

PR #7873 完整报告

verl-project/verl

[perf] fix: remove full gc during weight refit

合并时间: 2026-09-15 22:25

原文链接: <http://prhub.com.cn/verl-project/verl/pull/7873>

执行摘要

- 一句话: 移除权重同步 full GC 并修复 IPC ACK 竞态
- 推荐动作: 值得精读, 尤其是 `verl/workers/rollout/vllm_rollout/bucketed_weight_transfer.py`: 它演示了如何用内存快照把“性能优化”还原成“跨进程竞态修复”, 以及 CUDA IPC 的 `CudaIPCSentDataLimbo_ + ipc_collect()` 组合在什么条件下无法回收 block。建议关注三个设计决策: 最后一帧 ACK 推迟到 `_cleanup` 之后、`_ack_pending` 守卫如何维持 REP socket 的收发热度、`del weights, tensor` 前置如何顺带覆盖 `_direct_send_large_weight`。需要提醒读者的未闭环项: 接收方 `_cleanup` 异常时的发送方永久阻塞 (应尽快与 #7366 的 ACK 超时对齐或补 `try/except` 兜底 ACK), 以及 `tests/utils/test_bucketed_weight_transfer.py` 缺失的 ACK 顺序断言——后者是未来重构最容易踩的坑。若团队正在维护 #7780 的 GC 开关, 本 PR 之后应考虑废弃或标注为过渡方案。

功能与动机

PR body 明确说明这是 #7864 / #7848 的后续: 前序 PR 只移除了 rollout 权重 resume 之前的 full GC, `colocated weight-refit` 路径上仍残留 GC。作者用

`torch.cuda.memory._dump_snapshot` 分析 vLLM bucketed 权重传输后发现,

`BucketedWeightSender._cleanup` / `BucketedWeightReceiver._cleanup` 中的 `gc.collect()` 在 4104 次调用中回收 0 CUDA 字节, 唯一效果是 433 ms 停顿, 而正是这段停顿让接收方来得及释放 mapping, 使随后的 `ipc_collect()` 侥幸生效——原文表述是 “the GC was functioning as an accidental sleep, papering over a cross-process race”。因此动机不是单纯省时间, 而是先消除 GC 的真实依赖: 不做任何修复就删 GC 会把整个 2 GiB bucket buffer (默认 `rollout.checkpoint_engine.update_weights_bucket_megabytes: 2048`) 滞留整整一轮, 恰好卡在 trainer 应该把显存交还给 KV-cache 唤醒的时刻。

实现拆解

1. 根因定位 (内存快照驱动的分析): 在 1 节点、Qwen3-0.6B、`model_engine=veomni (FSDP2)` 且 `param_offload=True/optimizer_offload=True`、vLLM TP=2、`gpu_memory_utilization=0.6`、`update_weights_bucket_megabytes=2048` 的 GRPO on GSM8K 配置下, 对 rank 0 (IPC 发送方) 连续 5 轮做 `torch.cuda.memory._dump_snapshot`。对比 A (保留 GC: reserved 0.039 GiB、GC 停顿 433.3 ms)、B (只删 GC: reserved 4.000 GiB、每轮滞留 2048 MiB)、C (删 GC + 重排 ACK: reserved 0.039 GiB、停顿 7.1 ms) 三组数据, 确认 GC 只是掩盖了竞态。

