 微秒 | 748.92 微秒 | 0.115 |
| zoo         | 4   | 300       | 866.98 微秒 | 898.40 微秒 | 902.02 微秒 | 0.041 |
| zoo         | 4   | 2700      | 3.307 毫秒  | 3.447 毫秒  | 3.485 毫秒  | 0.030 |
| zoo         | 4   | 10000     | 10.700 毫秒 | 10.837 毫秒 | 10.853 毫秒 | 0.012 |
| zoo         | 16  | 40        | 610.25 微秒 | 709.55 微秒 | 747.97 微秒 | 0.118 |
| zoo         | 16  | 300       | 878.71 微秒 | 976.84 微秒 | 1.017 毫秒  | 0.081 |
| zoo         | 16  | 2700      | 3.306 毫秒  | 3.405 毫秒  | 3.495 毫秒  | 0.029 |
| zoo         | 16  | 10000     | 10.596 毫秒 | 10.693 毫秒 | 11.836 毫秒 | 0.030 |
| zoo         | 64  | 40        | 565.92 微秒 | 759.15 微秒 | 837.97 微秒 | 0.170 |
| zoo         | 64  | 300       | 842.59 微秒 | 1.054 毫秒  | 1.138 毫秒  | 0.128 |
| zoo         | 64  | 2700      | 3.323 毫秒  | 3.551 毫秒  | 3.583 毫秒  | 0.035 |
| zoo         | 64  | 10000     | 10.636 毫秒 | 11.013 毫秒 | 11.146 毫秒 | 0.014 |

附录：框架自身开销占比

| 方案          | 并发度 | 执行体档位（微秒） | 框架开销      | 开销占比   | 执行体实测     |
|-------------|-----|-----------|-----------|--------|-----------|
| apscheduler | 1   | 40        | 236.46 微秒 | 80.50% | 57.28 微秒  |
| apscheduler | 1   | 300       | 257.58 微秒 | 46.08% | 301.46 微秒 |
| apscheduler | 1   | 2700      | 266.41 微秒 | 8.91%  | 2.723 毫秒  |
| apscheduler | 1   | 10000     | 336.85 微秒 | 3.41%  | 9.552 毫秒  |
| apscheduler | 4   | 40        | 211.85 微秒 | 79.29% | 55.35 微秒  |
| apscheduler | 4   | 300       | 208.91 微秒 | 39.72% | 317.10 微秒 |
| apscheduler | 4   | 2700      | 300.58 微秒 | 10.01% | 2.703 毫秒  |
| apscheduler | 4   | 10000     | —         | —      | 14.772 毫秒 |
| apscheduler | 16  | 40        | —         | —      | 53.66 微秒  |
| apscheduler | 16  | 300       | —         | —      | 299.58 微秒 |
| apscheduler | 16  | 2700      | —         | —      | 2.719 毫秒  |
| apscheduler | 16  | 10000     | —         | —      | 21.563 毫秒 |
| apscheduler | 64  | 40        | —         | —      | 50.45 微秒  |
| apscheduler | 64  | 300       | —         | —      | 304.01 微秒 |
| apscheduler | 64  | 2700      | —         | —      | 2.679 毫秒  |

