PR #6556 完整报告

verl-project/verl

[fully_async] feat: support dynamic resource scheduling

合并时间：2026-07-20 14:40

原文链接：http://prhub.com.cn/verl-project/verl/pull/6556

执行摘要

- 一句话：引入动态资源调度提升异步训练 GPU 利用率
- 推荐动作：建议：
  - 优先解决 ray.get() 阻塞问题：这是影响稳定性的关键 bug，应在合并前改用 await。
  - 增加单元测试：至少对 DynamicScheduleContext 计算、DefaultDynamicSchedulePolicy 决策逻辑、DynamicResourceController 状态转换进行模拟测试。
  - 关注 abort_replicas 位置：接受 wuxibin89 建议，将相关操作完全挪至 controller。
  - 值得精读：动态调度策略的设计模式（可插拔、上下文驱动）具有参考价值，可作为未来类似功能的样板。

功能与动机

PR 中指出当前 suboptimal GPU utilization: 'Trainer-node GPUs sit idle waiting for rollout data while Standalone Rollout nodes idle during training.' 动态调度通过让 Trainer 节点 GPU 在空闲时参与 rollout，提高整体利用率。

实现拆解

实现分为以下步骤：

1. 策略基类与注册机制（base.py）：定义 DynamicSchedulePolicyBase 抽象基类及 DynamicScheduleContext 上下文数据类，提供 register_policy 装饰器和 build_policy 工厂函数，使策略可插拔。
2. 内置策略实现：DefaultDynamicSchedulePolicy（自适应 deactivate_ratio，根据等待时间调整）、FixedRatioDynamicSchedulePolicy（固定比例）、StaticFullyAsyncPolicy（立即去激活且不复活），均位于 dynamic_schedule/ 下。
3. 生命周期控制器（dynamic_resource_controller.py）：DynamicResourceController 管理混合副本激活与去激活的顺序（先切路由、再中止请求、然后休眠 / 恢复），并通过 policy 决定何时切换。
4. Rollouter 改造（fully_async_rollouter.py）：FullyAsyncLLMServerManager 支持混合副本注册、激活 / 去激活接口、请求重平衡（rebalance_requests）、以及各副本 GPU 内存利用率独立控制（gpu_memory_utilization vs standalone_gpu_memory_utilization）。

5. Trainer 集成（fully_async_trainer.py）：FullyAsyncTrainer 新增 _setup_dynamic_resource_controller、修改 _fit_update_weights 以按策略决定是否同步 hybrid 权重，并记录资源利用率指标。
6. 配置与文档：添加 use_dynamic_resource_scheduling, dynamic_schedule_policy, dynamic_scaling_enable_rebalance 等配置项，并新增 docs/advance/dynamic_schedule.md 和模块内 README 说明策略开发方法。
7. 测试验证：使用 Qwen3.5-35B-A3B 在 DAPO-Math-17K 上验证，结果显示动态调度比 16+16 基线节省约 15.3% 时间（5.4h vs 6.546h），且未包含直接单元测试（但附有实验报告）。

关键文件：

- verl/experimental/fully_async_policy/dynamic_schedule/default_policy.py（模块 调度策略；类别 source；类型 core-logic；符号 DefaultDynamicSchedulePolicy, init, should_deactivate, deactivate_wait_samples）：核心自适应调度策略实现，包含 deactivate_ratio 动态调整和激活成本 / 收益检查，是动态调度逻辑的核心。
- verl/experimental/fully_async_policy/dynamic_schedule/base.py（模块 调度基类；类别 source；类型 dependency-wiring；符号 DynamicSchedulePolicyBase, register_policy, build_policy, DynamicScheduleContext）：定义动态调度策略的抽象基类、注册机制和统一上下文，是策略可插拔架构的基础。
- verl/experimental/fully_async_policy/dynamic_schedule/dynamic_resource_controller.py（模块 资源控制器；类别 source；类型 entrypoint；符号 DynamicResourceController, init, is_hybrid_active, has_hybrid_replicas）：管理混合副本激活 / 去激活生命周期，实现 STANDALONE_ONLY 与 HYBRID_ACTIVE 状态转换，是动态调度的执行器。
- verl/experimental/fully_async_policy/fully_async_rollouter.py（模块 Rollout 层；类别 source；类型 dependency-wiring；符号 FullyAsyncLLMServerManager._initialize_llm_servers, get_standalone_replicas, get_num_standalone_replicas, rebalance_requests）：扩展 LLMServerManager 支持混合副本，包括独立 GPU 内存控制、请求重平衡和混合副本注册接口。
- verl/experimental/fully_async_policy/fully_async_trainer.py（模块 训练器；类别 source；类型 core-logic；符号 _setup_dynamic_resource_controller, _fit_update_weights, _record_train_resource_utilization, _fit_log_aggregated_training_metrics）：整合动态资源控制器，修改权重同步流程，并记录资源利用率指标，是训练器侧核心改动。
- verl/experimental/fully_async_policy/dynamic_schedule/fixed_ratio_policy.py（模块 调度策略；类别 source；类型 core-logic；符号 FixedRatioDynamicSchedulePolicy, init, should_deactivate, deactivate_wait_samples）：提供固定比例的去激活策略，适用于稳定负载或调试场景。
- verl/workers/rollout/llm_server.py（模块 LLM 客户端；类别 source；类型 core-logic；符号 LLMServerClient.init, clear_sticky_cache, get_total_inflight, _acquire_server）：LLM 客户端新增 only_hybrid 模式和动态方法，支持混合副本的请求控制，是下游依赖的关键改动。

