

PR #1673 完整报告

radixark/miles

[feat] support `sample` rollout submission granularity to keep fully async concurrency

合并时间: 2026-08-05 09:12

原文链接: <http://prhub.com.cn/radixark/miles/pull/1673>

执行摘要

- 一句话: 新增 sample 粒度提交回填, 消除 fully-async 拖尾组瓶颈
- 推荐动作: 值得精读。重点学习 SubmissionScheduler 的并发账目设计、异常路径 cancel-and-settle、per-driver 默认值, 以及 review 中关于 credit 守恒的辩论——对做异步调度系统的人有教科书级借鉴价值。

功能与动机

PR body 明确指出: 在 group-based RL (GRPO 风格) 长 horizon agentic rollout 中, 异步调度器以 group 粒度回填, 每个组都被其最慢的兄弟任务拖住, 引擎容量闲置。原文称 "In our production runs (n=8, 32 concurrent groups on SWE tasks) this straggler gating was a primary rollout-throughput limiter", 即 straggler 门控是生产环境的主要吞吐瓶颈。

实现拆解

1. 新增提交调度器抽象 miles/rollout/submission_scheduler.py: 包含 GroupLevelSubmission (保留原始 group 粒度语义, 逐字复刻旧行为) 和 SampleBackfillSubmission (样本完成 credit 计数器 + asyncio.Event 唤醒), make_submission_scheduler 按 --rollout-submission-granularity 和 driver 默认值解析。容量判定从 "在飞 group 数 < 预算" 改为 "在飞样本数 + group_size <= 预算 × group_size", 并内置孤儿 credit 重同步。
2. 在 generate_and_rm_group (miles/rollout/inference_rollout/inference_rollout_common.py) 中挂接 sample_done_callback: 每个样本子任务通过 task.add_done_callback 注册回调, 成功 / 异常 / 取消均触发以保证 credit 守恒; 异常路径先取消并 settle 全部兄弟任务再抛出, 避免孤儿生成与迟到回调错误扣账。
3. 两个 driver 接入: generate_rollout_async (inference_rollout_train.py, 默认 group) 与 FullyAsyncRolloutFn._worker_loop (fully_async_rollout.py, 默认 sample) 统一将提交循环改为 scheduler.has_capacity + scheduler.wait_for_progress; _submit_one_group 增加 on_submit 记账, submit_generate_tasks 透传 sample_done_callback。
4. 参数与文档: miles/utils/arguments.py 新增 enum 参数 --rollout-submission-granularity {group,sample}, 默认 None 表示由 driver 各自决默认; docs/user-guide/cli-reference.md 同步说明。

5. 测试配套：新增 tests/fast/rollout/inference_rollout/test_sample_completion_backfill.py，使用 Harness 手工控制 group/sample 完成时序，验证 sync 默认 group、回填替换组提前提交、差一个样本不提交、异常取消 conserve credits、调度器单元测试；更新 tests/fast/rollout/test_fully_async_rollout.py 覆盖 worker 默认 sample 粒度与显式 group opt-out。

关键文件：

- miles/rollout/submission_scheduler.py（模块 调度器；类别 source；类型 core-logic；符号 GroupLevelSubmission, SampleBackfillSubmission, make_submission_scheduler, has_capacity）：核心新增文件，定义两种提交调度策略与工厂方法，是本次变更的调度语义核心。
- miles/rollout/inference_rollout/inference_rollout_common.py（模块 公共逻辑；类别 source；类型 core-logic；符号 generate_and_rm_group）：关键行为改造点：挂接样本完成回调并处理异常路径，保证 credit 守恒，是整个回填机制的账本基础。
- miles/rollout/inference_rollout/inference_rollout_train.py（模块 同步驱动；类别 source；类型 core-logic；符号 submit_generate_tasks, generate_rollout_async）：sync rollout 驱动接入调度器，默认 group 粒度，并新增 submit_generate_tasks 的 callback 透传。
- miles/rollout/fully_async_rollout.py（模块 异步驱动；类别 source；类型 core-logic；符号 FullyAsyncRolloutFn, _submit_one_group, _worker_loop, _generate_group）：fully-async driver 默认 sample 粒度，是本次优化主要受益方；worker 循环改为调度器驱动。
- miles/utils/arguments.py（模块 参数解析；类别 source；类型 configuration）：新增 --rollout-submission-granularity 参数并说明 per-driver 默认值，是 CLI 面入口。
- tests/fast/rollout/inference_rollout/test_sample_completion_backfill.py（模块 回填测试；类别 test；类型 test-coverage；符号 Harness, make_args, make_group, fake_submit_generate_tasks）：新增 285 行测试，用 Harness 手工控制完成时序，覆盖回填时机与边界行为，是本次行为验证主体。
- tests/fast/rollout/test_fully_async_rollout.py（模块 异步测试；类别 test；类型 test-coverage；符号 default_generate, blocking_generate, failing_generate, test_worker_defaults_to_sample_granularity）：更新 fully-async 测试 stub 并新增默认 sample 粒度与 group opt-out 两个用例。
- docs/user-guide/cli-reference.md（模块 用户文档；类别 docs；类型 documentation）：CLI 参考文档补充新参数说明。

