PR #6076 完整报告

verl-project/verl

[fully_async] feat: reuse trainer worker group for hybrid rollout to do validation

合并时间：2026-05-12 10:09

原文链接：http://prhub.com.cn/verl-project/verl/pull/6076

执行摘要

- 一句话：全异步训练中复用 trainer GPU 进行混合 rollout 验证
- 推荐动作：值得精读。本 PR 展示了如何在 Ray 异步环境中实现弹性副本和资源切换，负载均衡器设计、两阶段初始化、KV cache 分层管理等模式值得参考。但 ray.get 阻塞问题需尽快修复，当前版本遗留部分 TODO（如 FullyAsyncLLMServerManager 重构、trtllm 兼容等），适合跟进后续 PR 完善。

功能与动机

全异步训练模式中 use_trainer_do_validate 功能因无法动态管理副本而损坏。此 PR 通过运行时添加 / 移除副本修复该能力，并为未来弹性调度和容错建设提供基础设施。详见 PR body：'By dynamically adding or removing replicas at runtime, this PR fixes the use_trainer_do_validate capability that was broken in fully-async training mode. Furthermore, it provides the necessary infrastructure components for future elastic-scheduling / resilience building.'

实现拆解

1. 合并 Handle Registry 到 GlobalRequestLoadBalancer（verl/workers/rollout/llm_server.py）：之前每个 LLMServerClient 本地缓存 servers 字典，弹性扩缩容需广播到所有客户端。现在 LB 自身拥有 _servers 和 _inflight_requests，acquire_server 一次性返回 (server_id, handle)，实现原子操作。并新增 add_servers 和 remove_servers 方法支持批量操作。
2. 新增弹性副本管理类（verl/experimental/fully_async_policy/fully_async_rollouter.py）：FullyAsyncLLMServerClient 继承 LLMServerClient，重写 generate 方法支持部分 rollout 恢复，使中断对 AgentLoop 透明。FullyAsyncLLMServerManager 支持两阶段初始化：先初始化弹性混合副本（由 trainer worker group 注入，初始化后立即休眠），再初始化固定独立副本。
3. Trainer 侧验证流程（verl/experimental/fully_async_policy/fully_async_trainer.py）：引入三阶段验证循环：(1) TRAIN→ROLLOUT 阶段：同步权重、中止所有副本、激活弹性副本到 LB、恢复生成；(2) Validate 阶段：通过 RPC 执行验证；(3) ROLLOUT→TRAIN 阶段：中止副本、停用弹性副本、休眠弹性 GPU、恢复固定副本。使用 hybrid_checkpoint_manager（naive 后端）管理弹性副本池，与固定副本的 checkpoint_manager 分离。

4. CheckpointEngine 细化管理与 KV cache 优化（verl/checkpoint_engine/base.py）：新增 abort_replicas、resume_generation_replicas、release_kv_cache_replicas、resume_kv_cache_replicas 方法，替换原有的整体 sleep_replicas/wake_up_replicas 序列。其中 release_kv_cache 利用 vLLM 的 sleep(level=1) 仅释放 KV cache 而保留权重，减少显存峰值。注意：因与视觉语言模型冲突，目前 release_kv_cache/resume_kv_cache 仅保留接口，实作为空，待后续支持。
5. 入口层调整与适配（verl/experimental/fully_async_policy/fully_async_main.py）：调整组件初始化顺序，确保 trainer 先于 rollouter 创建以注入弹性 worker group。其他适配包括统一命名（elastic→hybrid/standalone）、修复变量 shadowing 和双计数错误、为 vLLM 和 SGLang 服务器添加 _is_aborted 状态标记等。

关键文件：

- verl/experimental/fully_async_policy/fully_async_rollouter.py（模块 rollouter；类别 source；类型 dependency-wiring；符号 FullyAsyncLLMServerClient, generate, FullyAsyncLLMServerManager, init）：实现弹性 rollout 核心：FullyAsyncLLMServerClient 和 FullyAsyncLLMServerManager，支持部分恢复和两阶段初始化。
- verl/checkpoint_engine/base.py（模块 检查点引擎；类别 source；类型 core-logic；符号 abort_replicas, resume_generation_replicas, release_kv_cache_replicas, resume_kv_cache_replicas）：核心 checkpoint 管理类，新增 abort、resume、release/resume KV cache 方法，支撑弹性副本生命周期。
- verl/workers/rollout/llm_server.py（模块 负载均衡器；类别 source；类型 dependency-wiring；符号 acquire_server, get_inflight_count, get_all_servers, get_status）：全局负载均衡器重构，合并 Handle Registry，支持原子获取和批量增删，是弹性副本路由基础。
- verl/experimental/fully_async_policy/fully_async_trainer.py（模块 trainer；类别 source；类型 core-logic；符号 _setup_checkpoint_manager, _setup_hybrid_checkpoint_manager, set_rollouter, _init_reward_loop）：实现 trainer 侧三阶段验证循环，管理弹性副本生命周期，使用 hybrid_checkpoint_manager 同步参数。
- verl/workers/rollout/vllm_rollout/vllm_async_server.py（模块 vLLM 服务器；类别 source；类型 core-logic；符号 clear_kv_cache, release_kv_cache, resume_kv_cache）：vLLM 服务器适配：添加 release_kv_cache 和 resume_kv_cache 接口，利用 sleep/wake_up 分层控制。
- tests/experimental/agent_loop/test_basic_agent_loop.py（模块 测试；类别 test；类型 test-coverage；符号 test_release_invalid_server_raises, test_release_invalid_server_silently_ignored, test_release_without_inflight_raises, test_release_without_inflight_silently_ignored）：核心测试文件，覆盖 LB 新行为：原子获取、批量增删、粘滞会话失效等。

