

PR #53621 完整报告

vllm-project/vllm

[CI][Ray] Fix flaky multi-node assignment test after placement-group teardown

合并时间: 2026-08-30 07:48

原文链接: <http://prhub.com.cn/vllm-project/vllm/pull/53621>

执行摘要

- 一句话: 修复 Ray PG 拆除后资源等待不足导致的 CI flaky
- 推荐动作: 值得快速浏览 (10 分钟内即可读完) 的 CI/ 测试修复 PR。核心学习点: Ray placement group 拆除是异步的, GCS 状态 REMOVED 与 raylet 实际归还资源之间存在时间差, 等待逻辑必须基于真实资源; 测试应复刻生产 teardown 路径, 而不是自创一套等待机制; 删除死代码前应确认无其他调用方。对维护 Ray executor 与 CI 稳定性的工程师尤其有参考价值, 但不需要作为架构级 PR 精读。

功能与动机

修复 CI 失败: `tests/distributed/test_multi_node_assignment.py` 在 Buildkite L4 Distributed 2-Node 上不稳定。PR body 明确指出根因: "The test waited via `_wait_until_pg_removed`, which production never called (production only calls `remove_placement_group`). The helper polled `get_current_placement_group()`, which is always `None` on the driver, so it returned immediately." 此外 PR body 还解释了为何备选的 GCS 状态检查也不可行: "Ray can record the placement group as removed before the per-node raylets return the reserved resources", 因此必须等待真实资源归还而非 PG 状态表。

实现拆解

整个修复按以下步骤展开:

1. 删除死代码: `vllm/v1/executor/ray_utils.py` 整体删除 `_wait_until_pg_removed` (约18行)。该函数先调用 `remove_placement_group`, 再按指数退避轮询 `ray.util.get_current_placement_group()`, 但该 API 只反映调用者所在 task/actor 被调度到的 placement group, 在 driver 上恒为 `None`, 等待立即结束。生产路径只在启动时创建 PG, 从不等待拆除, 此函数属于死代码。
2. 测试对齐生产 teardown: `tests/distributed/test_multi_node_assignment.py` 移除对 `_wait_until_pg_removed` 的导入, 每一轮循环末尾改为显式调用 `ray.util.remove_placement_group(config.placement_group)`, 与生产 teardown 路径保持一致。
3. 等待真实资源归还: 新增 60 秒截止时间 (`time.monotonic() + 60`), 通过 `available_resources_per_node()` 轮询 driver 节点 (

`ray.get_runtime_context().get_node_id())` 的空闲 GPU 数, 达到 `gpus_needed` (即本轮的 worker 数) 才进入下一轮; 超时抛出 `TimeoutError("placement group resources were not released")`。这一设计恰好覆盖了 GCS 记录 `REMOVED` 状态先于 per-node raylet 归还资源的异步时间差。

4. 按 review 收敛实现: 初版提交 `Add exponential backoff` 复制了旧 helper 的指数退避逻辑, 经 taneem-ibrahim 评审建议后, 改为固定 1 秒短轮询并用 `min(1, remaining)` 封顶剩余时间, 同时保留 deadline 最终检查 (提交 `Add comment`)。
5. 验证: 作者在 PR body 中给出 `VLLM_MULTI_NODE=1 pytest tests/distributed/test_multi_node_assignment.py` 的测试结果, 1 passed in 74.30s。

关键文件:

- `vllm/v1/executor/ray_utils.py` (模块 执行器; 类别 source; 类型 dead-code-removal; 符号 `_wait_until_pg_removed`): 源码主文件: 整体删除死代码 `_wait_until_pg_removed` (约 18 行)。该函数轮询 `get_current_placement_group()` 在 driver 上恒为 `None`, 等待形同虚设, 且生产路径从不调用, 删除无回归风险。
- `tests/distributed/test_multi_node_assignment.py` (模块 多节点测试; 类别 test; 类型 test-coverage): 修复核心所在: 测试改为复刻生产 `teardown` (直接 `remove_placement_group`), 并轮询 `available_resources_per_node` 等待 driver 空闲 GPU 达标, 附带 60 秒超时与固定 1 秒轮询。