关键符号：make_submission_scheduler, SampleBackfillSubmission.has_capacity, SampleBackfillSubmission.sample_done_callback, SampleBackfillSubmission.wait_for_progress, GroupLevelSubmission.has_capacity, generate_and_rm_group, submit_generate_tasks, generate_rollout_async, FullyAsyncRolloutFn._worker_loop, FullyAsyncRolloutFn._submit_one_group

关键源码片段

miles/rollout/submission_scheduler.py

核心新增文件，定义两种提交调度策略与工厂方法，是本次变更的调度语义核心。

```
"""Paces prompt-group submission for the rollout drivers (--rollout-submission-granularity).
```

```
本模块只负责“何时允许提交新组”；在飞 group 任务集合由 driver 持有，  
通过 has_capacity 的 pending_groups 参数传入。调度器只记录在飞样本数。
```

```
"""
```

```
import asyncio
```

```
import logging
```

```
from argparse import Namespace
```

```
from miles.utils.types import Sample
```

```
logger = logging.getLogger(__name__)
```

```
class GroupLevelSubmission:
```

```
    """A submission slot frees only when the whole group task returns."""
```

```
    sample_done_callback = None # sample 粒度专属，group 粒度不接线
```

```
    def has_capacity(self, *, pending_groups: int, group_budget: int) -> bool:
```

```
        # 原始语义：在飞 group 数小于预算即可继续提交
```

```
        return pending_groups < group_budget
```

```
    def on_submit(self, groups: list[list[Sample]]) -> None:
```

```
        pass
```

```
    async def wait_for_progress(self, pendings: set[asyncio.Task]) -> tuple[set[asyncio.Task],  
set[asyncio.Task]]:
```

```
        # group 粒度下，进度只由 group 任务完成驱动
```

```
        return await asyncio.wait(pendings, return_when=asyncio.FIRST_COMPLETED)
```

```
class SampleBackfillSubmission:
```

```
    """Each finished sample frees its own slot; a replacement group fits once ``group_size``  
samples complete."""
```

```
    def __init__(self, group_size: int):
```

```
        self.group_size = group_size # 每组样本数，攒够这些 credit 释放一个替换名额
```

```
        self.samples_in_flight = 0 # 在飞样本计数，用于容量判定
```

```
        self._sample_done = asyncio.Event() # 样本完成唤醒事件
```

```
    def sample_done_callback(self) -> None:
```

```
        # 每个样本任务结束（成功 / 异常 / 取消）恰好触发一次，保证在飞计数守恒
```

```
        self.samples_in_flight -= 1
```

```
        self._sample_done.set()
```

```

def has_capacity(self, *, pending_groups: int, group_budget: int) -> bool:
    """A False return arms the sample wakeup: only later completions wake ``wait_for_
    progress``."""
    if pending_groups == 0 and self.samples_in_flight:
        # 兜底重同步: group 返回但从未生成样本任务 (如 stub 跳过 callback) 时,
        # 这些 credit 永远不会被归还, 会永久卡死容量, 此处直接清零
        logger.warning(f"samples_in_flight={self.samples_in_flight} with no pending groups;
        resetting to 0")
        self.samples_in_flight = 0
    if self.samples_in_flight + self.group_size <= group_budget * self.group_size:
        return True
    # 容量不足时清除事件, 消除 lost-wakeup 窗口
    self._sample_done.clear()
    return False

def on_submit(self, groups: list[list[Sample]]) -> None:
    self.samples_in_flight += sum(len(group) for group in groups)

async def wait_for_progress(self, pendings: set[asyncio.Task]) -> tuple[set[asyncio.Task],
set[asyncio.Task]]:
    # 同时等待 group 完成或单个样本完成, 任一发生即返回
    waiter = asyncio.create_task(self._sample_done.wait())
    try:
        done, pending = await asyncio.wait(pendings | {waiter}, return_when=asyncio.FIRST_
        COMPLETED)
    finally:
        waiter.cancel()
    return done - {waiter}, pending - {waiter}

def make_submission_scheduler(args: Namespace, *, default: str) -> GroupLevelSubmission |
SampleBackfillSubmission:
    # 显式参数优先, 否则按 driver 各自默认: fully-async 为 sample, sync 为 group
    granularity = args.rollout_submission_granularity or default
    if granularity == "group":
        return GroupLevelSubmission()
    assert granularity == "sample", f"unknown submission granularity: {granularity}"
    return SampleBackfillSubmission(args.n_samples_per_prompt)

