PR #52372 完整报告

vllm-project/vllm

[Bugfix][Mooncake] Reference GPU blocks for in-flight store jobs and key the store ledger by store_job_id

合并时间：2026-08-17 07:09

原文链接：http://prhub.com.cn/vllm-project/vllm/pull/52372

执行摘要

- 一句话：修复 Mooncake 抢占下 KV 污染与 req_id ABA，改按 store_job_id 引用块记账
- 推荐动作：值得精读。这是分布式 KV 缓存连接器中异步作业生命周期与 id 复用（ABA）问题的教科书级修复：一是用全局单调 store_job_id 替代 req_id 计数，从数据结构上消除旧代 job 影响新一代的可能；二是把块生命周期从“请求完成时延迟释放”改为“job 引用计数归零才释放”，同时覆盖正常完成与抢占两条路径；三是 review 中关于 assert-vs-raise、eviction 释放顺序、try/finally 单出口的讨论都值得关注。对维护 kv-connector 或任何后台线程异步读取 GPU 内存的模块的工程师，建议通读 scheduler.py 与 worker.py 的核心 diff。

功能与动机

本 PR 修复两个独立报告的并发缺陷，二者都在抢占场景下触发：#52360 中 storing 是异步的——step 0 调度器只是把请求 enqueue 进 Mooncake 队列就返回，真正的 GPU 读与 PUT 发生在后台线程；step 1 起调度器可以抢占该请求并把块交还 free pool。worker 直到下一次 IPC 才得知抢占，这个窗口内发出的任何 PUT 都会读到另一请求的 KV。PR body 强调“wrong data does not heal itself”：key 命名固定 token 区间，一旦不属于该 key 的 KV 被写进去，后续所有命中同一前缀的请求都会读错，爆炸半径随时间扩大。旧代码唯一的保护 request_finished -> delay_free_blocks 只覆盖正常完成路径，preemption 走 build_connector_meta 重置 tracker 并立即释放块，无人知道还有排队的 job 需要读这些块。#51637 中 store 线程的计数器只按 req_id 键控，无法区分属于当前这一代请求的 job 与上一代遗留、尚未执行的 job。遗留 job 完成时会扣减新一代的计数：计数到 0 即 retire，当前代仍在进行的 save 被静默丢弃；继续扣减变负后永远无法归零，finished_sending 永不发出、scheduler 永不释放块、引擎最终 KV 饥饿；遗留 job 还可能回退新一代的 resume offset（_saved_offset）或在压力下把新一代标记为 skipped。

实现拆解

1. 引入 job 身份与完成汇报契约（data.py）：ReqMeta 新增 store_job_id: int | None 字段，注释明确说明 req_id 会在抢占恢复后复用，不能作为 job 身份；新增 MooncakeStoreWorkerMetadata（completed_saves: dict[int, int]）并实现 aggregate()，用于跨 rank 聚合“每个 store job 有多少 rank 已完成”，通过 KVConnectorWorkerMetadata 基类接入现有 worker 元数据通道。