关键符号: `_wait_until_pg_removed`, `test_multi_node_assignment`

关键源码片段

`vllm/v1/executor/ray_utils.py`

源码主文件: 整体删除死代码 `_wait_until_pg_removed` (约 18 行)。该函数轮询 `get_current_placement_group()` 在 driver 上恒为 `None`, 等待形同虚设, 且生产路径从不调用, 删除无回归风险。

```
# vllm/v1/executor/ray_utils.py
# _wait_until_pg_removed 在本 PR 中被整体删除 (原实现约 18 行)。
# 删除理由:
# - 生产路径从不调用它, 属于死代码;
# - 它轮询 ray.util.get_current_placement_group() 判断 PG 是否移除,
# 该 API 在 driver 上恒为 None, 等待立即返回, 无法覆盖
# "GCS 已标记 REMOVED 但 raylet 尚未归还 GPU 资源" 的时间差。
```

```
def initialize_ray_cluster(
    parallel_config: ParallelConfig,
    ray_address: str | None = None,
    require_gpu_on_driver: bool = True,
):
    """Initialize the distributed cluster with Ray.
```

```
    it will connect to the Ray cluster and create a placement group
    for the workers, which includes the specification of the resources
```

for each distributed worker.

Args:

parallel_config: The configurations for parallel execution.

ray_address: The address of the Ray cluster. If None, uses the default Ray cluster address.

require_gpu_on_driver: If True (default), require at least one GPU on the current (driver) node and pin the first PG bundle to it. Set to False for executors like RayExecutorV2 where all GPU work is delegated to remote Ray actors.

"""

assert_ray_available()

from vllm.platforms import current_platform

关闭 Ray 使用统计上报

```
if os.environ.get("RAY_USAGE_STATS_ENABLED", "0") != "1":
    os.environ["RAY_USAGE_STATS_ENABLED"] = "0"
```

在 Ray 调度前预校验 GPU 数量，给出更友好的报错信息

```
if current_platform.is_cuda() and parallel_config.world_size > 1:
    available_gpus = current_platform.device_count()
    if parallel_config.world_size > available_gpus:
```

tests/distributed/test_multi_node_assignment.py

修复核心所在：测试改为复刻生产 teardown（直接 `remove_placement_group`），并轮询 `available_resources_per_node` 等待 driver 空闲 GPU 达标，附带 60 秒超时与固定 1 秒轮询。

tests/distributed/test_multi_node_assignment.py

该测试在 2 节点集群上验证：placement group 创建后，GPU worker

必须被调度到 driver 节点；拆除 PG 后要等资源真正归还，才能进入下一轮。

import os

import time

import pytest

import ray

from ray.util.scheduling_strategies import PlacementGroupSchedulingStrategy

from vllm import initialize_ray_cluster

from vllm.config import ParallelConfig

from vllm.utils.network_utils import get_ip

from vllm.v1.executor.ray_utils import available_resources_per_node

VLLM_MULTI_NODE = os.getenv("VLLM_MULTI_NODE", "0") == "1"

@pytest.mark.skipif(

not VLLM_MULTI_NODE, reason="Need at least 2 nodes to run the test."

)

```

def test_multi_node_assignment() -> None:
    # NOTE: important to keep this class definition here
    # to let ray use cloudpickle to serialize it.
    class Actor:
        def get_ip(self):
            return get_ip()

    for _ in range(10):
        config = ParallelConfig(1, 2)
        initialize_ray_cluster(config)

        current_ip = get_ip()
        workers = []
        for bundle_id, bundle in enumerate(config.placement_group.bundle_specs):
            if not bundle.get("GPU", 0):
                continue
            scheduling_strategy = PlacementGroupSchedulingStrategy(
                placement_group=config.placement_group,
                placement_group_capture_child_tasks=True,
                placement_group_bundle_index=bundle_id,
            )
            worker = ray.remote(
                num_cpus=0,
                num_gpus=1,
                scheduling_strategy=scheduling_strategy,
            )(Actor).remote()
            # 关键断言: GPU worker 必须落在 driver 节点
            assert ray.get(worker.get_ip.remote()) == current_ip
            workers.append(worker)

        for worker in workers:
            ray.kill(worker)

    # 与生产 teardown 保持一致: 直接调用 remove_placement_group,
    # 而不是依赖测试专用等待函数。
    ray.util.remove_placement_group(config.placement_group)

    # ray 的 GCS 可能在 per-node raylet 归还资源前就把 PG 标记为
    # REMOVED, 因此必须轮询真实空闲 GPU 数而不是查 PG 状态表。
    # 旧实现轮询 get_current_placement_group() 在 driver 上恒为 None,
    # 会立即返回, 导致下一轮 initialize_ray_cluster 时 GPU 尚未归还。
    gpus_needed = len(workers)
    deadline = time.monotonic() + 60
    node_id = ray.get_runtime_context().get_node_id()
    while True:
        free_gpus = available_resources_per_node().get(node_id, {}).get(
            "GPU", 0
        )
        if free_gpus >= gpus_needed:

