PR #1965 完整报告

radixark/miles

dashboard: fix phase visibility for manager events and idle processes

合并时间：2026-07-30 10:49

原文链接：http://prhub.com.cn/radixark/miles/pull/1965

执行摘要

- 一句话：修复 dashboard 阶段可见性与空闲事件滞留
- 推荐动作：值得精读，尤其是 store.py 中 open 区间语义的处理和 hooks.py 中后台 flush 线程的设计：前者体现了“事件半开区间 + 渲染端负责延伸”的约定，后者解决批处理 sink 在静默进程上的滞留问题。建议后续为这三类场景补充针对性回归测试（open manager 事件、空 gpus 拓扑、空闲进程 flush），并将 t1=-1 的渲染约定写入模块文档。

功能与动机

PR body 明确指出：“Two timeline bugs that together made a fully-async run's step-0 eval invisible and mislabeled”——utilization 视图把初始权重同步的 finalize_and_resume_engines / update_weights_implementation 画满整个 eval 期，而 eval 本身完全不显示。根因有三：manager 角色的 open 事件被 t1=-1 的裁剪逻辑丢弃；外部引擎 gpus:[] 导致覆盖集为空；进程空闲时 PhaseSink 不冲刷批量缓冲区。本 PR 的目标是让 fully-async 场景下阶段事件与真实执行周期对齐。

实现拆解

1. 修正 manager 角色 open 区间的窗口裁剪（miles/dashboard/store.py）：原逻辑对 t1=-1 的 open 事件直接计算 clip1 = min(event.t1, window["t1"])，导致 clip0 >= clip1 恒成立、事件被跳过。现改为对 event.t1 < 0 走单独分支：窗口右边界开放时保持 -1.0，由渲染器延伸到“现在”，与 train 角色事件的 open 语义保持一致。
2. 空覆盖集回退到引擎节点 lane（miles/dashboard/store.py）：外部 worker 拓扑快照携带 gpus: [] 时，covered 集合为空，manager 事件无法扩散到任何 lane。新增回退逻辑：从 engine["addr"] 解析节点名，再用 self.lanes() 取该节点上全部已知 (node, gpu) 作为覆盖集。
3. PhaseSink 增加后台冲刷线程（miles/dashboard/hooks.py）：PhaseSink 原本只在有新事件到达时调用 _take_batch_if_due() 检查批次，空闲进程（如 shared-engine eval 期间的 train rank）的最后一批关闭事件会无限滞留。新增 _ensure_flusher()，在首个事件落地后启动 daemon 线程 _loop，每 BATCH_MAX_SECONDS 检查 buffer 非空即调用 flush()。
4. 验证与测试：在 GLM-5.2 16x-GB300 fully-async 运行上验证，serve 重启后前两个修复生效，eval_rollout 在所有 12 条 engine lane 上渲染正确；tests/fast/dashboard 185 个用例全部通过。本 PR 未新增专门针对这三个场景的测试文件。

关键文件：

- miles/dashboard/hooks.py（模块 仪表盘；类别 source；类型 core-logic；符号 _ensure_flusher, _loop）：为 PhaseSink 增加后台守护线程 _ensure_flusher/_loop，解决空闲进程最后一批 phase 关闭事件滞留缓冲区、UI 上长期显示为未关闭进度条的问题，是本次三大修复之一。
- miles/dashboard/store.py（模块 仪表盘；类别 source；类型 core-logic）：修复 manager 角色 open 事件在窗口裁剪时被丢弃的问题，并在引擎无 GPU 身份（gpus:[]）时回退到引擎节点上的已知 lane，是 eval 阶段可见性的核心修复。

关键符号：_ensure_flusher, _loop

关键源码片段

miles/dashboard/hooks.py

为 PhaseSink 增加后台守护线程 _ensure_flusher/_loop，解决空闲进程最后一批 phase 关闭事件滞留缓冲区、UI 上长期显示为未关闭进度条的问题，是本次三大修复之一。

```
# miles/dashboard/hooks.py —— PhaseSink 空闲冲刷机制
# 原先只有新事件到达时才调用 _take_batch_if_due()，
# 进程一旦空闲，最后一批关闭事件会一直滞留在 buffer 中。
class PhaseSink:
    def __call__(self, name: str, t0: float, t1: float) -> None:
        try:
            with self._lock:
                # lazy: torch.distributed 通常尚未初始化，持续重解析直到真实 rank 出现
                if self._identity is None or (self._identity.rank < 0 and self.role == Role.TRAIN):
                    self._identity = _resolve_identity()
                identity = self._identity
                self._buffer.append(
                    PhaseEvent(name=name, t0=t0, t1=t1,
                               node=identity.node, gpus=identity.gpus,
                               rank=identity.rank, role=self.role)
                )
                batch = self._take_batch_if_due()
            # 首个事件落地后启动后台线程，保证空闲期也能 flush
            self._ensure_flusher()
            if batch:
                self._handle.push_phases.remote(batch)
        except Exception:
            self._warner.warn("dashboard phase sink failed; dropping events")

    def _ensure_flusher(self) -> None:
        """后台冲刷线程：进程静默时也能把关闭事件推给 collector。"""
        if self._flusher is not None:
            return
        import threading

        def _loop():
```

```
            while True:
                time.sleep(BATCH_MAX_SECONDS)
                with self._lock:
                    due = bool(self._buffer)
                if due:
                    # flush() 会把整个 buffer 通过 push_phases.remote 发出
                    self.flush()

        self._flusher = threading.Thread(target=_loop, daemon=True, name="dashboard-phase-
        flush")
        self._flusher.start()
```