2. 调度侧 GPU 块引用管理（scheduler.py）：新增 bind_gpu_block_pool() 在调度器与 BlockPool 之间建立绑定；build_connector_meta() 末尾调用新增的 _reference_save_blocks(meta)，为每个 can_save 的 ReqMeta 分配单调递增的 store_job_id，把 partial-tail CoW 块（core 刻意放在请求 block table 之外、但 worker 同样异步 DMA）以及该 job 可能读取的所有 allocated block（不止本次 token 范围，因为 rank 的 resume offset 可能滞后）通过 pool.touch() 引用住；update_connector_output() 消费 worker metadata 中的 completed_saves，按 rank 数递减剩余引用，归零后以 tail-first 顺序 pool.free_blocks() 释放；has_pending_push_work() 通过 _pinned_saves 非空判断保持引擎在有 PUT 在飞时持续步进，避免引用永久滞留。
3. 移除旧的 delay-free 路径（connector.py / scheduler.py）：request_finished_all_groups() 不再转发给 scheduler 判断，而是直接返回 (False, None)，因为块的生命周期现在由 job 自身的引用接管；scheduler 侧旧的 request_finished() 判断逻辑整体删除，finished_sending 汇报机制随之退出。
4. worker 侧 ledger 重构与单出口（worker.py）：stored_requests 从 defaultdict[str, int] 改为 dict[str, set[int]]（req_id -> live store job ids）；add_request() 在入队前先注册 job 保证“永不未记账出队”；is_live_store_job() 判断 job 是否属于当前代；finish_store_job() 统一负责从集合中 discard 并累计 _completed_saves；_handle_request() 将 preamble 与 liveness check 全部移入 try 内，使 finally 成为唯一出口，任何异常路径都会 report 从而释放块引用；_record_saved() 与 _mark_request_skipped_for_pressure() 都改为以 job 活度为前置条件，防止遗留 job 回写 offset 或判决 skipped。
5. 测试与验证配套：新增针对引用生命周期（test_store_job_blocks_are_released_once_every_rank_reports、test_partial_tail_cow_block_is_referenced_for_the_job）、跨 rank 完成聚合（test_worker_metadata_aggregates_completions_across_ranks）、遗留 job 不能触碰复用 req_id（test_stale_store_job_cannot_touch_a_reused_request_id）、store 异常时 job 仍被汇报（test_store_sending_thread_reports_job_when_store_raises 等）的定向测试；测试统一改用 _run_store_req 帮助函数走“注册 + 入队 + 执行”的完整路径。PR body 还提供了基于内容指纹的端到端复现与 A/B 性能对比。

关键文件：

- vllm/distributed/kv_transfer/kv_connector/v1/mooncake/store/scheduler.py（模块 存储调度；类别 source；类型 core-logic；符号 bind_gpu_block_pool, request_finished, _reference_save_blocks, update_connector_output）：实现修复核心：新增 _reference_save_blocks 为每个 store job 在 BlockPool 上持有块引用，新增 update_connector_output 消费跨 rank 完成计数并释放引用，新增 has_pending_push_work 保持引擎步进，删除旧的 request_finished delay-free 逻辑。
- vllm/distributed/kv_transfer/kv_connector/v1/mooncake/store/worker.py（模块 存储线程；类别 source；类型 core-logic；符号 add_stored_request, add_request, dec_stored_request, is_live_store_job）：worker 侧 ledger 从 per-req_id 计数重构为 req_id -> {store_job_id 集合}，解决 #51637 的 ABA；_handle_request 改为 try/finally 单出口，保证任何路径都 report 并释放块引用。

- vllm/distributed/kv_transfer/kv_connector/v1/mooncake/store/data.py（模块 数据契约；类别 source；类型 data-contract；符号 MooncakeStoreWorkerMetadata, aggregate）：数据契约变更：ReqMeta 新增 store_job_id 字段，新增 MooncakeStoreWorkerMetadata（含 aggregate），支撑跨 rank 完成计数从 worker 聚合到 scheduler。
- vllm/distributed/kv_transfer/kv_connector/v1/mooncake/store/connector.py（模块 连接器；类别 source；类型 core-logic；符号 bind_gpu_block_pool, has_pending_push_work, build_connector_worker_meta）：连接器入口：新增 bind_gpu_block_pool / has_pending_push_work / build_connector_worker_meta 通道，request_finished_all_groups 简化为无条件不延迟释放，将生命周期完全交给 job 引用。
- tests/v1/kv_connector/unit/test_mooncake_store_scheduler.py（模块 调度测试；类别 test；类型 test-coverage；符号 _make_worker_output, test_preemption_resets_tracker, test_partial_tail_cow_block_is_referenced_for_the_job, test_store_job_blocks_are_released_once_every_rank_reports）：调度侧测试：覆盖所有 rank 报告后才释放引用、partial-tail CoW 块被引用、跨 rank 完成聚合、抢占重置 tracker 等关键路径。
- tests/v1/kv_connector/unit/test_mooncake_store_worker.py（模块 线程测试；类别 test；类型 test-coverage；符号 _run_store_req, test_store_sending_thread_reports_job_when_store_raises, test_store_sending_thread_reports_job_when_the_preamble_raises, test_store_sending_thread_releases_pin_on_batch_put_failure）：worker 侧测试：新增 _run_store_req 统一执行路径，覆盖 store 异常时 job 仍被汇报、遗留 job 不能触碰复用 req_id、失败路径释放引用等关键行为。
- tests/v1/kv_connector/unit/test_mooncake_store_hma_e2e.py（模块 端到端测试；类别 test；类型 test-coverage）：端到端 HMA 测试适配：add_stored_request 改为 add_request，并显式设置 store_job_id，验证新记账路径在 e2e 场景可用。

