PR #22898 完整报告

sgl-project/sglang

[Ray] Auto-create placement group in RayEngine when none is detected

合并时间：2026-04-16 06:17

原文链接：http://prhub.com.cn/sgl-project/sglang/pull/22898

执行摘要

- 一句话：RayEngine 在未检测到 placement group 时自动创建，简化部署流程。
- 推荐动作：该 PR 值得精读，特别是对于使用 Ray 部署 SGLang 的工程师。关注点包括：自动计算 GPU 需求的逻辑、placement group 策略选择（STRICT_PACK vs SPREAD）、以及 review 中提到的未解决问题（如 _ACTOR_RUNTIME_ENV 变量）是否在其他 PR 中处理。

功能与动机

根据 PR body 描述，之前 RayEngine 在没有 placement group 时会抛出 RuntimeError，强制调用者手动构造一个。此变更旨在自动处理此场景，简化部署流程，使 RayEngine 在未检测到 placement group 时能够自动创建，提升用户体验。

实现拆解

1. 检测与自动创建逻辑：在 python/sglang/srt/ray/engine.py 的 _launch_scheduler_processes 方法中，当 ray.util.get_current_placement_group() 返回 None 时，不再抛出 RuntimeError，而是导入 ray.util.placement_group.placement_group 并自动创建。
2. 计算 GPU 需求：根据 server_args.enable_dp_attention 标志决定总 GPU 数：若启用 DP attention，则 total_gpus = tp_size * pp_size；否则 total_gpus = dp_size * tp_size * pp_size。然后计算每个节点的 GPU 数：gpus_per_node = total_gpus // nnodes。
3. 选择放置策略：根据节点数 nnodes 决定策略：单节点时使用 STRICT_PACK（所有 bundle 位于同一节点），多节点时使用 SPREAD（bundle 分布到不同节点）。
4. 创建并等待 placement group：使用 create_placement_group 创建包含 nnodes 个 bundle 的 placement group，每个 bundle 资源为 {"CPU": 1, "GPU": gpus_per_node}，并调用 ray.get(pg.ready()) 等待就绪。
5. 日志与兼容性：添加日志输出告知用户自动创建行为，并保留原有逻辑：若已有 placement group 则直接使用，确保向后兼容。

关键文件：

- python/sglang/srt/ray/engine.py（模块 Ray 引擎；类别 source；类型 core-logic；符号 _launch_scheduler_processes）：这是核心变更文件，修改了 RayEngine 启动 scheduler 进程的逻辑，实现了自动创建 placement group 的功能。

关键符号：_launch_scheduler_processes

关键源码片段

python/sglang/srt/ray/engine.py

这是核心变更文件，修改了 RayEngine 启动 scheduler 进程的逻辑，实现了自动创建 placement group 的功能。

```
@classmethod
def _launch_scheduler_processes(
    cls,
    server_args: ServerArgs,
    port_args: PortArgs,
    run_scheduler_process_func: Callable,
) -> tuple[SchedulerInitResult, None]:
    """Launch schedulers as Ray actors.

    Returns:
        Tuple of (RaySchedulerInitResult, None).
        scheduler_procs is None since Ray uses actors instead of mp.Process.
    """
    pg = ray.util.get_current_placement_group()
    if pg is None:
        # 导入 placement group 创建函数
        from ray.util.placement_group import (
            placement_group as create_placement_group,
        )

        # 根据 DP attention 标志计算总 GPU 数量
        if server_args.enable_dp_attention:
            total_gpus = server_args.tp_size * server_args.pp_size
        else:
            total_gpus = (
                server_args.dp_size * server_args.tp_size * server_args.pp_size
            )

        nnodes = server_args.nnodes
        gpus_per_node = total_gpus // nnodes # 计算每个节点 GPU 数
        strategy = "STRICT_PACK" if nnodes == 1 else "SPREAD" # 单节点用 PACK，多节点用
        SPREAD

        logger.info(
            "No placement group detected. Auto-creating one with "
            f"{nnodes} bundle(s), {gpus_per_node} GPU(s)/bundle, "
            "To avoid this, create a placement group explicitly and schedule the Engine onto it."
        )

        # 创建 placement group，每个 bundle 包含 1 CPU 和 gpus_per_node GPU
        pg = create_placement_group(
```

```
            [{"CPU": 1, "GPU": gpus_per_node}] * nnodes,
            strategy=strategy,
        )
        ray.get(pg.ready()) # 等待 placement group 就绪

    # 后续逻辑保持不变，使用自动创建或现有的 placement group
    nnodes = server_args.nnodes
    # ...
```

评论区精华

review 评论中，gemini-code-assist[bot] 指出了三个问题：

1. 变量未定义：在 engine.py 中使用了未定义的 _ACTOR_RUNTIME_ENV 变量，可能导致运行时 NameError。
2. 日志消息不完整：自动创建 placement group 的日志消息 "placement group explicitly and schedule the Engine onto it." 不连贯，建议改为 "To avoid this, create a placement group explicitly and schedule the Engine onto it."。
3. 缺少 runtime_env：在 data_parallel_controller.py 中启动 SchedulerActor 时未传递 runtime_env，可能导致环境变量（如 RAY_EXPERIMENTAL_NOSET_CUDA_VISIBLE_DEVICES）未正确传播。 这些评论未在 PR 中直接解决，但 PR 已合并，可能在其他地方处理或作为已知问题。

- 变量 _ACTOR_RUNTIME_ENV 未定义 (correctness): 未在 PR 中直接解决，可能在其他地方处理或作为已知问题。
- 日志消息不完整 (documentation): 未在 PR 中直接解决，但建议已被记录。
- 缺少 runtime_env 传递 (correctness): 未在 PR 中直接解决，可能在其他地方处理。

风险与影响

- 风险：1. 运行时错误风险：如果 _ACTOR_RUNTIME_ENV 变量确实未定义，在启动 scheduler actors 时可能抛出 NameError，导致引擎启动失败。 2. 资源计算错误：自动计算 GPU 数量时依赖 server_args 的 dp_size、tp_size、pp_size 和 enable_dp_attention，若这些参数配置不当（如除零错误或资源不足），可能导致 placement group 创建失败或资源分配不合理。 3. 策略选择风险：单节点使用 STRICT_PACK 和多节点使用 SPREAD 是合理的默认策略，但在特定部署场景下可能不是最优选择，缺乏配置选项。 4. 兼容性风险：自动创建 placement group 可能干扰现有手动管理 placement group 的部署流程，但 PR 保留了检测到现有 group 时直接使用的逻辑，降低了此风险。
- 影响：1. 对用户的影响：简化了 RayEngine 的部署流程，用户无需手动创建 placement group，降低了使用门槛，提升了开发体验。 2. 对系统的影响：自动创建 placement group 增加了启动时的资源管理逻辑，可能轻微增加启动时间，但避免了因缺少 placement group 导致的启动失败。 3. 对团队的影响：此变更与 #21887（ray-dp-support）相关，是 Ray 支持数据并行功能的一部分，表明团队在增强 Ray 集成和分布式支持方面的持续投入。
- 风险标记：变量未定义风险，资源计算依赖配置，策略选择固化

关联脉络

- PR #21887 ray-dp-support: PR body 提到此 PR 基于 #21887，是 Ray 数据并行支持功能的一部分，涉及相关变更。