PR #2499 完整报告

radixark/miles

feat(ci): add weekly full-suite cadence

合并时间：2026-08-14 02:36

原文链接：http://prhub.com.cn/radixark/miles/pull/2499

执行摘要

- 一句话：每周六新增全量 CI 节奏，GPU 矩阵限流至单 runner
- 推荐动作：CI 工具链与基础设施负责人建议精读，特别是 tests/ci/ci_policy.py 中 " 每个精确 cron 显式映射独立 cadence" 的设计原则和 RunPolicy 字段化改造，这两点对后续扩展 CI 节奏、避免新 cadence 静默继承 nightly 行为很有参考价值。普通产品开发者只需了解每周六存在全量 CI 运行即可，无需深入源码。

功能与动机

PR body 明确指出："Periodic full-suite coverage is needed for long and otherwise label-gated tests, but it must consume only a small slice of self-hosted GPU capacity instead of monopolizing the fleet." 即长耗时与 label-gated 测试缺少周期性全量覆盖窗口，而全量跑又必须限制在自托管 GPU 集群的小份额内，避免与日常 PR 队列争抢资源。

实现拆解

1. 策略层改造（tests/ci/ci_policy.py）：新增 WEEKLY_CADENCE 常量，将 SCHEDULE_POLICIES 从单条 cron `0 15 * * *` 拆分为 `0 15 * * 0-5`（nightly）与 `0 15 * * 6`（weekly）两条精确映射；RunPolicy 删除 is_nightly property，改为 admit_nightly_tests、bypass_fastfail、write_baseline 三个显式字段，其中 weekly 与 nightly 同步置 True；resolve_policy 分支顺序调整为 run-ci-all / weekly > nightly > run-ci-image，weekly 的 include 集合等于全部 KNOWN_LABELS（含 long、ft-long）。
2. 运行层语义泛化（tests/ci/run_suite.py、tests/ci/ci_utils.py）：filter_tests 的参数 nightly 更名为 admit_nightly_tests，run_a_suite 中 gate_nightly 改为 gate_write_baseline=policy.write_baseline，使 baseline 写入不再与 nightly 名称耦合；run_gate_hook 参数 nightly 更名为 write_baseline，日志由动态详情 (trusted=..., len(value(s))) 收敛为静态摘要 [CI Gate][baseline] ...: wrote baseline，持久化行为不变。
3. 工作流层（.github/workflows/pr-test.yml、pr-test-rocm.yml）：schedule 拆分为周日到周五 nightly + 周六 weekly 两条 cron；各 GPU stage 的 strategy 增加 max-parallel: ${{ needs.resolve-ci-policy.outputs.cadence == 'weekly' && 1 || N }}，weekly 时每矩阵只占用 1 个 runner，其余节奏维持原并行度（2/2/3/2）。
4. 测试与文档配套：test_ci_policy.py 新增 test_weekly_schedule_resolves_to_independent_full_policy；test_run_suite.py 新增 test_scheduled_runs_use_utc_1500、test_weekly_serializes_each_gpu_matrix、test_weekly_full_scope_admits_nightly_only_

tests、test_weekly_policy_reaches_cuda_runner 等锁定测试；test_gate_integration.py 将 TestNightlyWrite 更名为 TestBaselineWrite 并全部改用 write_baseline 参数；docs/ci/00-stage.md、01-label.md、03-metric-history-gate.md 与 contributor-guide 同步描述新节奏。