关键符号：DefaultDynamicSchedulePolicy.should_deactivate, DefaultDynamicSchedulePolicy.deactivate_wait_samples, DefaultDynamicSchedulePolicy.update_after_step, DefaultDynamicSchedulePolicy.should_activate_after_step, DynamicResourceController.activate_hybrid_replicas, DynamicResourceController.deactivate_hybrid_replicas, FullyAsyncTrainer._setup_dynamic_resource_controller, FullyAsyncTrainer._fit_update_weights, FullyAsyncRollouter.rebalance_requests, FullyAsyncLLMServerManager._initialize_llm_servers

关键源码片段

verl/experimental/fully_async_policy/dynamic_schedule/default_policy.py

核心自适应调度策略实现，包含 deactivate_ratio 动态调整和激活成本 / 收益检查，是动态调度逻辑的核心。

```
@register_policy("default")
class DefaultDynamicSchedulePolicy(DynamicSchedulePolicyBase):
    """默认自适应调度策略：实现去激活比率自适应调整与成本/收益激活检查。"""

    def __init__(self, deactivate_ratio: float = 0.3, only_hybrid: bool = False):
        self.deactivate_ratio = deactivate_ratio
        self.only_hybrid = only_hybrid
        # 开关开销滑动窗口（秒），用于激活成本 / 收益评估
        self._recent_switch_overheads: list[float] = []
        # 每样本生成时间估计，初始设为较大值，在有真实数据后更新
        self._last_per_sample_time: float = INITIAL_PER_SAMPLE_TIME_S
        if only_hybrid:
            self.deactivate_ratio = 1.0

    def should_deactivate(self, global_steps: int, is_hybrid_active: bool, ctx:
    DynamicScheduleContext) -> bool:
        # 只要混合副本激活就标记为可去激活，具体时机由 deactivate_wait_samples 控制
        return is_hybrid_active

    def deactivate_wait_samples(self, ctx: DynamicScheduleContext) -> int:
        # 等待 deactivate_ratio × step_required_samples 个样本后再去激活
        return int(ctx.step_required_samples * self.deactivate_ratio)

    def update_after_step(self, global_steps: int, ctx: DynamicScheduleContext) -> None:
        if global_steps <= 0 or self.only_hybrid:
            return
        # 更新开关开销
        overhead = ctx.last_activate_duration_s + ctx.last_deactivate_duration_s
        if overhead > 0:
            self._recent_switch_overheads.append(overhead)
        # 根据等待样本量调整 deactivate_ratio
        total_wait_samples = sum(ctx.step_wait_samples)
```

```
        if total_wait_samples > 0:
            # 有实际等待，说明 rollout 是瓶颈，应提高阈值（延迟去激活）
            step = min(max(total_wait_samples / ctx.step_required_samples,
                           DEACTIVATE_RATIO_INCREASE_MIN),
                       DEACTIVATE_RATIO_INCREASE_MAX)
            self.deactivate_ratio += step
        else:
            # 无等待，训练才是瓶颈，应降低阈值（提早去激活）
            self.deactivate_ratio -= DEACTIVATE_RATIO_DECREASE_STEP
        self.deactivate_ratio = max(0.0, min(1.0, self.deactivate_ratio))

    def should_activate_after_step(self, global_steps: int, is_hybrid_active: bool, ctx:
    DynamicScheduleContext) -> bool:
        if self.only_hybrid or not is_hybrid_active:
            return not is_hybrid_active # 若未激活且仅有 hybrid，则持续激活
        # 成本 / 收益检查：若独立 rollout 的生成短缺时间预计超过开关成本，则激活
        if ctx.total_generated_samples >= ctx.expected_samples - ctx.buffer_samples:
            return False # 样本充足，无需激活
        # 估算短缺时间
        shortfall = ctx.expected_samples - ctx.buffer_samples - ctx.total_generated_samples
        estimated_time = shortfall * self._last_per_sample_time
        return estimated_time > self._switch_cost()

    def _switch_cost(self) -> float:
        """滚动平均开关开销（秒）。"""
        if self._recent_switch_overheads:
            recent = self._recent_switch_overheads[-SWITCH_OVERHEAD_WINDOW_SIZE:]
            return sum(recent) / len(recent)
        return DEFAULT_SWITCH_THRESHOLD_S
```