2. 修复协议时序 ([verl/workers/rollout/vllm_rollout/bucketed_weight_transfer.py](#), 本 PR 的实质改动) : BucketedWeightReceiver.receive_weights 不再在循环内为最后一桶回 ACK, 而是置 `_ack_pending = True` 后 `break`, ACK 改由 `_cleanup` 在 `del self.buffer`、`shm.close()`、`ipc_collect()`、`empty_cache()` 全部完成之后补发 (`_ack_pending` 守卫保证 ZMQ REP socket 的 `recv/send` 严格交替) ; `_cleanup` 因此不再先 `socket.close()` ; `del weights, tensor` 从 ACK 之后提前到 ACK 之前, 从而同样覆盖 `_direct_send_large_weight` 通过 IPC 直传的大权重; 发送侧在发最后一桶前补 `name = weight = None`, 避免 `async for` 循环变量在循环退出后仍把最后一个 (可能 IPC 共享的) 权重保留到 `_cleanup` 之后。两处 `_cleanup` 的 `gc.collect()` 与模块级 `import gc` 一并删除。
3. 调用方收尾改造 ([verl/workers/engine_workers.py](#)) : `update_weights` 中模型 offload 之后的 `aggressive_empty_cache(force_sync=True)` 换成 `get_torch_device().synchronize() + get_torch_device().empty_cache()`, 保留前序 `non_blocking D2H` 拷贝所需的同步语义 (PR body 引 #7848 实测 <0.8 ms), 但不触发 full GC; 同时新增 `log_gpu_memory_usage("After offload model to cpu")`。
4. 引擎与 Megatron 工具侧 GC 清理: `verl/workers/engine/{fsdp,torchitan,automodel}/transformer_impl.py` 的 `to()` 只在 `load-to-GPU` 分支删除 `gc.collect()` (offload 分支原本就没有), 并移除各自 `import gc`; `verl/utils/megatron_utils.py` 删除四处 `gc.collect()` (`offload_megatron_model_to_cpu`、`load_megatron_model_to_gpu`、`offload_megatron_copy_params`、`load_megatron_optimizer`) 与 `import gc`, 这些点位保留 `get_torch_device().empty_cache()` 作为显存回收手段。
5. 测试与配套: 第二个 commit “fix ci” 修正 `tests/utils/test_megatron_bshd_preprocess.py::test_preprocess_thd_engine_pads_multidimensional_router_data` 的 CP rank 0 期望值 (`[2, 3, 1, 1] → [2, 3, 0, 1]`, 对应 padding 后 roll 得到的全局行 `{0, 1, 6, 7}`), 与本 PR 的 GC 改动无因果关系, 属于夹带的 CI 修复。作者在 PR body 中承认 `tests/utils/test_bucketed_weight_transfer.py` 尚未补充“最后一帧 ACK 在 buffer 释放之后到达”的断言, 显存释放行为依赖真实 CUDA IPC, CPU 不可复现。

关键文件:

- `verl/workers/rollout/vllm_rollout/bucketed_weight_transfer.py` (模块 权重同步; 类别 source; 类型 core-logic; 符号 `BucketedWeightSender.async_send_weights`, `BucketedWeightSender._cleanup`, `BucketedWeightReceiver.init`, `BucketedWeightReceiver.receive_weights`) : 本 PR 的实质改动所在: 重排接收方最后一桶的 ACK 时机、前置 `del weights, tensor`、引入 `_ack_pending` 守卫并删除两处 `gc.collect()`, 是消除 CUDA IPC limbo 竞态和 2 GiB buffer 滞留的核心。
- `verl/workers/engine_workers.py` (模块 权重更新; 类别 source; 类型 core-logic; 符号 `update_weights`) : 权重同步的调用方: `update_weights` 收尾处把 `aggressive_empty_cache(force_sync=True)` 换成显式 `synchronize() + empty_cache()`, 是本次 GC 清理在关键路径上的直接落点, 并补了 offload 后的显存日志。
- `verl/workers/engine/fsdp/transformer_impl.py` (模块 训练引擎; 类别 source; 类型 core-logic; 符号 `to`) : FSDP 引擎 `to()` 的 `load-to-GPU` 分支删除 `gc.collect()` 与 `import gc`, 是三个引擎中同一模式的代表; offload 分支本就无 GC, 说明改动只针对加载方向。

- verl/workers/engine/torchtitan/transformer_impl.py (模块 训练引擎; 类别 source; 类型 core-logic; 符号 to) : TorchTitan 引擎 to() 中 load-to-GPU 分支的 gc.collect() 与 import gc 被移除, 属同一批引擎侧 GC 清理。
- verl/workers/engine/automodel/transformer_impl.py (模块 训练引擎; 类别 source; 类型 core-logic; 符号 to) : Automodel 引擎 to() 中 load-to-GPU 分支的 gc.collect() 与 import gc 被移除, 属同一批引擎侧 GC 清理。
- verl/utils/megatron_utils.py (模块 模型卸载; 类别 source; 类型 core-logic; 符号 offload_megatron_model_to_cpu, load_megatron_model_to_gpu, offload_megatron_copy_params, load_megatron_optimizer) : Megatron 侧四处 offload / load 函数统一去掉 gc.collect(), 是本 PR 删除 GC 调用最密集的文件 (-5 行, 含 import gc) 。
- tests/utils/test_megatron_bshd_preprocess.py (模块 序列预处理; 类别 test; 类型 test-coverage; 符号 test_preprocess_thd_engine_pads_multidimensional_router_data) : 第二个 commit “fix ci” 修正 THD/CP padding 用例的期望值, 与本 PR 的 GC 主题无关, 属于夹带的 CI 修复, 需单独评估其正确性。

