PR #48596 完整报告

vllm-project/vllm

[Bugfix][KV Offloading] Offload last block at request finish and prevent reuse race

合并时间：2026-07-17 21:50

原文链接：http://prhub.com.cn/vllm-project/vllm/pull/48596

执行摘要

- 一句话：修复请求结束时末块 KV 未卸载及块重用竞态
- 推荐动作：建议合并。该修复解决了实际性能与正确性问题，实现简洁。值得关注的决策：链入 finished_req_ids 简化循环、worker 中提交顺序的调整。代码 review 时可注意调度器状态管理的复杂性。

功能与动机

PR body 指出：_build_store_jobs 在 schedule() 时执行，此时 EOS 尚未处理，最后一个块仍是部分块而被跳过。request_finished 在 EOS 之后调用，但未触发卸载逻辑，导致该块被静默丢弃。另外，由于存储作业被延迟提交，当块被重用且 handle_preemptions 中的 wait 调用时，作业可能尚未 submit，造成 wait 空操作，块被覆盖。

实现拆解

1. 提取可卸载 token 计算：在 scheduler.py 新增 _calc_num_offloadable_tokens 方法，统一计算当前请求的可卸载 token 数，对于已结束请求直接使用 req.num_tokens。
2. 状态清理辅助：新增 _maybe_cleanup_finished_req 方法，在请求完成且无 in-flight 传输 job 时删除 _req_status 条目。
3. 修改构建 store job 的循环：在 _build_store_jobs 中使用 itertools.chain 将 finished_req_ids 混入原有 num_scheduled_tokens 迭代中。对已完成请求使用 req.num_tokens 作为计算基础，使其最后一个完整块参与卸载决策。循环末尾调用 _maybe_cleanup_finished_req。
4. 保持请求状态存活：在 request_finished 中调用 req_status.update_offload_keys()，并确保 req_status 不被立即删除（通过返回 True 延迟清理）。
5. 修复 worker 竞态：在 worker.py 的 handle_preemptions 中，先遍历 jobs_to_flush 从 store_jobs 中 pop 对应条目追加到 _unsubmitted_store_jobs，随后通过现有循环批量 submit_store，最后调用 wait 保证阻塞有效。
6. 测试覆盖：新增 test_last_block_offloaded_at_request_finish 测试，验证 req_status 在请求完成后保持存活；更新 test_two_groups_full_and_sliding_window 以符合新行为。

关键文件：

- vllm/distributed/kv_transfer/kv_connector/v1/offloading/scheduler.py（模块 卸载调度；类别 source；类型 core-logic；符号 _calc_num_offloadable_tokens,

_maybe_cleanup_finished_req, _build_store_jobs）：核心修复：新增辅助函数并修改 _build_store_jobs 以处理 finished_req_ids，确保最后一个块被卸载。

- vllm/distributed/kv_transfer/kv_connector/v1/offloading/worker.py（模块 卸载执行；类别 source；类型 core-logic；符号 handle_preemptions）：修复 handle_preemptions 中的竞态：在 wait 前提交 jobs_to_flush 的 store，确保阻塞正确。
- tests/v1/kv_connector/unit/offloading_connector/test_scheduler.py（模块 测试；类别 test；类型 test-coverage；符号 test_last_block_offloaded_at_request_finish）：新增测试验证末块卸载，并更新现有测试以适配新行为。

关键符号：_calc_num_offloadable_tokens, _maybe_cleanup_finished_req, _build_store_jobs, handle_preemptions