| 方案           | 并发度 | 执行体档位（微秒） | 框架开销      | 开销占比   | 执行体实测     |
|--------------|-----|-----------|-----------|--------|-----------|
| apscheduler  | 64  | 10000     | —         | —      | 30.342 毫秒 |
| asyncio_pool | 1   | 40        | 96.90 微秒  | 66.60% | 48.59 微秒  |
| asyncio_pool | 1   | 300       | 100.32 微秒 | 25.72% | 289.67 微秒 |
| asyncio_pool | 1   | 2700      | 127.46 微秒 | 4.87%  | 2.490 毫秒  |
| asyncio_pool | 1   | 10000     | 146.90 微秒 | 1.51%  | 9.561 毫秒  |
| asyncio_pool | 4   | 40        | 99.09 微秒  | 64.75% | 53.96 微秒  |
| asyncio_pool | 4   | 300       | 106.69 微秒 | 24.92% | 321.38 微秒 |
| asyncio_pool | 4   | 2700      | 138.11 微秒 | 4.93%  | 2.662 毫秒  |
| asyncio_pool | 4   | 10000     | —         | —      | 9.919 毫秒  |
| asyncio_pool | 16  | 40        | —         | —      | 53.59 微秒  |
| asyncio_pool | 16  | 300       | —         | —      | 305.70 微秒 |
| asyncio_pool | 16  | 2700      | —         | —      | 2.620 毫秒  |
| asyncio_pool | 16  | 10000     | —         | —      | 24.461 毫秒 |
| asyncio_pool | 64  | 40        | —         | —      | 52.03 微秒  |
| asyncio_pool | 64  | 300       | —         | —      | 293.02 微秒 |
| asyncio_pool | 64  | 2700      | —         | —      | 2.703 毫秒  |
| asyncio_pool | 64  | 10000     | —         | —      | 40.859 毫秒 |
| bare_thread  | 1   | 40        | 92.74 微秒  | 64.52% | 51.01 微秒  |
| bare_thread  | 1   | 300       | 105.44 微秒 | 26.40% | 293.91 微秒 |
| bare_thread  | 1   | 2700      | 112.38 微秒 | 4.18%  | 2.578 毫秒  |
| bare_thread  | 1   | 10000     | 124.95 微秒 | 1.31%  | 9.427 毫秒  |
| bare_thread  | 4   | 40        | 107.74 微秒 | 67.82% | 51.12 微秒  |
| bare_thread  | 4   | 300       | 120.65 微秒 | 29.04% | 294.76 微秒 |
| bare_thread  | 4   | 2700      | 123.95 微秒 | 4.60%  | 2.569 毫秒  |
| bare_thread  | 4   | 10000     | —         | —      | 14.787 毫秒 |
| bare_thread  | 16  | 40        | —         | —      | 51.50 微秒  |
| bare_thread  | 16  | 300       | —         | —      | 295.16 微秒 |
| bare_thread  | 16  | 2700      | —         | —      | 2.578 毫秒  |
| bare_thread  | 16  | 10000     | —         | —      | 15.061 毫秒 |
| bare_thread  | 64  | 40        | —         | —      | 50.98 微秒  |
| bare_thread  | 64  | 300       | —         | —      | 296.18 微秒 |
| bare_thread  | 64  | 2700      | —         | —      | 2.627 毫秒  |
| bare_thread  | 64  | 10000     | —         | —      | 14.661 毫秒 |
| process_pool | 1   | 40        | 282.18 微秒 | 83.87% | 54.25 微秒  |
| process_pool | 1   | 300       | 299.03 微秒 | 49.32% | 307.25 微秒 |
| process_pool | 1   | 2700      | 342.47 微秒 | 11.36% | 2.673 毫秒  |
| process_pool | 1   | 10000     | 319.72 微秒 | 3.30%  | 9.383 毫秒  |
| process_pool | 4   | 40        | 99.69 微秒  | 51.69% | 93.17 微秒  |
| process_pool | 4   | 300       | —         | —      | 569.80 微秒 |
| process_pool | 4   | 2700      | —         | —      | 4.906 毫秒  |
| process_pool | 4   | 10000     | —         | —      | 17.803 毫秒 |
| process_pool | 16  | 40        | —         | —      | 91.27 微秒  |
| process_pool | 16  | 300       | —         | —      | 583.13 微秒 |