关键符号: BucketedWeightReceiver.receive_weights, BucketedWeightReceiver._cleanup, BucketedWeightSender.async_send_weights, BucketedWeightSender._cleanup, update_weights, to, offload_megatron_model_to_cpu, load_megatron_model_to_gpu, offload_megatron_copy_params, load_megatron_optimizer

关键源码片段

verl/workers/engine_workers.py

权重同步的调用方: `update_weights` 收尾处把 `aggressive_empty_cache(force_sync=True)` 换成显式 `synchronize() + empty_cache()`, 是本次 GC 清理在关键路径上的直接落点, 并补了 offload 后的显存日志。

```
# verl/workers/engine_workers.py —— update_weights 收尾段 (colocated 权重 refit 路径)
async def update_weights(self, global_steps: int = None, mode: str = "auto"):
    ...
    # 3. sync weights: For SGLang, we need base first (when needed), then adapter/merged
    await self.rollout.update_weights(
        per_tensor_param, peft_config=peft_config, base_sync_done=True, global_steps=global_steps
    )
    log_gpu_memory_usage("After update_weights", logger=logger)

    # 3. offload model to cpu
    if self.actor.engine.is_param_offload_enabled:
        self.actor.engine.to("cpu", model=True, optimizer=False, grad=False)
        # 原实现是 aggressive_empty_cache(force_sync=True), 它会顺带做一次 full GC;
        # 这里拆成显式的 device synchronize + empty_cache, 保留前序 non_blocking
        # D2H 拷贝所需的同步语义 (#7848 实测 <0.8 ms), 但不触发 Python 全量 GC
        get_torch_device().synchronize()
        get_torch_device().empty_cache()
```

```

log_gpu_memory_usage("After offload model to cpu", logger=logger)

# 4. resume kv_cache
if self.config.rollout.free_cache_engine:
    # All trainer ranks enter this naive sync path via ONE_TO_ALL. Wait for
    # their offload and device cleanup before a leader wakes the rollout
    # workers; local device synchronization alone cannot prevent OOM on peers.
    await asyncio.to_thread(torch.distributed.barrier)
    await self.rollout.resume(tags=["kv_cache"])
log_gpu_memory_usage("After resume kv_cache", logger=logger)

self.base_sync_done = True
set_expandable_segments(True)

```

verl/workers/engine/fsdp/transformer_impl.py

FSDP 引擎 `to()` 的 load-to-GPU 分支删除 `gc.collect()` 与 `import gc`，是三个引擎中同一模式的代表；offload 分支本就无 GC，说明改动只针对加载方向。

```

# verl/workers/engine/fsdp/transformer_impl.py —— 只删 load-to-GPU 分支的 gc.collect()
# torchtitan / automodel 的 transformer_impl.py::to() 使用完全相同的处理方式
def to(self, device: str, model: bool = True, optimizer: bool = True, grad: bool = True):
    """
    Move FSDP model and/or optimizer to CPU or GPU with offload support.
    Note that this function executes irrespective of offload config. It serves as manual control
    """

    super().to(device=device, model=model, optimizer=optimizer, grad=grad)

    if self.engine_config.forward_only:
        # force cpu_offload
        return

    device_name = get_device_name()
    assert device in (device_name, "cpu")
    if device == device_name:
        if model:
            load_fsdp_model_to_gpu(self.module)
        if optimizer and self.optimizer is not None:
            load_fsdp_optimizer(self.optimizer, device)
        # 这里原有的 gc.collect() 被删除：证据来自 #7780 对 *_device_load 点位的测量
        # （456 次稳态调用、0 CUDA allocated delta、无可重复 RSS 下降），
        # 而非本 PR 在 colocated IPC 路径上的 profile
    elif device == "cpu":
        if model:
            offload_fsdp_model_to_cpu(self.module)
        if optimizer and self.optimizer is not None:
            offload_fsdp_optimizer(self.optimizer)
        # offload 分支原本就没有 gc.collect()，本次保持不动
    else:
        raise ValueError(f"Invalid device type: {device}")

