

PR #49502 完整报告

vllm-project/vllm

[3/N][Core][KV Connector] Support reliable partial-tail KV offload for sub-block prompts

合并时间: 2026-07-27 14:22

原文链接: <http://prhub.com.cn/vllm-project/vllm/pull/49502>

执行摘要

- 一句话: 支持子块提示的可靠部分尾部 KV 卸载
- 推荐动作: 建议精读。该 PR 展示了在混合模型场景下, 如何通过分离 hash 粒度与块粒度来实现精确的部分缓存命中。设计上采用了 CoW 目标块暴露、pin 机制、卸载进度分离等手法, 技术值得借鉴。对于维护 MooncakeStore 或类似 KV offload 系统的工程师尤其重要。

功能与动机

参见 Issue #45702。对于 Full Attention + Mamba 混合模型, Mamba 状态块大小 (如 1024 tokens) 决定了全局 KV 块粒度, 导致传统前缀缓存命中率极低。本变更分离 hash 匹配粒度与物理块粒度, 引入 `hash_block_size`, 使得在共享块边界 (block boundary) 之前结束的提示 (sub-block prompts) 也能被缓存和命中, 大幅提升聊天轮次、工具调用边界等场景的缓存复用率。

实现拆解

1. KV 缓存管理器核心转发 (`kv_cache_manager.py` / `single_type_kv_cache_manager.py`)
 - 在 `kv_cache_manager.py` 中新增 `take_partial_tail_offloads()` 方法, 用于在调度步骤中排出每个请求的部分尾部手递手 (hand-off)。
 - 新增 `truncate_computed_blocks()` 方法, 用于在查找结果中对齐到指定 token 端点, 保证前缀截断纯视图且不修改引用计数。
 - 在 `single_type_kv_cache_manager.py` 中为 Mamba align 管理器新增 `_pending_partial_tail_offloads` 列表, 在 CoW 发生时记录 (`req_id`, `group_id`, `block`, `boundary_tokens`)。
 - 新增 `take_pending_partial_tail_offloads()` 排出该列表, 供上层取用。
2. 协调层与配置路由 (`coordinator.py` / `scheduler.py`)
 - `coordinator.py` 新增 `enable_partial_hash_hits` 标识以及 `partial_hash_hits_enabled` 属性, 同步 `hash_block_size` 给所有下游。
 - `scheduler.py` 在 `build_connector_meta` 时嵌入部分尾部 offload 信息, 并在预清空重置 `has_pending_offload` 状态。
3. Worker 端实际卸载 (`worker.py`)
 - 新增 `_maybe_offload_partial_tail()` 方法, 负责将每个请求的部分尾部批量写出到 MooncakeStore。
 - 该方法遍历所有 token database, 从上次保存偏移到边界之间的每个块进行 put, 对 Mamba align 的边界块使用 CoW 目标 ID, 其他组 (如 full attention) 使用请求块表中的 ID。
 - 卸载过程支持压力保护 (`_store_pressure_active`), put 失败时触发压力门限, 阻止后续常规保存直至恢复。

4. 测试配套 - 共 5 个测试文件被新增 / 大幅修改, 覆盖:

- `test_partial_prefix_cache_hits.py`: 验证 offload 正确返回 CoW 目标块、pin 生存周期、请求释放时 offload 丢弃、无 partial tail 时返回空。
- `test_mooncake_store_scheduler.py`: 验证调度器中部分 hash 命中的查找边界、本地命中溢出外部命中的跳过、子块 prompt 查找开关、预清空中 `pending_offload` 重置等。
- `test_mooncake_store_hma_e2e.py`: 端到端测试子块 partial tail offload 读取 CoW 块、覆盖更小组块、事件同步等。
- `test_mooncake_store_worker.py`: 测试 null source block 跳过、压力门限、put 失败激活压力、lookup 使用 hash 对齐。
- `test_mooncake_store_coordinator.py`: 测试协调器细粒度 partial tail 命中、单组缺失时裁剪。

5. 全路径集成: 通过 `create_kv_cache_blocks` 工厂方法、`free` 时释放 pin、`pop_blocks_for_free` 中携带 pin 块, 确保所有资源生命周期正确。

关键文件:

- `vllm/v1/core/kv_cache_manager.py` (模块 缓存管理器; 类别 source; 类型 core-logic; 符号 `truncate_computed_blocks`, `take_partial_tail_offloads`): 核心分发入口: 新增 `take_partial_tail_offloads()` 和 `truncate_computed_blocks()`, 管理全局部分尾部手递手和结果截断。
- `vllm/v1/core/single_type_kv_cache_manager.py` (模块 单类型缓存管理器; 类别 source; 类型 core-logic; 符号 `take_pending_partial_tail_offloads`): Mamba align 管理器: 新增 `_pending_partial_tail_offloads` 列表和 `take_pending_partial_tail_offloads()` 方法, 在 CoW 发生时记录手递手信息。
- `vllm/distributed/kv_transfer/kv_connector/v1/mooncake/store/worker.py` (模块 存储工作者; 类别 source; 类型 dependency-wiring; 符号 `_maybe_offload_partial_tail`): Worker 端实际卸载落盘点: 新增 `_maybe_offload_partial_tail()` 方法, 负责将所有组的部分尾部写到 MooncakeStore。
- `tests/v1/core/prefix_cache/test_partial_prefix_cache_hits.py` (模块 前缀缓存单元测试; 类别 test; 类型 test-coverage; 符号 `test_take_partial_tail_offloads_returns_cow_target`, `test_partial_tail_pin_survives_released_cow_retention`, `test_partial_tail_offload_dropped_when_request_freed_before_drain`, `test_take_partial_tail_offloads_empty_without_partial_tail`): 最核心的单元测试, 覆盖 offload 返回值、CoW 目标确认、pin 生命周期、请求释放后 offload 丢弃、空 offload 边界。
- `vllm/distributed/kv_transfer/kv_connector/v1/mooncake/store/coordinator.py` (模块 存储协调器; 类别 source; 类型 core-logic; 符号 `partial_hash_hits_enabled`): 协调器新增 `enable_partial_hash_hits` 标识和 `partial_hash_hits_enabled` 属性, 同步 `hash_block_size` 给 worker 等组件。
- `vllm/distributed/kv_transfer/kv_connector/v1/mooncake/store/scheduler.py` (模块 存储调度器; 类别 source; 类型 dependency-wiring): 调度器: 在 `build_connector_meta` 中嵌入 `partial_tail_offloads`, 处理预清空中重置 `has_pending_offload`, 调整查找边界对齐

逻辑。

- `vllm/v1/core/sched/scheduler.py` (模块 调度器; 类别 source; 类型 core-logic) : 主线调度器: 在调度步骤中调用 `take_partial_tail_offloads` 获取手递手并将其注入到 connector 元数据中。
- `tests/v1/kv_connector/unit/test_mooncake_store_scheduler.py` (模块 调度器单元测试; 类别 test; 类型 test-coverage; 符号 `_make_bare_scheduler`, `test_partial_hash_hit_block_aligned_local_loads_partial_tail`, `test_partial_hash_hit_no_remote_gain_skips_load`, `test_sub_block_prompt_looks_up_with_fine_grained`) : 调度器单元测试, 覆盖子块 prompt 查找逻辑、部分 hash 命中跳过加载、预清空 `pending_offload` 重置等。
- `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; 符号 `test_sub_block_partial_tail_offload_reads_cow_block`, `_RecordingStore`, `init`, `batch_put_from_multi_buffers`) : 端到端测试, 验证 worker 中 partial tail offload 读取 CoW 块、覆盖更小组块、事件同步等。
- `tests/v1/kv_connector/unit/test_mooncake_store_worker.py` (模块 工作器单元测试; 类别 test; 类型 test-coverage; 符号 `_make_partial_tail_send_thread`, `_make_partial_tail_req`, `test_partial_tail_offload_skips_null_source_blocks`, `test_partial_tail_offload_honors_active_pressure_gate`) : Worker 单元测试, 验证 null source block 跳过、压力门限、put 失败激活压力、lookup 对齐。
- `vllm/distributed/kv_transfer/kv_connector/v1/mooncake/store/data.py` (模块 数据结构; 类别 source; 类型 core-logic; 符号 `prepare_value_for_block`) : 数据层: 修改 `prepare_value_for_block` 以支持 sub-block 粒度的键值打包; 修改 `ChunkedTokenDatabase` 以支持 sub-block 的 `process_tokens`。
- `tests/v1/kv_connector/unit/test_mooncake_store_coordinator.py` (模块 协调器单元测试; 类别 test; 类型 test-coverage; 符号 `_mamba_align`, `test_coordinator_fine_grained_partial_tail_hit`, `test_coordinator_fine_grained_clips_when_one_group_missing_tail`) : 协调器单元测试, 验证 fine-grained partial tail 命中、单组缺失时裁剪。

关键符号: `take_partial_tail_offloads`, `truncate_computed_blocks`, `_maybe_offload_partial_tail`, `take_pending_partial_tail_offloads`, `partial_hash_hits_enabled`, `find_longest_cache_hit`

关键源码片段

`vllm/v1/core/kv_cache_manager.py`

核心分发入口: 新增 `take_partial_tail_offloads()` 和 `truncate_computed_blocks()`, 管理全局部分尾部手递手和结果截断。

```
def take_partial_tail_offloads(self) -> dict[str, list[tuple[int, int, int]]]:
    """Drain producer partial-tail offload hand-offs per request.

    Returns ``{request_id: [(group_id, block_id, boundary_tokens), ...]}``
    for the durable boundary blocks of producers' last-prompt-boundary
```

partial tails. Only mamba "align" groups contribute; empty otherwise.
A KV connector reads the referenced blocks and offloads them so a later request can hit the sub-block prefix.