| 方案           | 并发度 | 执行体档位（微秒） | 框架开销      | 开销占比   | 执行体实测      |
|--------------|-----|-----------|-----------|--------|------------|
| process_pool | 16  | 2700      | —         | —      | 9.667 毫秒   |
| process_pool | 16  | 10000     | —         | —      | 52.111 毫秒  |
| process_pool | 64  | 40        | —         | —      | 89.02 微秒   |
| process_pool | 64  | 300       | —         | —      | 574.61 微秒  |
| process_pool | 64  | 2700      | —         | —      | 27.072 毫秒  |
| process_pool | 64  | 10000     | —         | —      | 160.095 毫秒 |
| thread_pool  | 1   | 40        | 30.68 微秒  | 40.99% | 44.16 微秒   |
| thread_pool  | 1   | 300       | 41.87 微秒  | 12.38% | 296.39 微秒  |
| thread_pool  | 1   | 2700      | 46.97 微秒  | 1.76%  | 2.622 毫秒   |
| thread_pool  | 1   | 10000     | 61.46 微秒  | 0.64%  | 9.515 毫秒   |
| thread_pool  | 4   | 40        | 46.09 微秒  | 48.45% | 49.03 微秒   |
| thread_pool  | 4   | 300       | 52.58 微秒  | 14.55% | 308.89 微秒  |
| thread_pool  | 4   | 2700      | 59.67 微秒  | 2.23%  | 2.616 毫秒   |
| thread_pool  | 4   | 10000     | —         | —      | 19.820 毫秒  |
| thread_pool  | 16  | 40        | —         | —      | 49.32 微秒   |
| thread_pool  | 16  | 300       | —         | —      | 298.88 微秒  |
| thread_pool  | 16  | 2700      | —         | —      | 2.704 毫秒   |
| thread_pool  | 16  | 10000     | —         | —      | 26.429 毫秒  |
| thread_pool  | 64  | 40        | —         | —      | 46.87 微秒   |
| thread_pool  | 64  | 300       | —         | —      | 299.35 微秒  |
| thread_pool  | 64  | 2700      | —         | —      | 2.683 毫秒   |
| thread_pool  | 64  | 10000     | —         | —      | 56.167 毫秒  |
| zoo          | 1   | 40        | 534.31 微秒 | 88.13% | 71.94 微秒   |
| zoo          | 1   | 300       | 539.47 微秒 | 62.15% | 328.48 微秒  |
| zoo          | 1   | 2700      | 556.01 微秒 | 17.04% | 2.707 毫秒   |
| zoo          | 1   | 10000     | 554.64 微秒 | 5.28%  | 9.951 毫秒   |
| zoo          | 4   | 40        | 547.26 微秒 | 89.54% | 63.91 微秒   |
| zoo          | 4   | 300       | 544.31 微秒 | 62.78% | 322.67 微秒  |
| zoo          | 4   | 2700      | 649.15 微秒 | 19.63% | 2.658 毫秒   |
| zoo          | 4   | 10000     | —         | —      | 14.948 毫秒  |
| zoo          | 16  | 40        | —         | —      | 65.98 微秒   |
| zoo          | 16  | 300       | —         | —      | 317.65 微秒  |
| zoo          | 16  | 2700      | —         | —      | 2.642 毫秒   |
| zoo          | 16  | 10000     | —         | —      | 14.780 毫秒  |
| zoo          | 64  | 40        | —         | —      | 55.17 微秒   |
| zoo          | 64  | 300       | —         | —      | 321.35 微秒  |
| zoo          | 64  | 2700      | —         | —      | 2.753 毫秒   |
| zoo          | 64  | 10000     | —         | —      | 15.234 毫秒  |

附录：吞吐与并发伸缩

| 方案          | 并发度 | 执行体档位（微秒） | 吞吐（任务/秒） |
|-------------|-----|-----------|----------|
| apscheduler | 1   | 40        | 3404.3   |
| apscheduler | 1   | 300       | 1788.8   |

| 方案           | 并发度 | 执行体档位（微秒） | 吞吐（任务/秒） |
|--------------|-----|-----------|----------|
| apscheduler  | 1   | 2700      | 334.5    |
| apscheduler  | 1   | 10000     | 101.1    |
| apscheduler  | 4   | 40        | 14970.0  |
| apscheduler  | 4   | 300       | 7604.4   |
| apscheduler  | 4   | 2700      | 1331.9   |
| apscheduler  | 4   | 10000     | 394.8    |
| apscheduler  | 16  | 40        | 70711.5  |
| apscheduler  | 16  | 300       | 33030.1  |
| apscheduler  | 16  | 2700      | 5375.2   |
| apscheduler  | 16  | 10000     | 1548.6   |
| apscheduler  | 64  | 40        | 285777.7 |
| apscheduler  | 64  | 300       | 126173.2 |
| apscheduler  | 64  | 2700      | 21702.3  |
| apscheduler  | 64  | 10000     | 6046.2   |
| asyncio_pool | 1   | 40        | 6873.2   |
| asyncio_pool | 1   | 300       | 2564.2   |
| asyncio_pool | 1   | 2700      | 382.1    |
| asyncio_pool | 1   | 10000     | 103.0    |
| asyncio_pool | 4   | 40        | 26136.1  |
| asyncio_pool | 4   | 300       | 9344.3   |
| asyncio_pool | 4   | 2700      | 1428.5   |
| asyncio_pool | 4   | 10000     | 412.6    |
| asyncio_pool | 16  | 40        | 115629.2 |
| asyncio_pool | 16  | 300       | 40462.9  |
| asyncio_pool | 16  | 2700      | 5831.5   |
| asyncio_pool | 16  | 10000     | 1605.9   |
| asyncio_pool | 64  | 40        | 530094.2 |
| asyncio_pool | 64  | 300       | 172085.1 |
| asyncio_pool | 64  | 2700      | 22599.7  |
| asyncio_pool | 64  | 10000     | 6281.4   |
| bare_thread  | 1   | 40        | 6956.3   |
| bare_thread  | 1   | 300       | 2504.1   |
| bare_thread  | 1   | 2700      | 371.7    |
| bare_thread  | 1   | 10000     | 104.7    |
| bare_thread  | 4   | 40        | 25180.5  |
| bare_thread  | 4   | 300       | 9629.0   |
| bare_thread  | 4   | 2700      | 1485.5   |
| bare_thread  | 4   | 10000     | 405.2    |
| bare_thread  | 16  | 40        | 93082.7  |
| bare_thread  | 16  | 300       | 37756.1  |
| bare_thread  | 16  | 2700      | 5917.6   |
| bare_thread  | 16  | 10000     | 1604.5   |
| bare_thread  | 64  | 40        | 376362.9 |
| bare_thread  | 64  | 300       | 150577.9 |