关键符号：bind_gpu_block_pool, _reference_save_blocks, update_connector_output, has_pending_push_work, request_finished_all_groups, build_connector_worker_meta, add_request, is_live_store_job, finish_store_job, take_completed_saves, _record_saved, _mark_request_skipped_for_pressure, MooncakeStoreWorkerMetadata.aggregate

关键源码片段

vllm/distributed/kv_transfer/kv_connector/v1/mooncake/store/scheduler.py

实现修复核心：新增 _reference_save_blocks 为每个 store job 在 BlockPool 上持有块引用，新增 update_connector_output 消费跨 rank 完成计数并释放引用，新增 has_pending_push_work 保持引擎步进，删除旧的 request_finished delay-free 逻辑。

```
class MooncakeStoreScheduler:
    # ... 此前省略 __init__ 与 build_connector_meta 主流程 ...

    def _reference_save_blocks(self, meta: MooncakeStoreConnectorMetadata) -> None:
        """为每个本 step 发出的 store job 在 GPU block pool 上持有引用。

        worker 在本 step 之后才从这些块 DMA 数据，因此即使请求本身
```

```
        已被释放，只要仍有未完成 job，块就不能回到 free queue。
        引用在 update_connector_output 中按 rank 完成计数递减，
        所有 rank 报告后才会真正释放。
        """
        pool = self._gpu_block_pool
        for req_meta in meta.requests:
            if not req_meta.can_save:
                continue
            assert pool is not None, (
                "GPU block pool must be bound before any store job is emitted"
            )
            # 每个 job 分配全局单调递增 id，作为引擎生命周期内的唯一身份；
            # req_id 会在抢占后复用，不能作为 job 身份。
            req_meta.store_job_id = store_job_id = self._next_store_job_id
            self._next_store_job_id += 1
            block_ids: list[int] = []
            if req_meta.partial_tail_offloads:
                # partial-tail 的 CoW 块被 core 刻意放在请求 block table 之外，
                # 不会出现在 req_meta.block_ids 中，但 worker 同样异步 DMA，
                # 因此必须单独引用；放在列表头部，释放时逆序后恰好垫底。
                block_ids += [bid for _, bid, _ in req_meta.partial_tail_offloads]
            # 引用该 job 可能读取的 * 所有 * allocated block，而不止本次 token
            # 范围：rank 从各自最近成功 offset 恢复，可能滞后于 scheduler，
            # 因此会读到范围以下的任意块。
            block_ids += [bid for group in req_meta.block_ids for bid in group]
            if not block_ids:
                continue
            # remaining 初始化为 world_size：所有 rank 都报告才释放。
            self._pinned_saves[store_job_id] = (block_ids, self._num_workers)
            pool.touch([pool.blocks[bid] for bid in block_ids])

    def update_connector_output(self, connector_output: KVConnectorOutput) -> None:
        """消费 worker 元数据，释放所有 rank 均已完成 job 的块引用。"""
        meta = connector_output.kv_connector_worker_meta
        if not isinstance(meta, MooncakeStoreWorkerMetadata):
            return
        pool = self._gpu_block_pool
        assert pool is not None
        for store_job_id, count in meta.completed_saves.items():
            pinned = self._pinned_saves.get(store_job_id)
            if pinned is None:
                # 该 job 没有引用任何块（如空 save），无需处理。
                continue
            block_ids, remaining = pinned
            remaining -= count
            if remaining > 0:
                self._pinned_saves[store_job_id] = (block_ids, remaining)
                continue
            # 到达这里说明所有 rank 都报告完成。注意：若 count 使 remaining
```

```
            # 越过 0，说明存在重复报告或聚合错误，块可能在 DMA 仍在飞时被
            # 释放——这正是本 PR 要消灭的 KV 污染场景，故用 assert 兜底
            #  （review 指出 python -O 会剥离 assert，可改用显式 raise）。
            assert remaining == 0, (
                f"store job {store_job_id} reported by too many ranks"
            )
            del self._pinned_saves[store_job_id]
            # 按 eviction 优先级逆序释放：与既有 tail-first 规则一致，
            # 让共享前缀在 cache 中驻留最久。
            pool.free_blocks(pool.blocks[bid] for bid in reversed(block_ids))

    def has_pending_push_work(self) -> bool:
        """仍有 store job 持有块引用时，保持引擎持续步进。

        完成事件只随 step 的 worker metadata 回到 scheduler；若引擎在
        job 在飞时静默，引用将无限期持有。请求完成不再延迟自身释放后，
        这是唯一驱动引擎继续推进的机制。
        """
        return bool(self._pinned_saves)
```