关键符号：FullyAsyncLLMServerClient.generate, FullyAsyncLLMServerManager.init, FullyAsyncLLMServerManager._initialize_llm_servers, FullyAsyncLLMServerManager.add_replicas, FullyAsyncLLMServerManager.remove_replicas, GlobalRequestLoadBalancer.acquire_server,

GlobalRequestLoadBalancer.add_servers, GlobalRequestLoadBalancer.remove_servers, CheckpointEngineManager.abort_replicas, CheckpointEngineManager.resume_generation_replicas, CheckpointEngineManager.release_kv_cache_replicas, CheckpointEngineManager.resume_kv_cache_replicas, FullyAsyncTrainer._trainer_side_validate, FullyAsyncTrainer._setup_hybrid_checkpoint_manager, FullyAsyncRollouter.get_num_hybrid_replicas

关键源码片段

verl/experimental/fully_async_policy/fully_async_rollouter.py

实现弹性 rollout 核心：FullyAsyncLLMServerClient 和 FullyAsyncLLMServerManager，支持部分恢复和两阶段初始化。

```
# verl/experimental/fully_async_policy/fully_async_rollouter.py

class FullyAsyncLLMServerClient(LLMServerClient):
    """支持在部分 rollout 中断后恢复生成，对 AgentLoop 透明。"""

    @rollout_trace_op
    async def generate(
        self,
        request_id,
        *,
        prompt_ids: list[int],
        sampling_params: dict[str, Any],
        image_data=None,
        video_data=None,
    ) -> TokenOutput:
        # 归一化 token ids
        prompt_ids = normalize_token_ids(prompt_ids)

        # 确定 max_tokens 限制键
        limit_key = None
        if "max_tokens" in sampling_params:
            limit_key = "max_tokens"
        elif "max_new_tokens" in sampling_params:
            limit_key = "max_new_tokens"
        original_max_tokens = sampling_params.get(limit_key) if limit_key else None

        final_output = TokenOutput(
            token_ids=[], log_probs=[], num_preempted=0,
        )

        while True:
            # 1. 调用父类 generate（可能被部分中止）
            output = await super().generate(
                request_id=request_id,
                prompt_ids=prompt_ids + final_output.token_ids, # 拼上已生成的部分
```

```
            sampling_params=sampling_params,
            image_data=image_data,
            video_data=video_data,
        )

        # 2. 合并本次生成到 final_output
        final_output.token_ids.extend(output.token_ids)
        if output.log_probs is not None:
            final_output.log_probs.extend(output.log_probs)
        if output.routed_experts is not None and len(output.token_ids) > 0:
            if final_output.routed_experts is None:
                final_output.routed_experts = output.routed_experts
            else:
                final_output.routed_experts = torch.cat(
                    [final_output.routed_experts,
                     output.routed_experts[-len(output.token_ids):]],
                    dim=0,
                )
        if output.num_preempted is not None:
            final_output.num_preempted += output.num_preempted
        final_output.stop_reason = output.stop_reason

        # 3. 更新权重版本信息
        global_steps = output.extra_fields.get("global_steps")
        # ... 最小 / 最大 global_steps 记录 ...

        # 4. 如果正常结束（非中止）则跳出
        if output.stop_reason != "aborted":
            break

        # 若被中止，则调整 sampling_params 的 max_tokens 为剩余量，继续循环
        if limit_key and original_max_tokens:
            generated_so_far = len(final_output.token_ids)
            sampling_params[limit_key] = original_max_tokens - generated_so_far

    return final_output
```