```

miles/rollout/inference_rollout/inference_rollout_common.py

关键行为改造点: 挂接样本完成回调并处理异常路径, 保证 credit 守恒, 是整个回填机制的账本基础。

```

async def generate_and_rm_group(
    state: GenerateState,
    group: list[Sample],
    sampling_params: dict[str, Any],
    evaluation: bool = False,
    sample_done_callback: Callable[[], None] | None = None,

```

```

) -> list[Sample]:
    args = state.args

    if state.aborted:
        return group

    if policy_uses_routing_key(args):
        for sample in group:
            if sample.routing_key is None:
                sample.routing_key = str(uuid.uuid4())

    log_prefix = f"[group indices={getattr(s, 'index', '?') for s in group}]"
    logger.debug(f"{log_prefix} Starting group with {len(group)} samples")
    tasks = []
    for idx, sample in enumerate(group):
        current_sampling_params = sampling_params.copy()
        if getattr(args, "sglang_enable_deterministic_inference", False):
            current_sampling_params["sampling_seed"] = args.rollout_seed + idx
        task = asyncio.create_task(generate_and_rm(state, sample, current_sampling_params,
            evaluation=evaluation))
        if sample_done_callback is not None:
            # 成功、异常、取消三种结局都触发 callback: 每个最终结束的样本都对应一次容量释放,
            # 失败样本也会释放引擎额度, 跳过它们会导致并发额度单调泄漏
            task.add_done_callback(lambda _task: sample_done_callback())
        tasks.append(task)

    try:
        group = await asyncio.gather(*tasks)
    except BaseException:
        # 单个样本失败时 gather 会传播第一个异常, 但兄弟任务仍在运行:
        # 它们既浪费引擎资源, 其迟到 callback 又可能在调度器重同步后错误扣减新组计数。
        # 先取消并等待所有样本任务 settle, 保证抛异常时账目已经精确
        for task in tasks:
            task.cancel()
        await asyncio.gather(*tasks, return_exceptions=True)
        raise
    logger.debug(f"{log_prefix} [group] All {len(group)} samples completed")
    if state.aborted:
        return group

    if args.group_rm:
        await batched_async_rm(args, group, inplace_set_reward_field=True)

    return group

```

评论区精华

1. guapisolo 在 issue 中提问为何不做直接 SampleLevelSubmission 而用回填: 担心大 group (64/128) 下 group 粒度回填仍不够细。结论: 以 credit 聚合的回填在保留组提交

