PR #42959 完整报告

vllm-project/vllm

[BugFix][kv_offload]: Prevent offloading stale sliding window blocks

合并时间：2026-06-02 10:59

原文链接：http://prhub.com.cn/vllm-project/vllm/pull/42959

执行摘要

- 一句话：修复滑动窗口块在卸载失败后变脏的问题
- 推荐动作：建议审核者重点审查 _update_req_states 中的全量遍历逻辑及其对性能的影响，确认设计权衡合理。同时鼓励在滑动窗口功能相关的集成测试中运行本 PR 的新测试用例。总体修复思路正确，值得精读。

功能与动机

在滑动窗口卸载中，如果滑动窗口块未立即卸载（例如 CPU 分配失败），该块可能被 remove_skipped_blocks 释放并重新分配给不同的逻辑块。此前该情况未被检测到，可能导致 KV 数据错误卸载或断言错误（如 _remove_pending_job 中的 KeyError）。详见 Issue #42761。

实现拆解

1. 在 OffloadingConnectorScheduler.__init__ 中新增 _current_batch_allocated_block_ids 集合，用于记录当前 engine step 分配的 GPU block ID。
2. 在 update_state_after_alloc 中，遍历 group_blocks 时将非零 block_id 加入该集合。
3. 重命名并重构 _build_store_jobs 为 _update_req_states，使其在更新请求状态时，检测滑动窗口组的块 ID 是否在当前步分配集合中。如果不在，说明该块已被释放并重新分配，将其 block_id 置零，防止后续卸载使用错误 ID。同时，在新的块 ID 加入前，也检查 pending_jobs 冲突，触发 flush。
4. 移除了旧代码中在 update_state_after_alloc 中针对 _block_id_to_pending_jobs 的 fence 逻辑，该逻辑现在由 _update_req_states 统一处理。
5. 新增回归测试，模拟 prepare_store 失败导致块未成功卸载，随后滑动窗口推进释放块并重用，最终 prepare_store 成功时必须跳过脏块。

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 _update_req_states, update_state_after_alloc, _build_store_jobs, _current_batch_allocated_block_ids）：核心逻辑修改：新增 current_batch_allocated_block_ids 跟踪，重构 _build_store_jobs 为 _update_req_states，加入 stale block 检测与清零逻辑

- tests/v1/kv_connector/unit/offloading_connector/test_scheduler.py（模块 测试；类别 test；类型 test-coverage；符号 test_stale_sliding_window_block_after_prepare_store_failure）：新增回归测试，模拟 prepare_store 失败后滑动窗口块被释放和重分配场景，验证不会触发 KeyError

关键符号：_update_req_states, update_state_after_alloc, test_stale_sliding_window_block_after_prepare_store_failure

关键源码片段

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

核心逻辑修改：新增 current_batch_allocated_block_ids 跟踪，重构 _build_store_jobs 为 _update_req_states，加入 stale block 检测与清零逻辑

```
# scheduler.py - _update_req_states 方法（核心修复逻辑）
def _update_req_states(self, scheduler_output: SchedulerOutput) -> None:
    """更新请求状态，并检测因滑动窗口释放而导致的脏块 ID。"""
    # new_block_ids_end 记录每个滑动窗口组中旧块 ID 的数量，
    # 用于区分哪些是新分配块 ID。
    new_block_ids_end: dict[str, tuple[int, ...]] = {}

    for req_id, new_block_id_groups, preempted in yield_req_data(scheduler_output):
        req_status = self._req_status[req_id]
        req_status.update_offload_keys()

        if preempted:
            for group_state in req_status.group_states:
                group_state.block_ids.clear()

        if new_block_id_groups:
            if self._sliding_window_groups:
                # 记录每个滑动窗口组现有块 ID 的长度
                new_block_ids_end[req_id] = tuple(
                    len(req_status.group_states[grp_idx].block_ids)
                    for grp_idx in self._sliding_window_groups
                )
            req_status.update_block_id_groups(new_block_id_groups)

            # 处理与新块 ID 冲突的待处理存储作业
            if self._block_id_to_pending_jobs:
                new_blocks_flat = [bid for new_blocks in new_block_id_groups for bid in new_blocks]
                if not self._block_id_to_pending_jobs.keys().isdisjoint(new_blocks_flat):
                    # 触发 flush 以释放旧的待处理作业
                    self._current_batch_jobs_to_flush.update(
                        jid
                        for bid in new_blocks_flat
                        for jid in self._block_id_to_pending_jobs.get(bid, ())
                    )
```

```
        # 针对滑动窗口组检测脏块 ID
        for grp_idx in self._sliding_window_groups:
            group_state = req_status.group_states[grp_idx]
            start = new_block_ids_end.get(req_id, (0,))[list(self._sliding_window_groups).index(grp_
            idx)]
            for i in range(start, len(group_state.block_ids)):
                bid = group_state.block_ids[i]
                # 如果该块 ID 不在当前步分配的块 ID 集合中，说明已被释放重用
                if bid != 0 and bid not in self._current_batch_allocated_block_ids:
                    # 置零表示该块为脏，后续卸载将跳过它
                    group_state.block_ids[i] = 0
```

