PR #39186 完整报告

vllm-project/vllm

[KV Offload] Per-job store completion for CPU offloading connector

合并时间：2026-04-29 13:52

原文链接：http://prhub.com.cn/vllm-project/vllm/pull/39186

执行摘要

- 一句话：逐作业通知 KV 卸载完成，加速前缀缓存重用
- 推荐动作：此 PR 值得深刻理解，尤其关注 TransferJobStatus 的 fencing 机制和 OffloadingWorkerMetadata 的聚合模式。建议团队在后续开发中复用 build_connector_worker_meta 模式来收集异步传输完成状态。作者与 reviewer 的多次迭代展示了良好的工程权衡，设计文档可以从中提炼。

功能与动机

当 GPU→CPU KV 缓存存储完成时，调度器目前只在整个请求生成完成（EOS）时才得知。CPU 上的块在多个步骤中保持不可见（ref_cnt=-1），共享相同前缀的未来请求无法重用这些块，必须重新计算。此 PR 报告逐作业、逐步的存储完成情况，以便调度器可以在单个块组的 DMA 完成时立即调用 complete_store，使卸载的块立即可加载。

实现拆解

1. 引入新的数据结构：在 common.py 中添加 TransferJob（作业描述）、OffloadingWorkerMetadata（工人元数据），并在 scheduler.py 中添加 TransferJobStatus（调度器侧作业状态），分别封装作业 ID、请求 ID、传输规格、待处理工人计数和关联的 offload keys。
2. 作业 ID 统一分配：调度器使用 _job_counter 为每个存储和加载操作分配全局唯一 ID，替代原先按请求 ID 索引的字典；存储作业和加载作业共享 ID 空间，通过 TransferJobStatus.is_store 区分。
3. 请求状态精简：RequestOffloadState 将原来的 load_job/store_jobs 合并为单一 transfer_jobs: set[int]，并维护 load-or-store 不变性（一个请求要么有一个加载作业，要么有一个或多个存储作业，不能同时存在）。
4. 工人侧简化：OffloadingConnectorWorker 移除内部的 _job_counter、_load_job、_store_jobs 等字段，直接使用调度器分配的作业 ID；新增 _connector_worker_meta 用于收集本步完成的作业，由 build_connector_worker_meta() 返回并由调度器聚合。
5. 完成处理与 Fence 机制：调度器在 update_connector_output() 中解析工人元数据，对达到 pending_count=0 的作业调用 complete_store() 或 complete_load()；同时维护 _block_id_to_pending_jobs 索引，当一个已完成请求的存储作业仍在飞行时，该请求释放的 GPU 块不能立即让给新请求加载（通过 request_finished 触发填充，

update_state_after_alloc 和 _build_store_jobs 检查并刷新）。

6. 测试覆盖：新增 test_worker_metadata.py 验证 OffloadingWorkerMetadata.aggregate 的计数正确性；在 test_scheduler.py 中添加 4 个新测试（test_complete_store_called_per_job、test_fence_at_update_state_after_alloc、test_fence_at_build_store_jobs、test_loads_do_not_populate_fence_index），并给 3 个已有测试追加 fence 索引检查。

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 TransferJobStatus, _generate_job_id, _get_reqs_to_store, _build_store_jobs）：核心调度器，引入 TransferJobStatus、作业 ID 生成、逐作业完成处理以及 fence 索引，PR 最重大的变更所在。
- vllm/distributed/kv_transfer/kv_connector/v1/offloading/common.py（模块 公共定义；类别 source；类型 core-logic；符号 TransferJob, OffloadingWorkerMetadata, mark_completed, aggregate）：定义 TransferJob 和 OffloadingWorkerMetadata 两个关键数据结构，是调度器与工人之间传递逐作业完成信息的契约。
- vllm/distributed/kv_transfer/kv_connector/v1/offloading/worker.py（模块 卸载工人；类别 source；类型 core-logic；符号 _generate_job_id, build_connector_worker_meta）：工人侧简化，移除内部作业 ID，改为直接使用调度器分配的作业 ID，新增 _connector_worker_meta 收集完成状态。
- vllm/distributed/kv_transfer/kv_connector/v1/offloading_connector.py（模块 连接器入口；类别 source；类型 core-logic；符号 build_connector_worker_meta）：连接器入口，新增 build_connector_worker_meta 方法桥接调度器与工人之间的元数据传递。
- tests/v1/kv_connector/unit/offloading_connector/test_scheduler.py（模块 调度器测试；类别 test；类型 test-coverage；符号 test_loads_do_not_populate_fence_index, test_fence_at_update_state_after_alloc, test_fence_at_build_store_jobs, test_complete_store_called_per_job）：新增 4 个测试验证逐作业完成和 fence 不变性，并在现有测试中补充断言。
- tests/v1/kv_connector/unit/offloading_connector/test_worker_metadata.py（模块 工人元数据测试；类别 test；类型 test-coverage；符号 test_aggregate_sums_counts, test_aggregate_disjoint_jobs, test_aggregate_multiple_workers）：新增独立测试验证 OffloadingWorkerMetadata.aggregate 的计数、分离和多次聚合正确性。
- tests/v1/kv_connector/unit/offloading_connector/utils.py（模块 卸载测试工具；类别 test；类型 test-coverage）：测试工具，适配新的作业 ID 元数据接口。
- tests/v1/kv_connector/unit/utils.py（模块 全局测试工具；类别 test；类型 test-coverage）：全局测试工具小修改，适配 OffloadingWorkerMetadata。