关键源码片段

vllm/distributed/kv_transfer/kv_connector/v1/offloading/scheduler.py

核心修复：新增辅助函数并修改 _build_store_jobs 以处理 finished_req_ids，确保最后一个块被卸载。

```
def _calc_num_offloadable_tokens(
    self, req_status: RequestOffloadState, num_computed_tokens: int
) -> int:
    '计算实际可卸载的 token 数量，用于已结束请求和运行中请求。'
    num = min(num_computed_tokens, req_status.req.num_tokens)
    max_offload = req_status.max_offload_tokens
    if max_offload is not None:
        num = min(num, max_offload)
    if self.config.offload_prompt_only:
        num = min(num, req_status.req.num_prompt_tokens)
    return num

def _maybe_cleanup_finished_req(
    self, req_id: str, req_status: RequestOffloadState
) -> None:
    '当请求完成且无 in-flight 任务时清理状态。'
    if req_status.req.is_finished() and not req_status.transfer_jobs:
        del self._req_status[req_id]

# 在 _build_store_jobs 中：
for req_id in chain(
    scheduler_output.num_scheduled_tokens,
    scheduler_output.finished_req_ids or (),
):
    req_status = self._req_status.get(req_id)
    if req_status is None:
        continue
    req = req_status.req
    if req.is_finished():
        num_tokens_after_batch = req.num_tokens
```

```
        else:
            num_scheduled = scheduler_output.num_scheduled_tokens[req_id]
            num_tokens_after_batch = req.num_computed_tokens + num_scheduled
        # ... 后续现有逻辑 ...
        # 循环末尾：
        self._maybe_cleanup_finished_req(req_id, req_status)
```

vllm/distributed/kv_transfer/kv_connector/v1/offloading/worker.py

修复 handle_preemptions 中的竞态：在 wait 前提交 jobs_to_flush 的 store，确保阻塞正确。

```
    def handle_preemptions(self, kv_connector_metadata: OffloadingConnectorMetadata):
        # 将 jobs_to_flush 中的 store job 提前提交以保证 wait 生效。
        if kv_connector_metadata.jobs_to_flush:
            for job_id in kv_connector_metadata.jobs_to_flush:
                entry = kv_connector_metadata.store_jobs.pop(job_id, None)
                if entry is not None:
                    self._unsubmitted_store_jobs.append(
                        (job_id, entry.src_spec, entry.dst_spec)
                    )
        # 提交所有缓存的 store job（包括上面新加入的）。
        for job_id, src_spec, dst_spec in self._unsubmitted_store_jobs:
            self.worker.submit_store(job_id, src_spec, dst_spec)
        self._unsubmitted_store_jobs.clear()

        if kv_connector_metadata.jobs_to_flush:
            self.worker.wait(kv_connector_metadata.jobs_to_flush)
```

评论区精华

主要讨论围绕简化实现展开。orozery 建议直接使用 itertools.chain 在 _build_store_jobs 中处理 finished_req_ids，而不是引入 separate function，并希望降低测试辅助工具的改动。作者采纳后进行了代码精简。此外，orozery 指出在 handle_preemptions 中应使用 pop 从 store_jobs 中取出条目而非 get，以确保条目被移除，避免后续误用。作者按建议修改。

- 简化 _build_store_jobs 使用 itertools.chain (design): 作者采纳，重构后使用 chain 并在循环内判断 is_finished，代码更简洁。
- worker 中修复 store 提交顺序 (correctness): 作者按建议实现，使用 pop 确保条目移除，然后通过统一循环提交。
- 避免修改测试工具文件 (test): 作者移除对 utils.py 的修改，仅使用现有工具编写测试。

风险与影响

- 风险：修改了卸载调度时序，可能引入延迟卸载或状态残留。worker 中提交顺序提前，可能改变异步 store 的延迟分布，但不影响正确性。对 finished_req_ids 集合的依赖要求上游确保其正确性。测试覆盖了主要场景但未覆盖所有并发条件。
- 影响：影响使用 KV offloading（异步调度）的用户，提升前缀缓存命中率，减少 prefill 计算。非 offloading 用户无影响。代码改动集中在 offloading 模块，相对独立。测试覆盖增

强降低了回归风险。

- 风险标记：调度时序变更，store 提交顺序提前，依赖 finished_req_ids 正确性

关联脉络

- PR #47107 [OffloadingConnector] Kick off final block offload at request finish: 相同修复目标但被本 PR 替代并关闭。
- PR #41777 [Bugfix] Flush final KV block when SimpleCPUOffload request finishes: 类似 bug 的不同组件修复，但原理相似。