tests/v1/kv_connector/unit/offloading_connector/test_scheduler.py

新增回归测试，模拟 prepare_store 失败后滑动窗口块被释放和重分配场景，验证不会触发 KeyError

```
# test_scheduler.py - 新增回归测试
@pytest.mark.parametrize("async_scheduling", [True, False])
def test_stale_sliding_window_block_after_prepare_store_failure(
    request_runner, async_scheduling: bool
):
    """回归测试：prepare_store 失败后滑动窗口块被释放并重新分配的情况。"""
    block_size = 4
    sliding_window = 8 # 2 blocks window
    num_gpu_blocks = 4 # 紧预算，确保 freed 块立即被重用

    kv_cache_groups = [
        KVCacheGroupSpec(
            ["layer0"],
            SlidingWindowSpec(
                block_size=block_size,
                num_kv_heads=1,
                head_size=1,
                dtype=torch.float32,
                sliding_window=sliding_window,
            ),
        ),
    ]

    runner = request_runner(
        block_size=block_size,
        num_gpu_blocks=num_gpu_blocks,
        async_scheduling=async_scheduling,
        kv_cache_groups=kv_cache_groups,
    )

    # 请求具有 3 个 prompt 块（0,1,2），窗口后块 0 将被释放
    runner.new_request(token_ids=[0] * block_size * 3)
```

```
# 第一步：prepare_store 失败，块未卸载
runner.manager.prepare_store.side_effect = lambda keys, req_context: None
runner.run(decoded_tokens=[0])
runner.manager.prepare_store.assert_called()

# 第二步：解码产生又一个块（需求 3 个块）
# 滑动窗口导致块 0 被释放，然后立即被重用作块 3 的位置
runner.manager.prepare_store.side_effect = lambda keys, req_context: None
runner.run(decoded_tokens=[0] * block_size)

# 第三步：prepare_store 现在成功
runner.manager.prepare_store.side_effect = lambda keys, req_context: (
    generate_store_output(keys)
)
# 预期只存储位置 2、3 的块（位置 0 的块已被释放并变为脏）
runner.run(
    decoded_tokens=[EOS_TOKEN_ID],
    expected_stored=(2, 3),
    expected_flushed=(2, 3) if not async_scheduling else (),
)
```

评论区精华

@gemini-code-assist[bot]建议：当前 _update_req_states 遍历所有 _req_status 中的请求，在大规模部署中可能带来性能开销，建议仅遍历当前调度输出中的请求。

@orozery回应：限制范围不够，因为即使请求当前没有调度令牌，其状态中可能残留脏块，且只有一次检测机会。必须全局遍历。

结论：保持当前遍历所有请求的设计，暂不优化。

- 优化 stale block 检测的遍历范围 (performance): 保持当前遍历所有请求的设计，暂不优化

风险与影响

- 风险：核心路径变更：涉及 kv_offload 的调度核心逻辑，修改了请求状态更新路径，可能影响非滑动窗口场景。 性能开销：_update_req_states 中遍历所有 _req_status，在请求数量大时可能增加调度步骤延迟，但滑动窗口场景本身较少，开销仍在可控范围。 回归风险：新增测试覆盖了主要边界，但仍需关注大规模 batch 下的稳定性。
- 影响：影响范围：仅在使用 kv_offload 并启用滑动窗口的场景下生效，其他用户无影响。 影响程度：修复了潜在的数据损坏和进程崩溃问题，提升了系统可靠性。 对团队：需要关注合并后是否有相关回归报告。
- 风险标记：核心路径变更，全量遍历可能带来性能开销

关联脉络

- 暂无明显关联 PR