```

评论区精华

本 PR 的 `comments_count` 与 `review_comments_count` 均为 0，没有任何 reviewer 评论，因此所有“讨论”都来自作者在 PR body 中的自述分析，可信度依赖作者自证。三个最有价值的论点：

1. GC 是竞态的意外遮羞布：“the GC was functioning as an accidental sleep, papering over a cross-process race”，并给出 A/B/C 三组快照证据，其中 B 组显示没有 `free_requested` 事件、block 停在 `active_allocated + 2031.8 MiB inactive`，稳定 4 GiB 而非无界增长（即“一轮深的泄漏”）。
 2. ACK 时序是结构性缺陷而非偶发：接收方在释放 mapping 之前 ACK 最后一桶，发送方 `del self.buffer` 时消费者引用计数非 0，PyTorch 将 block 停在 `CudaIPCSentDataLimbo_`，而 `ipc_collect()` 只能排空计数已归零的条目，`empty_cache()` 又动不了分配器仍视为 live 的块。
 3. 明确的验证边界自陈：Megatron/TorchTitan/Automodel 引擎、SGLang、`use_shm=True` 共享内存路径、LoRA 权重同步、多节点、`_direct_send_large_weight`（无单权重超过 2 GiB）均未实测；`{fsdp,torch,titan,automodel}` 的 GC 移除证据来自 #7780 对 `*_device_load` 点位的测量，而非本 PR 的 profile。文章还提到与 #7366（ACK 超时与错误 ACK）需要一次 trivial rebase。
- 能否直接删除 bucketed 路径的 `gc.collect()` (correctness): 必须先修 ACK 时序再删 GC，否则每轮滞留 2048 MiB bucket buffer；本 PR 按此顺序实现，配置 C 的 free 归因回到 `bucketed_weight_transfer.py` 的 `del self.buffer`，`ipc_collect()` 产生 0 个 allocator 事件。
 - 接收方最后一桶 ACK 的时机与 CUDA IPC limbo (design): 方案落地：`_cleanup` 不再先 `close socket`，最后一帧 ACK 在 `buffer/shm/ipc_collect/empty_cache` 之后补发；`_direct_send_large_weight` 路径因 `del weights, tensor` 前置而“按构造”同样受益，但缺少实测。
 - 引擎侧 `gc.collect()` 移除的验证边界 (testing): 作者以 TODO 与显式 caveat 形式记录缺口，未在本 PR 闭环；`_direct_send_large_weight` 也被标注为“unmeasured”。
 - 接收方 `_cleanup` 异常时的发送方阻塞 (other): 由分析者补充识别，PR 内未讨论、未处理。

风险与影响

- 风险：
 1. ACK 推迟引入的阻塞风险（最高优先级）：最后一帧 ACK 现在依赖 `_cleanup` 正常执行完毕。若接收方在 `_cleanup` 中（`get_torch_device().synchronize()`、`ipc_collect()`、`empty_cache()`、`shm.close()`）抛异常或进程被 kill，ACK 永不发出，发送方 `BucketedWeightSender.async_send_weights` 的 `self.socket.recv()` 会永久阻塞；本 PR 未加超时或兜底 ACK，#7366 的 ACK 超时机制是唯一的缓解路径且尚未落地。
 2. 关键路径协议变更、测试缺口明显：`bucketed_weight_transfer.py` 的 ACK 顺序、`_ack_pending` 守卫、socket 关闭顺序都属于跨进程时序契约，但本 PR 新增测试为 0；`tests/utils/test_megatron_bshd_preprocess.py` 的改动与本次功能无关。显存收益（2 GiB 不滞留）只能用真实 CUDA IPC 复现，CI 无法守门。