关键文件：

- tests/ci/ci_policy.py（模块 策略解析；类别 source；类型 core-logic；符号 WEEKLY_CADENCE, SCHEDULE_POLICIES, RunPolicy, resolve_policy）：CI 策略解析核心：新增 WEEKLY_CADENCE、拆分 SCHEDULE_POLICIES、RunPolicy 字段化（admit_nightly_tests / write_baseline），是本次变更的策略源头
- tests/ci/ci_utils.py（模块 门禁钩子；类别 source；类型 core-logic；符号 run_gate_hook, run_unittest_files）：gate hook 的 write_baseline 语义落地处，也是 CodeQL 日志告警所在文件；参数从 nightly 泛化为 write_baseline 后 nightly 与 weekly 均可持续化基线
- .github/workflows/pr-test.yml（模块 CI 编排；类别 infra；类型 infrastructure）：NVIDIA 侧 schedule 拆分为 nightly/weekly 两条 cron，并在各 GPU stage 通过 max-parallel 条件表达式对 weekly 限流
- tests/ci/run_suite.py（模块 套件调度；类别 source；类型 core-logic；符号 filter_tests, run_a_suite）：filter_tests 参数 nightly 更名为 admit_nightly_tests，run_a_suite 改用 policy.write_baseline 传递 gate 写库信号
- tests/ci/test/test_run_suite.py（模块 套件测试；类别 test；类型 test-coverage；符号 test_scheduled_runs_use_utc_1500, test_weekly_serializes_each_gpu_matrix, test_pr_nightly_and_dispatch_share_policy, test_pr_schedules_and_dispatch_share_policy）：新增 6 个锁定测试：UTC 1500 定时、GPU 矩阵 weekly 串行化、weekly 全量放行 nightly-only、weekly 策略贯通到 CUDA runner 等，是本次行为契约的最强保障
- tests/ci/test/test_gate_integration.py（模块 门禁测试；类别 test；类型 test-coverage；符号 TestBaselineWrite）：TestNightlyWrite 更名为 TestBaselineWrite，所有调用点从 nightly 参数迁移到 write_baseline，验证写库语义与 cadence 解耦
- .github/workflows/pr-test-rocm.yml（模块 ROCm 编排；类别 infra；类型 infrastructure）：ROCm 侧同步拆分 schedule 并为 stage-c-4-gpu-mi300x 增加 weekly max-parallel 限流，保持与 NVIDIA 侧一致
- tests/ci/test/test_ci_policy.py（模块 策略测试；类别 test；类型 test-coverage；符号 test_weekly_schedule_resolves_to_independent_full_policy）：新增 test_weekly_schedule_resolves_to_independent_full_policy，验证周六 cron 解析到独立 weekly 策略且 admit_nightly_tests/bypass_fastfail/write_baseline 全为 True
- tests/ci/ci_register.py（模块 CI 注册表；类别 source；类型 refactor）：注册语义文档更新：nightly=True 的测试由 nightly 与 weekly 共同放行，明确 cadence 门控范围
- docs/ci/00-stage.md（模块 CI 文档；类别 docs；类型 documentation）：CI 阶段文档同步描述 nightly/weekly 双节奏与 GPU 限流策略，是贡献者理解新节奏的入口文档

关键符号：resolve_policy, resolve_workflow_inputs, filter_tests, run_a_suite, run_gate_hook, run_unittest_files