vllm/distributed/kv_transfer/kv_connector/v1/mooncake/store/worker.py

worker 侧 ledger 从 per-req_id 计数重构为 req_id -> {store_job_id 集合}，解决 #51637 的 ABA；_handle_request 改为 try/finally 单出口，保证任何路径都 report 并释放块引用。

```
class KVCacheStoreSendingThread:
    # 旧结构是 req_id -> 计数（defaultdict[int]），会被抢占后的 req_id
    # 复用破坏（ABA）：遗留 job 扣减新一代计数。新结构改为
    # req_id -> { 存活的 store job id 集合 }，job id 全局唯一，旧代遗留
    # job 不在新一代建立的集合中，因此无法 retire 新一代、无法回退其
    # resume offset、也无法将其标记为 skipped。
    self.stored_requests: dict[str, set[int]] = {}
    # store_job_id -> 本 rank 完成该 job 的报告次数，每 step 清空，
    # 供 scheduler 据此释放块引用。
    self._completed_saves: dict[int, int] = {}

    def add_request(self, request: ReqMeta) -> None:
        # 先注册再入队，保证 job 不会被未记账地取出。
        assert request.store_job_id is not None
        with self.done_task_lock:
            self.stored_requests.setdefault(request.req_id, set()).add(
                request.store_job_id
            )
        super().add_request(request)

    def is_live_store_job(self, req_meta: ReqMeta) -> bool:
        with self.done_task_lock:
            return req_meta.store_job_id in self.stored_requests.get(
                req_meta.req_id, ()
            )
```

```
def finish_store_job(self, req_meta: ReqMeta) -> None:
    """从 ledger 退役一个 job，并汇报其块不再被读取。

    所有离开 job 的路径都必须到达这里，包括跳过与失败：不汇报的
    job 会让其块引用在整个运行期间永久持有。discard 对已退役代
    是 no-op。
    """
    store_job_id = req_meta.store_job_id
    assert store_job_id is not None, (
        "a queued store job always carries a store_job_id"
    )
    with self.done_task_lock:
        live = self.stored_requests.get(req_meta.req_id)
        if live is not None:
            live.discard(store_job_id)
        self._completed_saves[store_job_id] = (
            self._completed_saves.get(store_job_id, 0) + 1
        )

def _handle_request(self, req_meta: ReqMeta):
    # 唯一的 finally 是所有出口：无论 job 以何种方式结束（含 preamble
    # 或 store 调用抛错），都先完成报告再 task_done，scheduler 才能
    # 释放该 job 引用的 GPU 块。若 job 从不汇报，块引用会永久泄漏。
    try:
        lcm_block_size = self.coord.lcm_block_size
        token_len = req_meta.token_len_chunk // lcm_block_size * lcm_block_size
        block_ids_per_group = req_meta.block_ids
        req_id = req_meta.req_id
        current_event = req_meta.current_event

        # 只有当前代的 job 才允许继续执行；旧代遗留 job 直接退役。
        if not self.is_live_store_job(req_meta):
            return

        if self._should_skip_request(req_id):
            logger.debug(
                "Skipping Mooncake store for request %s while CPU/disk "
                "offloading is under pressure",
                req_id,
            )
            return

        # ... 正常的 batch_is_exist / batch_put 主流程（省略） ...
    finally:
        self.finish_store_job(req_meta)
        self.request_queue.task_done()
```