关键符号：_generate_job_id, _get_reqs_to_store, _build_store_jobs, update_connector_output, build_connector_worker_meta, mark_completed, aggregate, request_finished, handle_preemptions, get_finished

关键源码片段

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

核心调度器，引入 TransferJobStatus、作业 ID 生成、逐作业完成处理以及 fence 索引，PR 最重大的变更所在。

```
@dataclass(slots=True)
class TransferJobStatus:
    """Tracks scheduler-side state for a single transfer job."""

    req_id: ReqId
    # Number of workers still pending. Starts at num_workers,
    # decremented as each worker reports completion. Job is done at 0.
    pending_count: int
    # Offload keys this job covers; passed to manager.complete_*().
    keys: set[OffloadKey]
    is_store: bool
    # GPU blocks the fence tracks. Store src blocks; None for loads.
    gpu_block_ids: list[int] | None = None
```

评论区精华

- 性能扫描优化（gemini-code-assist）：_cleanup_store_jobs_for_req 中线性扫描所有活跃存储作业可能成为瓶颈，建议增加反向映射。作者采用 _req_to_jobs 字典，将清理从 O(N) 降低到 O(1)。
- 统一数据结构（orozery）：将分散的 _store_job_hashes、_store_job_to_req、_store_job_pending_counts 浓缩为单个 dict[int, TransferJobStatus]，并将 OffloadKeys 内聚到状态中。
- 完成报告统一（orozery）：将 completed_store_jobs 与 completed_load_jobs 合并为单个 completed_jobs: dict[int, int]，采用“计数 = 工人数”模型等待所有 TP 工人确认。
- Fence 设计迭代：最初在 request_finished 时填充 fence 索引，后改为在 _build_store_jobs 和 update_state_after_alloc 中按需刷新，避免过早争用。
  - 加载与存储区别对待（orozery）：加载受 delay_free_blocks 保护，无需 fence，最终只在存储侧保留 fence。
  - _cleanup_store_jobs_for_req 线性扫描性能 (performance): 作者增加了反向映射 _req_to_jobs，将清理复杂度从 O(N) 降为 O(1)。
  - 将多个字典合并为单一 TransferJobStatus (design): 作者实现了一个统一的 TransferJobStatus 类，包含 req_id、pending_count、keys 和 is_store。
  - 统一 completed_store_jobs 和 completed_load_jobs (design): 作者合并为单个 OffloadingWorkerMetadata.completed_jobs，聚合时对相同 job_id 累加计数。
  - block reuse fence 的触发时机 (correctness): 最终设计为仅在 request_finished 时填充 _block_id_to_pending_jobs，其他位置通过 isdisjoint 检查并触发 flush。
  - 加载作业不需要 fence (design): 作者移除了加载侧的 fence 检测，只在存储侧保留。

风险与影响

- 风险：尽管 benchmark 显示无回归且 24 个单元测试全部通过，此 PR 仍存在以下风险：

1. 并发正确性：_block_id_to_pending_jobs 的 fence 机制在两个位置触发（update_state_after_alloc 和 _build_store_jobs），如果状态转移遗漏（如请求被提前清理）可能导致断言 assert req_id in self.requests 失败。PR 作者曾在迭代中修复过类似问题（transfer_jobs 生命周期绑定的 bug）。
2. 性能峰值：新增的聚合和字典操作在高 TP 并发（如 8 卡）下可能引入额外延迟，虽然 benchmark 显示无回归，但场景有限。
3. API 兼容性：OffloadingConnectorMetadata 的字段从 reqs_to_load/reqs_to_store 改为 load_jobs/store_jobs，任何外部自定义 connector 需要适配。
4. Fence 的死锁风险：如果某个存储作业因网络延迟迟迟未完成，其 fence 索引会阻止新请求加载同一块，可能导致吞吐下降或调度死锁（暂未实现超时机制）。 - 影响：
   - 对用户：使用 CPU KV offloading 的用户将显著改善前缀缓存重用效率，benchmark 显示中位数 TTFT 降低 ~1.3%，TPOT 几乎不变。
   - 对系统：调度器与工人之间的通信从按请求 ID 变为按作业 ID，为未来支持批量异步传输、流水线传输等高级特性奠定基础。
   - 对团队：此 PR 是 KV offloading 架构从粗粒度（请求级）向细粒度（作业级）演进的关键里程碑，后续可在此基础上实现更灵活的生命周期管理。
   - 风险标记：核心路径变更，并发安全，性能退化风险，API 兼容性变更

关联脉络

- 暂无明显关联 PR