```

```
break
remaining = deadline - time.monotonic()
if remaining <= 0:
    raise TimeoutError("placement group resources were not released")
# 固定短轮询：最多等 1 秒，且不超过剩余截止时间
time.sleep(min(1, remaining))
```

评论区精华

核心讨论集中在测试等待逻辑的实现方式。taneem-ibrahim 在 CHANGES_REQUESTED 中建议："Could we use a short fixed polling interval here, or cap the sleep to the remaining time and perform a final resource check at the deadline?" 作者 yzong-rh 采纳并回复："Fixed and retested. Originally just copied the logic from `_wait_until_pg_removed`, but this is cleaner." 作者坦承初版只是照搬旧 helper 的指数退避，按建议改为固定短轮询 + 截止检查。评审最终由 taneem-ibrahim 和 mgoin 双双 APPROVED。

- 等待资源释放的轮询策略 (testing): 作者采纳建议，将初版的指数退避改为固定 1 秒轮询 + `min(1, remaining)` 封顶 + deadline 最终检查，重新测试通过后评审转 APPROVED。
- 实现思路：照搬旧 helper 还是重写 (design): 作者确认照搬逻辑不够好，按 review 意见改为更简洁的固定短轮询方案，并说明 "this is cleaner"。

风险与影响

- 风险：
 1. 时序依赖：60 秒 deadline 是经验值，在高负载或共享 CI 集群上 Ray 资源归还可能更慢，测试仍有小概率超时失败。
 2. 轮询仅覆盖 driver 节点：等待逻辑只读取 `available_resources_per_node().get(node_id, {})`，不检查其他节点。当前用例中所有 worker 都被 `assert worker_ip == current_ip` 确认 pin 在 driver，因此局部检查成立；未来若用例让 bundle 分布到多节点，该等待逻辑需要扩展。
 3. 并发干扰：若同一节点上有其他 job 并发占用 GPU，driver 空闲 GPU 数可能迟迟不达标，造成误报超时；vLLM CI 通常独占节点，风险有限。
 4. 运行时回归风险极低：`_wait_until_pg_removed` 除测试外无其他调用方（源码分析已确认），纯删除不影响推理路径。
 - 影响：对用户和推理运行时零影响，仅删除未使用的源码函数并调整测试逻辑。对 CI 的影响是直接的：L4 Distributed 2-Node 测试从频繁 flaky 恢复稳定，减少失败重跑与人工排查成本。对团队而言，该 PR 提供了一个值得复用的测试模式——等待 placement group 拆除时应轮询真实资源（`available_resources_per_node`）而非 PG 状态表，且测试 teardown 应与生产路径保持一致。代码库净减约 18 行死代码。
 - 风险标记：资源归还时序依赖，轮询仅覆盖 driver 节点，超时阈值为经验值

关联脉络

- PR #54468 [CI] Restore gpu_1_queue routing for torch-abi audit: 同为 CI 稳定性修复方向（回滚队列路由以恢复 CI），与本 PR 一起体现近期对 CI 可靠性的持续投入。

- PR #54482 [CI/Build] Fix Kimi K3 Eagle3 test fixture: 同为测试 /CI 修复类 PR (补全测试 fixture 修复 CI 失败) , 与本 PR 属于同一维护主题。