miles/dashboard/store.py

修复 manager 角色 open 事件在窗口裁剪时被丢弃的问题，并在引擎无 GPU 身份（gpus:[]）时回退到引擎节点上的已知 lane，是 eval 阶段可见性的核心修复。

```
# miles/dashboard/store.py —— manager 角色事件扩散到 engine lane 的核心分支
# manager 是 GPU-less 驱动进程，事件需要按拓扑窗口扩散到所有引擎 GPU
if event.role == Role.ROLLOUT_MANAGER:
    for window in windows:
        clip0 = max(event.t0, window["t0"])
        if event.t1 < 0:
            # 尚未关闭的区间：clip1 取窗口右边界，
            # 窗口本身开放时保持 -1.0，由渲染器画到“现在”
            clip1 = -1.0 if window["t1"] is None else window["t1"]
        else:
            clip1 = event.t1 if window["t1"] is None else min(event.t1, window["t1"])
        if clip1 >= 0 and clip0 >= clip1:
            continue

        covered = {(node, gpu) for engine in window["engines"] for node, gpu in engine["gpus"]}
        if not covered:
            # 外部引擎注册时没有 GPU 身份（gpus=[]），空覆盖集导致事件全部丢失；
            # 回退到引擎节点上所有已知 lane，节点名从 engine["addr"] 解析
            engine_nodes = {engine["addr"].split("//")[-1].split(":")[0] for engine in window["engines"]}
            covered = {(lane["node"], lane["gpu"]) for lane in self.lanes() if lane["node"] in engine_
            nodes}
        if lanes is not None:
            covered &= lanes
        for node, gpu in sorted(covered):
            out.append(dict(name=event.name, t0=clip0, t1=clip1,
                            node=node, gpu=gpu, rank=event.rank, role=event.role))
```

评论区精华

该 PR 的 review 通道没有产生实质技术讨论：Zhichenzzz 直接 APPROVED，未留下任何评论。唯一评论来自 gemini-code-assist[bot] 的 issue 级通知，说明 Gemini Code Assist 消费者版已停用，与本次变更新无内容关联，可忽略。

- 暂无高价值评论线程

风险与影响

- 风险：
  1. 后台线程生命周期：每个 PhaseSink 实例（每个进程一个）都会启动一个 daemon 线程并常驻进程生命周期，多进程场景下线程数量较多，虽然每 BATCH_MAX_SECONDS 才唤醒一次、开销很小，但没有显式关闭机制。
  2. open 区间语义依赖渲染端：store.py 对 t1 < 0 的事件直接透传 -1.0，要求渲染端始终把 t1 < 0 解释为“延伸到 now”，未来渲染逻辑调整时需保持兼容。
  3. 节点解析依赖地址格式：engine["addr"].split("//")[-1].split(":")[0] 依赖 //host:port 格式，IPv6 或地址格式变化会导致解析失败或落到错误节点。
  4. 测试覆盖不足：仅依赖现有 185 个 dashboard 用例，没有为 open 区间、空 gpus 拓扑、空闲 flush 三个场景新增回归测试，后续回归风险较高。 - 影响：用户侧：fully-async 运行下 dashboard 时间线能正确展示 eval / weight-sync 阶段，避免把权重同步误判为长期占用的 eval 过程，提升训练监控可信度。系统侧：PhaseSink 增加守护线程，带来轻微 CPU 与线程开销；manager 与 train 角色的事件语义统一为“open 事件延伸到 now”。团队侧：dashboard 模块对开放区间与空闲刷新的处理成为后续观测功能的基础，相关约定值得沉淀进模块文档。 - 风险标记：守护线程常驻，open 区间语义变更，依赖引擎地址格式，缺少针对性测试

关联脉络

- 暂无明显关联 PR