| 方案           | 并发度 | 执行体档位（微秒） | 吞吐（任务/秒） |
|--------------|-----|-----------|----------|
| bare_thread  | 64  | 2700      | 23154.0  |
| bare_thread  | 64  | 10000     | 6718.2   |
| process_pool | 1   | 40        | 2972.4   |
| process_pool | 1   | 300       | 1649.4   |
| process_pool | 1   | 2700      | 331.7    |
| process_pool | 1   | 10000     | 103.1    |
| process_pool | 4   | 40        | 20740.7  |
| process_pool | 4   | 300       | 14093.4  |
| process_pool | 4   | 2700      | 2878.6   |
| process_pool | 4   | 10000     | 868.1    |
| process_pool | 16  | 40        | 99052.0  |
| process_pool | 16  | 300       | 57486.6  |
| process_pool | 16  | 2700      | 11345.1  |
| process_pool | 16  | 10000     | 3451.0   |
| process_pool | 64  | 40        | 325635.1 |
| process_pool | 64  | 300       | 212645.4 |
| process_pool | 64  | 2700      | 47269.4  |
| process_pool | 64  | 10000     | 13826.1  |
| thread_pool  | 1   | 40        | 13362.6  |
| thread_pool  | 1   | 300       | 2956.3   |
| thread_pool  | 1   | 2700      | 374.6    |
| thread_pool  | 1   | 10000     | 104.4    |
| thread_pool  | 4   | 40        | 42055.6  |
| thread_pool  | 4   | 300       | 11065.7  |
| thread_pool  | 4   | 2700      | 1494.9   |
| thread_pool  | 4   | 10000     | 405.4    |
| thread_pool  | 16  | 40        | 173329.2 |
| thread_pool  | 16  | 300       | 45260.1  |
| thread_pool  | 16  | 2700      | 5735.0   |
| thread_pool  | 16  | 10000     | 1577.0   |
| thread_pool  | 64  | 40        | 715882.7 |
| thread_pool  | 64  | 300       | 180117.2 |
| thread_pool  | 64  | 2700      | 22956.6  |
| thread_pool  | 64  | 10000     | 6220.2   |
| zoo          | 1   | 40        | 1649.5   |
| zoo          | 1   | 300       | 1152.1   |
| zoo          | 1   | 2700      | 306.5    |
| zoo          | 1   | 10000     | 95.2     |
| zoo          | 4   | 40        | 6544.9   |
| zoo          | 4   | 300       | 4613.7   |
| zoo          | 4   | 2700      | 1209.4   |
| zoo          | 4   | 10000     | 373.8    |
| zoo          | 16  | 40        | 26218.8  |
| zoo          | 16  | 300       | 18208.4  |

| 方案  | 并发度 | 执行体档位（微秒） | 吞吐（任务/秒） |
|-----|-----|-----------|----------|
| zoo | 16  | 2700      | 4839.3   |
| zoo | 16  | 10000     | 1510.0   |
| zoo | 64  | 40        | 113090.9 |
| zoo | 64  | 300       | 75956.6  |
| zoo | 64  | 2700      | 19258.5  |
| zoo | 64  | 10000     | 6017.5   |