3. 未覆盖路径的回归不确定性：_direct_send_large_weight 只是“按构造”成立而未实测；use_shm=True、SGLang、LoRA 权重同步、多节点的 ACK 语义未被验证；多节点下 ZMQ/IPC 的时延差异可能放大 ACK 推迟带来的等待窗口。
4. 全局删除 gc.collect() 的主机侧影响：megatron_utils.py 的四处与三个引擎 to() 的 GC 移除会改变 Python 对象回收节奏。若存在循环引用（模块级缓存、闭包、TE FP8 workspace 之类的长生命周期对象），主机 RSS 可能缓慢增长；作者的支撑证据仅是 #7780 在等效 *_device_load 点位上“0 CUDA allocated delta、456 次稳态调用无可重复 RSS 下降”，覆盖范围有限。
5. 易静默回归的语义依赖：name = weight = None 依赖 async for 循环变量作用域这一隐式语义，del weights, tensor 与 _ack_pending 的先后顺序也耦合于 REP socket 的严格交替约定，后续重构（如 #7591 / #5619 的 overlap / double buffering）很容易在不报错的情况下破坏它。
 - 影响：对训练性能与显存：colocated 权重 refit 每轮省掉约 426 ms 的 GC 停顿（433.3 ms → 7.1 ms），并消除每轮 2 GiB bucket buffer 的滞留（reserved 0.039 GiB vs 4.000 GiB），直接扩大 KV-cache 唤醒时可用的显存余量，与 #7851 的跨 rank barrier 形成互补的 OOM 防护。对系统行为：改变了 vLLM bucketed 权重传输的成功路径 ACK 契约，属于跨进程协议层面的语义调整，影响 sender/receiver 双方的时序假设；同时把 GC 从 engine_workers.py、三个训练引擎 to() 和 megatron_utils.py 的 offload/load 边界统一移除，影响所有使用这些边界的训练配置。对团队与工程范式：本 PR 把“用 gc.collect() 当同步 / 回收手段”判定为反模式，并给出了快照式举证方法，可能使 #7780 那种“把 GC 做成可调开关”的方案失去意义（其测量与本 PR 结论一致：0 CUDA 字节回收）。对用户：无 API、CLI、配置项变更，行为收益透明；但未覆盖 Megatron / SGLang / use_shm / LoRA / 多节点用户，收益与风险在该范围内均未验证。
 - 风险标记：核心路径协议变更，跨进程 ACK 时序，缺少新增测试，_direct_send_large_weight 未实测，无 ACK 超时兜底，多引擎 / 多路径未验证，无 reviewer 复核

关联脉络

- PR #7864 [rollout, perf] fix: avoid full GC before weight resume: 本 PR 的直接前序：同改 verl/workers/engine_workers.py，把权重 resume 前的 aggressive_empty_cache(force_sync=True) 换成轻量 empty_cache；本 PR 继续收尾 refit 路径上剩余的 full GC，并引用其 <0.8 ms 的同步开销测量。
- PR #7851 [worker] fix: sync ranks before resume kv_cache to avoid OOM on non-leader ranks: 同一 update_weights 收尾段（offload → barrier → resume kv_cache）：前序 PR 加了跨 rank barrier 防非 leader OOM，本 PR 消除 bucket buffer 滞留，两者共同决定 KV-cache 唤醒时的可用显存。
- PR #7780 make weight-transfer cleanup GC configurable: PR body 点名关联：把发送方 _cleanup 的 GC 做成可调开关并保留 full-GC 默认，其 4104 次调用 0 CUDA 字节回收的测量与本 PR 结论一致；本 PR 从根因上消除了对该开关的需求。
- PR #7366 surface bucketed receiver failures: PR body 点名关联：为同一 ZMQ bucketed 协议加入 ACK 超时与显式错误 ACK，改善失败路径；本 PR 改动成功路径的 ACK 顺序，两者需一次 trivial rebase，且是覆盖本 PR 新增阻塞风险的唯一机制。

- PR #7702 [megatron] fix: pad VLM THD input to TP/CP alignment before SP scatter:
本 PR 第二个 commit “fix ci” 修正 tests/utils/test_megatron_bshd_preprocess.py 中 THD/CP padding 的期望值，与该 PR 同属 mcore THD padding 面，推测是前序改动遗留的用例期望过时。