关键源码片段

tests/ci/ci_policy.py

CI 策略解析核心：新增 WEEKLY_CADENCE、拆分 SCHEDULE_POLICIES、RunPolicy 字段化（admit_nightly_tests / write_baseline），是本次变更的策略源头

```
# tests/ci/ci_policy.py —— CI 节奏与选择策略核心（PR#2499 变更后）

REGULAR_CADENCE = "regular"
NIGHTLY_CADENCE = "nightly"
WEEKLY_CADENCE = "weekly" # 新增：周六全量节奏
CI_CADENCES = frozenset({REGULAR_CADENCE, NIGHTLY_CADENCE, WEEKLY_CADENCE})

# 每个精确 cron 都必须显式映射到独立 cadence，
# 防止未来新增定时任务静默继承 nightly 行为。
SCHEDULE_POLICIES: dict[str, tuple[str, tuple[str, ...]]] = {
    "0 15 * * 0-5": (NIGHTLY_CADENCE, ()), # 周日到周五 15:00 UTC
    "0 15 * * 6": (WEEKLY_CADENCE, ()), # 周六 15:00 UTC 全量
}


@dataclass(frozen=True)
class RunPolicy:
    cadence: str
    include_labels: frozenset[str]
    admit_nightly_tests: bool # 是否放行 nightly-only 注册
    bypass_fastfail: bool # 失败后继续跑完剩余文件，还是立即停止
    write_baseline: bool # 是否把 gate 结果写入指标基线


def resolve_policy(cadence: str, raw_labels: set[str]) -> RunPolicy:
    # 先校验 cadence 合法性，并限制 nightly label 只能用于 nightly 节奏
    if cadence not in CI_CADENCES:
        raise ValueError(f"Unknown CI cadence {cadence!r}; expected one of {sorted(CI_
        CADENCES)}")
    if "nightly" in raw_labels and cadence != NIGHTLY_CADENCE:
        raise ValueError("The nightly workflow label requires cadence='nightly'")

    # 分支顺序即优先级：run-ci-all / weekly > nightly > run-ci-image
    requested = strip_run_ci_prefix(raw_labels) & set(KNOWN_LABELS)
    if "run-ci-all" in raw_labels or cadence == WEEKLY_CADENCE:
        scope = set(KNOWN_LABELS) # weekly 全量：包含 long / ft-long 等全部 label
    elif cadence == NIGHTLY_CADENCE:
        scope = set(KNOWN_LABELS) - {"long", "ft-long"}
    elif "run-ci-image" in raw_labels:
        scope = set(KNOWN_LABELS) - {"long", "ft-short", "ft-long"}
```

```
    else:
        scope = set()

    # 显式请求的 label 最后并入，保证显式指定优先于范围减法
    return RunPolicy(
        cadence=cadence,
        include_labels=frozenset(scope | requested),
        admit_nightly_tests=cadence in {NIGHTLY_CADENCE, WEEKLY_CADENCE},
        bypass_fastfail=cadence in {NIGHTLY_CADENCE, WEEKLY_CADENCE} or "bypass-fastfail"
        in raw_labels,
        write_baseline=cadence in {NIGHTLY_CADENCE, WEEKLY_CADENCE},
    )
```

tests/ci/ci_utils.py

gate hook 的 write_baseline 语义落地处，也是 CodeQL 日志告警所在文件；参数从 nightly 泛化为 write_baseline 后 nightly 与 weekly 均可持续化基线

```
def run_gate_hook(filename, merged_record_path, *, store, registry, write_baseline, provenance,
now_iso=None):
    """根据策略决定是否把 gate 结果写入指标基线。

    语义变化：原先只有 nightly 写 baseline，现在由 `write_baseline` 显式控制 ——
    nightly 与 weekly 都写，普通 PR 只输出 shadow 判定。任何异常都被吞掉，
    gate 永远不能改变测试的 pass/fail。
    """
    try:
        result = evaluate_gate(filename, merged_record_path, store, registry=registry)

        if write_baseline:
            if not result.metrics:
                # 没有声明 gate 的文件不写空 run 行
                logger.info(f"[CI Gate][baseline] {filename}: no gate declared; skipping write")
                return
            identity = RunIdentity(
                test_path=result.test_path, backend=result.backend, suite=result.suite
            )
            created_at = now_iso or datetime.datetime.now(datetime.timezone.utc).isoformat()
            # 相同坐标的 spec 只写一行，避免重复权重污染基线均值
            seen_coords = set()
            values = []
            for m in result.metrics:
                if m.current is None:
                    continue
                coord = (m.metric_key, m.steps_key, m.constraint_key, m.step)
                if coord in seen_coords:
                    continue
                seen_coords.add(coord)
                values.append(MetricSample(m.metric_key, m.steps_key, m.constraint_key, m.step,
                m.current))
```

```
            store.write_run(identity, provenance, created_at, trusted=result.trusted, values=values)
            # 日志保持静态摘要，不输出动态 gate 细节（回应 CodeQL 告警）
            logger.info(f"[CI Gate][baseline] {filename}: wrote baseline")
        else:
            # PR 运行：只写 shadow 判定到 step summary，绝不落库
            line = _shadow_verdict_line(filename, result)
            logger.info(line)
            detail = "\n".join(f"  - {m.metric_key} (step={m.step}): {m.reason}" for m in result.
            metrics)
            write_github_step_summary(line + ("\n" + detail if detail else "") + "\n")
    except Exception as e:
        logger.warning(f"[CI Gate] hook failed for {filename}: {type(e).__name__}: {e}")
```