结构的同时获得样本级释放，后续可演进。

2. guapisolo 提出 Codex 边缘情况：一个样本异常 → `asyncio.gather` 提前退出 → 兄弟任务仍在跑 → `pendings` 变空 → `has_capacity` 重置计数 → 迟到回调错误扣减替换组。最终由 commit 42bf672f 在异常路径 `cancel-and-settle` 解决。
 3. gemini-code-assist 建议只在成功时触发 `sample_done_callback`，ys-2020 反驳：成功 / 异常 / 取消都触发才能精确守恒并发额度，跳过失败样本会导致额度单调衰减。
 4. gemini-code-assist 提出数据源耗尽死锁，ys-2020 确认在 f814b44 修复。
 5. gemini-code-assist 建议 group 异常后立即 `submit_groups(1)` 补位，ys-2020 反驳：样本回调已为失败组返回 `n` 个 `credit`，再补会 `double-replenish`。
 6. nblintao 询问是否未来默认开启 `sample` 粒度，yueming-yuan 同意，后续演进为 `per-driver` 默认。
- 为何选择 `SampleBackfill` 而非直接 `SampleLevelSubmission` (design): 以 `credit` 聚合的回填在保持 group 提交结构的同时获得样本级释放，后续可演进为更细粒度；issue 评论中 yueming-yuan 说明调度器抽象让两种策略仅差两个决策点。
 - 异常引起的迟到回调是否会错误扣减替换组计数 (correctness): commit 42bf672f 在异常路径 `cancel-and-settle` 全部兄弟任务后再抛异常，保证账目在 `reap` 时精确。
 - 失败 / 取消的样本是否应触发 `done callback` (correctness): 保持 `callback` 对成功 / 异常 / 取消都触发，并补充注释说明不变量。
 - 数据源耗尽时死锁 (correctness): ys-2020 确认在 f814b44 修复： `submit_groups` 返回 0 且无在飞任务时跳出循环。
 - group 异常后是否需要立即补充提交 (design): 保持纯 `credit` 驱动回填，不额外补位。
 - 是否将 `sample` 粒度设为默认 (design): 后续演进为 `per-driver` 默认： `fully-async` 默认 `sample`， `sync` 默认 `group`。

风险与影响

- 风险：
 1. 并发账目精度敏感： `SampleBackfillSubmission` 的 `credit` 计数依赖回调恰好触发一次，曾出现孤儿 `credit` 死锁 (commit 45705738 修复， `stage-a-cpu` 挂到 60 分钟超时)，当前兜底逻辑依赖 `pending_groups == 0` 的时序判断， `driver` 改动若绕过该路径可能回归。
 2. `generate_and_rm_group` 异常路径复杂度上升： `cancel-and-settle` 与 `group_rm`、`aborted` 路径交织，异常时 `batched_async_rm` 不会执行，行为与旧版不同。
 3. `fully-async` 默认行为变更：存量用户未设参数时在飞 `group` 上限放大最多 `n_samples_per_prompt` 倍，对数据源、reward 模型和 SGLang 引擎负载有隐性压力。
 4. 作用域限定：仅 `class-based rollout API` (`MILES_EXPERIMENTAL_ROLLOUT_REFACTOR=1`) 生效，默认 `generate_rollout` 路径不受影响，降低回归面。
 5. 测试缺口：CPU 测试覆盖时序逻辑，但无真实 GPU straggler 场景的 e2e 验证。 - 影响：对用户： `fully-async` / 长 horizon agentic 训练用户默认获得样本级回填，引擎利用率提升； `sync` 用户需显式 `opt-in` 并理解 `abort` 丢弃语义。对系统：并发上限从

rollout_batch_size 扩展至约 $\text{rollout_batch_size} \times \text{n_samples_per_prompt}$ ，对数据源与 reward 模型并发压力上升。对团队：调度策略收敛为可插拔抽象，后续新策略成本低，并发不变量 " 每个样本任务恰好一次回调 " 被文档化。 - 风险标记：并发账目精度敏感，fully-async 默认行为变更，异常路径复杂度上升，仅实验 API 生效

关联脉络

- PR #1717 Rewrite fully-async rollout as FullyAsyncRolloutFn on the class-based rollout API: 本 PR 在 issue 评论中明确说明 rebase 到 #1717 之上，并按 FullyAsyncRolloutFn 类式 API 重构实现，两个 PR 是同一演进主线。
- PR #1716 Move fully-async rollout from examples into miles/rollout: fully-async rollout 迁入核心库的前奏，本 PR 继续在该路径上扩展调度能力。
- PR #1740 feat (async): support evaluation for fully-async training (dedicated fleet / pause-the-world / external service): 本 PR 合并 main 时解决了其引入的 eval-pause 测试 stub 冲突 (commit acba7df0 说明)，两 PR 均修改 fully_async_rollout.py。