评论区精华

1. 安全不变式用 assert 还是 raise（depthfirst-app[bot]，scheduler.py）：bot 指出 assert remaining == 0 会被 python -O 剥离，若 remaining 因 worker 误报或聚合 bug 变负，块会在 DMA 仍在飞时被释放，恰好复现本 PR 要消灭的 KV 污染；建议改为显式 if remaining != 0: raise RuntimeError(...)，并援引仓库内 example_hidden_states_connector.py:312 的先例。最终合并代码仍保留 assert，此建议未采纳，值得后续跟进。
2. seq 命名晦涩（ivanium）：seq 在 LLM 语境中易与 request/sequence 混淆，作者响应改为 store_job_id，并给出两个理由：仓库已有 _store_jobs / create_store_job 先例，且该名字表达"标识 store job 而非 request"——一个请求会映射到多个 store_job_id。
3. 释放顺序保持 eviction 优先级（ivanium）：free_blocks() 的输入按 eviction 优先级排序，请求块在其他路径都是 tail-first 释放以让共享前缀驻留最久；作者采纳并把 partial-tail pin 移到列表前部，与 pop_blocks_for_free 的 pins + blocks 顺序一致，逆序释放后 pin 恰好垫底。
4. _completed_saves 是否为 1 与单出口兜底（ivanium / chengy-sysu）：ivanium 指出简单 assert 不够（take_completed_saves() 可能清掉中间状态）。作者回应每个 rank 每个 job 恰好完成一次、两个调用点互斥，且另加集合只会无限增长故不加检查；但借此发现 try 之前的前置读取和 liveness check 的早退路径在 raise 时会泄漏块引用，已通过把 try 移到函数顶部、以 finally 作为唯一出口修复（commit a7f30aa98、df20c929）。
5. 后续演进方向（ivanium 总体评论）：建议限制并发 store job 数，避免 Mooncake 压力下 GPU 块被 pin 过久，可在独立 PR 用压力跳过实现；还建议把 worker 侧 stored_requests、_saved_offset、_skip_store_requests、finished_store_req 统一成 StoreRequestState dataclass，方便集中跟踪。

- 安全不变式：assert remaining == 0 在 python -O 下失效 (correctness): 该建议未被采纳，合并代码仍保留 assert；但作者对 worker 单出口的修复从另一侧降低了误报概率。作为安全不变式，后续值得改为显式 raise。
- seq 命名在 LLM 语境下晦涩，改名 store_job_id (design): 已采纳并完成重命名。
- 释放块引用时保持 eviction 顺序（tail-first） (design): 已采纳并配套修正测试断言。
- _completed_saves 计数校验与 try/finally 单出口兜底 (correctness): 以"把 try 移到函数顶部、finally 作为唯一出口"的方式兜底解决，相关失败路径测试已补充。
- 后续方向：限制并发 store job 与统一 worker 状态 (design): 作者与 reviewer 均认可但同意推迟到独立 PR。