verl/workers/rollout/llm_server.py

全局负载均衡器重构，合并HandleRegistry，支持原子获取和批量增删，是弹性副本路由基础。

```
# verl/workers/rollout/llm_server.py

@ray.remote
class GlobalRequestLoadBalancer:
    def __init__(self, servers: dict[str, ray.actor.ActorHandle],
                 max_cache_size=10000):
        self._servers = dict(servers) # 统一管理 handle
        self._inflight_requests = {sid: 0 for sid in servers}
        self._request_id_to_server = LRUCache(maxsize=max_cache_size)
```

```
    def acquire_server(self, request_id: str) -> tuple[str, ray.actor.ActorHandle]:
        """原子获取（server_id, handle），支持粘滞会话。"""
        if request_id in self._request_id_to_server:
            server_id = self._request_id_to_server[request_id]
            if server_id in self._inflight_requests: # 服务器仍在池中
                self._inflight_requests[server_id] += 1
                return server_id, self._servers[server_id]
            del self._request_id_to_server[request_id]

        if not self._inflight_requests:
            raise RuntimeError("No available servers in load balancer")
        server_id = min(self._inflight_requests, key=self._inflight_requests.get)
        self._request_id_to_server[request_id] = server_id
        self._inflight_requests[server_id] += 1
        return server_id, self._servers[server_id]

    def add_servers(self, servers: dict[str, ray.actor.ActorHandle]):
        """批量添加服务器，原子更新。"""
        for sid, handle in servers.items():
            self._inflight_requests[sid] = 0
            self._servers[sid] = handle
        logger.info(f"[GlobalLoadBalancer] added {len(servers)} servers")

    def remove_servers(self, server_ids: list[str]):
        """批量移除服务器，原子更新。"""
        for sid in server_ids:
            self._inflight_requests.pop(sid, None)
            self._servers.pop(sid, None)
        logger.info(f"[GlobalLoadBalancer] removed {len(server_ids)} servers")
```

评论区精华

- ray.get 在 async 方法中阻塞事件循环：gemini 多次指出 ray.get() 在 async 方法中阻塞事件循环，建议使用 await 或 asyncio.gather 并行化。wuxibin89 也要求处理。ArronHZG 回应将在下一个 PR 中修复所有 ray.get 逻辑，当前版本未修改。
- 变量 shadowing 和双计数：gemini 发现 num_elastic 变量被第二个 _initialize_elastic_replicas 的返回值覆盖，导致日志中弹性副本数报告错误；get_active_server_count 和 get_server_info 对弹性副本重复计数。ArronHZG 在后续提交中已修复。
- 初始化顺序导致崩溃：gemini 发现 trainer.set_rollouter 在 rollouter.init_workers 之前调用会导致崩溃，因为 _setup_hybrid_checkpoint_manager 访问未初始化的 async_rollout_manager。ArronHZG 已修复初始化顺序。
- FullyAsyncLLMServerManager 架构设计：wuxibin89 建议不继承 LLMServerManager 而管理两个子管理器，并移至 experimental 目录，同时应调用 super()._initialize_llm_servers() 以保持 trtllm 兼容。ArronHZG 认为改动大，留待后续。

- ray.get 在 async 方法中阻塞事件循环 (performance): ArronHZG 回应将在下一个 PR 中修复所有 ray.get 逻辑，当前版本未修改。
- 变量 shadowing 导致日志错误 (correctness): ArronHZG 后续提交中已通过引入 num_fixed 变量修复。
- 初始化顺序导致崩溃 (correctness): ArronHZG 在提交中已修复初始化顺序。
- 弹性副本双计数 (correctness): ArronHZG 后续提交中修复，通过维护 alive_replicas 映射来区分。
- FullyAsyncLLMServerManager 架构设计争议 (design): ArronHZG 认为改动大，留待后续 PR 处理。

风险与影响

- 风险：性能风险：大量 ray.get 在 async 方法中阻塞事件循环，可能造成严重性能下降和潜在死锁。尤其在验证阶段的循环中，RPC 未并行化。正确性风险：变量 shadowing 和双计数错误已修复，但类似问题可能仍存在于其他新加逻辑。兼容性风险：FullyAsyncLLMServerManager 未调用 super()._initialize_llm_servers()，可能破坏 trtllm 完全异步支持（PR #5631）。release_kv_cache/resume_kv_cache 对 vLLM 和 SGLang 引擎版本依赖性强，当前为空接口，功能未实际启用。回归风险：GlobalRequestLoadBalancer 接口变更（返回类型从 str 改为 tuple[str, ActorHandle]），所有调用点需同步更新。
- 影响：用户影响：use_trainer_do_validate=True 现可在全异步模式下正常工作；弹性副本为未来弹性调度和容错奠定基础。用户需注意配置 hybrid_engine 等参数。系统影响：加重了 Ray actor 间交互，增加了 LB 职责，但减少了广播操作。KV cache 优化能降低权重同步时的显存峰值。_is_aborted 机制增强了系统鲁棒性。团队影响：本次重构涉及多个模块，对后续开发维护要求提高。命名变更需同步文档和示例。
- 风险标记：ray.get 阻塞事件循环，双计数逻辑错误，trtllm 兼容性风险，KV cache 优化接口空实现，变量 shadowing

关联脉络

- PR #6129 refactor: (hold) 本 PR 依赖的重构：wuxibin89 在 Issue 评论中明确要求 'Hold until refactor: https://github.com/verl-project/verl/pull/6129'，说明本 PR 在该重构 PR 的基础上合并。
- PR #5631 trtllm fully async support: wuxibin89 指出本 PR 可能破坏 trtllm 完全异步支持，因为 FullyAsyncLLMServerManager 未调用 super()._initialize_llm_servers()。
- PR #6056 OPD fully async: wuxibin89 在评论中质问为何移除 teacher model resource pool，此改动可能影响 OPD 蒸馏功能。
- PR #6228 multi-output reward scoring: 本 PR 在 _init_reward_loop 中有修改，与多输出奖励评分功能有交互。