评论区精华

唯一一条 inline 评论来自 github-advanced-security[bot] 的 CodeQL 扫描，针对 tests/ci/ci_utils.py 中 `logger.info(f"[CI Gate][nightly] ... wrote baseline (trusted={result.trusted}, {len(values)} value(s))")` 报出 Clear-text logging of sensitive information。从上下文看实际记录的是布尔值与整数，并非真实密钥，属于疑似误报；但作者在第三个 commit（2c18721）中主动将动态 gate 结果细节从日志中移除，改为静态字符串 `[CI Gate][baseline] ...: wrote baseline`，客观上消除了该告警。人工 reviewer yueming-yuan 直接 APPROVED，无其他设计争议。

- ci_utils.py 日志是否泄露敏感信息 (security): 第三个 commit 已将动态详情移除，告警不再适用；baseline 持久化语义保持不变，仅收敛日志输出。

风险与影响

- 风险：
  1. 策略 API 破坏：RunPolicy.is_nightly 属性被移除，任何 tests/ci 之外的引用都会 ImportError/AttributeError；材料未显示其他引用点，建议合并前全仓搜索确认。
  2. GPU 资源叠加：weekly 虽将每个矩阵 max-parallel 限为 1，但独立 stage（如单例 2-GPU stage）仍可与矩阵 job 并发，PR body 也承认 "Independent stages, including the singleton 2-GPU stage, may still overlap"，极端情况下仍可能挤占 PR 队列。
  3. baseline 污染：weekly 与 nightly 都会写入指标 baseline，若全量跑期间存在 GPU 资源争抢导致性能 / 指标波动，可能写入 untrusted 行；现有 trusted 标记机制可兜底，但 baseline 刷新频率变化需要观察。
  4. cron 精确匹配：workflow 中两条 cron 字符串必须与 SCHEDULE_POLICIES 键严格一致，若 GitHub 端表达式被改写会直接 ValueError（有 test_every_configured_cron_has_an_explicit_python_policy 锁定，风险低）。 - 影响：影响范围集中在 CI 基础设施域，不涉及产品运行时（miles/ 训练相关代码零改动）。对团队的价值在于：每周六获得一次覆盖 long/ft-long 等 label-gated 与 nightly-only 测试的全量回归窗口，弥补日常 PR/nightly 的覆盖盲区；对自托管 GPU 集群的占用被限制为每矩阵单 runner，降低对工作日 PR 队列的挤占。对 CI 工具链维护者而言，

write_baseline 字段使 gate 写库语义与 cadence 解耦，后续新增节奏（如 biweekly）只需在 SCHEDULE_POLICIES 加一行并复用相同字段。文档（docs/ci/、contributor-guide）已同步，贡献者可预期周六存在全量运行。 - 风险标记：策略 API破坏（is_nightly 移除），weekly 独立 stage 仍可并发占用 GPU，双节奏写入 baseline存在指标波动风险，cron 与策略键强一致依赖

关联脉络

- PR #2403 fix(ci): unblock ROCm fork PRs at checkout: 同批 CI workflow（pr-test-rocm.yml / _run-ci-rocm.yml）的延续，本次继续调整 ROCm 侧 schedule 与 max-parallel，属于同一条 ROCm CI 维护线
- PR #2356 Replace all the .sh launch scripts with .py launch script: 建立了 tests/ci 下的 Python 化 CI 工具链体系，本次改动的 ci_policy.py、run_suite.py、ci_utils.py 正是该体系的核心模块，属同一演进方向
- PR #2389 fix: avoid Mooncake metrics port collisions: 同属 CI 基础设施维护域，涉及 command_utils 与 CI 测试，体现仓库近期对自托管 CI 稳定性的持续投入