风险与影响

- 风险：
  1. remaining 计数不当即复现 KV 污染（scheduler.py update_connector_output）：释放块的前提是 remaining == 0 且用 assert 保证。若 worker 重复报告或聚合出现 bug 使 remaining 变负，块会在 DMA 在飞时被释放——这正是本 PR 要消灭的 #52360 场景。depthfirst-app[bot] 已指出 python -O 下 assert 会被剥离，该安全不变式目前依赖解释器未开优化。
1. 引用覆盖范围保守导致块驻留延长：_reference_save_blocks 引用 job 可能读取的所有 allocated block 而非仅本次 token 范围（因为 rank resume offset 可能滞后），逻辑上正

确但会延长 KV 块驻留；Mooncake 压力下并发 job 多时可能压缩可用 GPU KV 池，ivanium 也建议后续限制并发 job 数。

2. 账本契约变更的潜在缺口：块不被延迟释放的保证完全依赖 _reference_save_blocks 对每个会触发 DMA 的 ReqMeta 都引用到位。任何 can_save=False 但仍产生 DMA 的路径（如 partial-tail offload-only ReqMeta 虽为 can_save=True 但 token_len_chunk=0）都会直接读已释放块；该覆盖面由新增测试支撑，但仍属行为变更风险。
3. finally 内再次抛错的风险：finish_store_job 中的 assert store_job_id is not None 若触发，会掩盖原始异常并跳过 task_done()；不过 add_request 已断言 store_job_id 非空，实际风险低。
4. 测试环境依赖 GPU：PR body 报告 251 个测试中 16 个失败全为 No CUDA GPUs are available（集中于未改动的 P2P 文件），说明该模块 CI 对 GPU 强依赖，无 GPU 环境下无法获得完整回归信号。 - 影响：影响范围集中在使用 MooncakeStoreConnector 做跨实例 / 跨复制体前缀缓存的部署（v1 + kv-connector 路径），不涉及模型执行主链路。正确性上，消除了两类严重问题：静默 KV 污染——错误 KV 写入固定 key 后永久毒化前缀缓存、影响所有后续命中请求，端到端指纹实测 main 上有 8/96 请求读到错误 KV、438 个中毒 key；以及 KV 块泄漏引发的引擎活锁（#51637 的负面计数）。性能上，PR 给出冷存储 A/B：wall +1.3%、TTFT p50 反而快 1.2s、外部前缀缓存查询数完全一致（134,777），结论为无可测量开销。团队与设计层面，该 PR 为 kv-connector 家族确立了两个可复用模式：用全局单调 id 而非请求 id 做异步作业身份以消除 ABA，以及“引用计数 + 单出口 finally”的异步 GPU 内存生命周期管理；KVConnectorWorkerMetadata 基类抽象也可供其他 connector 复用。 - 风险标记：核心记账路径变更，assert 在 python -O 下失效，块引用保守延长驻留，完成汇报依赖步进驱动，测试强依赖 GPU 环境

关联脉络

- PR #51662 Reporter's alternative fix for #51637 (epoch-based _StoreRequestJob): PR body 明确指出第二个 commit 与 #51662 解决同一 issue，二者选一即可；若 #51662 先合入，第二个 commit 可丢弃。
- PR #43742 Supplies the decrement missed on store failure: PR body 说明它补上失败路径遗漏的 decrement，而本 PR 第 2 点解决的是 decrement 落在错误 generation 上的问题，二者互补。
- PR #51595 Aggregates async completions with per-request count but no generation identity: PR body 将其列为相关工作：按 per-request 计数聚合异步完成，但未给 request generation 身份，无法防止 req_id 复用后的 ABA。
- PR #52419 [Bugfix][Spec Decode] Keep EAGLE cache registration on the partial-hash-hit path: 同仓库近期 bugfix，同样处理抢占 / 前缀缓存路径下 KV 协调状态被破坏的问题，体现抢占正确性主题的延续。