评论区精华

Review 中主要讨论：

- get_num_standalone_replicas 返回类型错误（由 gemini-code-assist[bot] 指出）：函数声明为返回 int，实际返回 list，导致调用方错误。建议改为计算数量。
- 阻塞的 ray.get() 调用：在多个 async 方法（如 fit_step, activate_hybrid_replicas, deactivate_hybrid_replicas, _setup_dynamic_resource_controller, _fit_update_weights）中存在同步 ray.get()，会阻塞事件循环，可能导致死锁或性能退化。建议替换为 await 或 asyncio.gather。
  - abort_replicas 和 resume_generation_replicas 的位置争议（wuxibin89）：认为不应在 checkpoint_engine/base.py 中添加这些操作，而应移至 fully_async controller 中管理。
- LLMServerClient 子类化建议：wuxibin89 建议为 fully_async 需求创建一个 LLMServerClient 子类，而不是直接在现有类上增加 only_hybrid 参数。
  - get_num_standalone_replicas 返回类型错误 (correctness): 建议修改为返回计数值 len(self.rollout_replicas)。

- async 方法中阻塞 ray.get() (performance): 建议替换为 await 或 asyncio.gather。
- abort_replicas 与 resume_generation_replicas 位置争议 (design): 待确认，但 reviewer 明确要求移动。
- LLMServerClient 子类化建议 (design): 建议已提出，是否采纳未知。

风险与影响

- 风险：主要风险包括：
  1. 异步阻塞风险：多个 async 方法中仍保留 ray.get()，可能导致事件循环阻塞，严重时引起死锁或训练停滞。
  2. 状态机正确性：混合副本激活 / 去激活涉及路由、请求中止、权重同步等多个步骤，步骤间时序依赖紧密，一旦出错可能导致请求丢失或幽灵副本。
  3. 缺乏单元测试：PR 未包含直接测试动态调度逻辑的单元测试或集成测试，仅依赖手动实验，回归风险较高。
  4. 配置复杂性：新增多项配置参数（gpu_memory_utilization 两个版本、deactivate_ratio 等），不当设置可能导致 OOM 或资源浪费。
  5. 兼容性：改动涉及 checkpoint_engine/base.py、llm_server.py 等通用模块，可能影响其他使用场景（如非动态调度训练）。
- 影响：影响范围：
  - 用户：如需使用动态调度，必须设置 use_dynamic_resource_scheduling=true 并调整 gpu_memory_utilization 等参数。开启后预期 wall-clock 时间减少 10-20%。
  - 系统：新增 dynamic_schedule/ 子模块，修改 FullyAsyncLLMServerManager、FullyAsyncTrainer、MetricsAggregator 等核心类，影响完全异步模式下的所有训练流程。
  - 团队：需要维护多个内置策略，并保持注册机制稳定。扩展新策略需遵循模板方法。影响程度：中到高。动态调度是可选功能，默认关闭，但一旦开启则深度影响资源管理逻辑。
- 风险标记：ray.get() 阻塞 async 事件循环，缺少测试覆盖，混合副本状态机复杂度，新配置参数可能错误配置

关联脉络

- PR #6974 [ckpt,rollout] feat: sharded delta weight sync over NCCL for disaggregated rollout: 该 PR 实现了分片 delta 权重同步，为动态调度中的高效权重同步提供了基础，可能被动态调度依赖。
- PR #7049 [trainer, perf] fix: include standalone rollout GPUs in throughput denominator for separate async: 该 PR 修正了分离模式下的吞吐量 GPU 计数，动态调度中的资源利用率指标可能复用其逻辑。