Each handed-off block lives off the request block table (it is the CoW target), so the caller must keep it alive until the connector has read it. The returned blocks are pinned in ``self.\_partial\_tail\_pins`` and freed when the request is freed (see ``free()``).

```
"""
offloads: dict[str, list[tuple[int, int, int]]] = {}
for manager in self.coordinator.single_type_managers:
    for req_id, group_id, block, boundary in (
        manager.take_pending_partial_tail_offloads()
    ):
        offloads.setdefault(req_id, []).append(
            (group_id, block.block_id, boundary)
        )
        # Pin block until request is freed so the connector can
        # safely read it even after the CoW retention is released.
        self._partial_tail_pins.setdefault(req_id, []).append(block)
return offloads
```

vllm/v1/core/single_type_kv_cache_manager.py

Mamba align 管理器：新增 `_pending_partial_tail_offloads` 列表和 `take_pending_partial_tail_offloads()` 方法，在 CoW 发生时记录手递手信息。

```
def take_pending_partial_tail_offloads(
    self,
) -> list[tuple[str, int, KVCacheBlock, int]]:
    """Drain producer partial-tail hand-offs.
```

Entries are ``(req\_id, group\_id, block, boundary\_tokens)``.

Only mamba "align" populates this. The block lives off the request block table, so the caller must pin it until the connector has read it — nothing else keeps it alive once the CoW retention is released.

```
"""
pending = self._pending_partial_tail_offloads
self._pending_partial_tail_offloads = []
return pending

# In _apply_cow, when a mamba "align" CoW happens and the source was
# a producer partial tail, record the target block:
if self.kv_cache_spec_kind == "mamba_align":
    producer_boundary = self._producer_partial_tail_reqs.pop(request_id, None)
    if producer_boundary is not None:
        self._pending_partial_tail_offloads.append(
            (request_id, self.kv_cache_group_id, cow_target, producer_boundary)
        )
```

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

Worker 端实际卸载落盘点：新增 `_maybe_offload_partial_tail()` 方法，负责将所有组的部分尾部写到 MooncakeStore。

```
def _maybe_offload_partial_tail(self, req_meta: ReqMeta) -> bool:
    """Offload the request's sub-block partial tail (its last prompt hash
    boundary) so a later request can hit the sub-block prefix.

    Covers every block from the normal save's lcm floor to the boundary:
    the normal save floors to ``lcm_block_size``, so a smaller-block
    group's full blocks in that gap are never persisted elsewhere, and
    the consumer's lookup needs every group at every probed boundary.
    Full blocks are keyed by their block-end hash, the partial boundary
    block by the boundary sub-hash; the mamba "align" boundary block is
    the core-provided CoW block. All keys are deduped against the store.

    Returns:
        True when no put is needed or every put succeeds, False otherwise.
    """
    if not self.coord.enable_partial_hash_hits or not req_meta.block_hashes:
        return True
    partial_tail_offloads = req_meta.partial_tail_offloads
    if not partial_tail_offloads:
        return True
    hash_block_size = self.coord.hash_block_size
    boundaries = {boundary for _, _, boundary in partial_tail_offloads}
    if len(boundaries) != 1:
        raise ValueError(
            "Partial-tail offloads for one request must share a boundary"
        )
    boundary = boundaries.pop()
    if boundary == 0:
        return True
    if boundary // hash_block_size - 1 >= len(req_meta.block_hashes):
        return True
    mamba_offloads = {
        group_id: block_id for group_id, block_id, _ in partial_tail_offloads
    }
    # ... ( 遍历所有 token database, 根据块步长分发到对应 rank,
    # 对边界块使用 mamba_offloads 中的 ID, 其他块使用 group_blocks)
```

评论区精华

审批者 ivanium 评论 "Discussed offline and LGTM! Please rebase and update the potential edge case for short decode requests." 表明主要设计已 offline 对齐，但指出短解码请求的边界情况仍需修正。无公开 review 评论，设计决策主要在内部沟通中完成。

- 审批与短解码场景 (correctness): 作者已通过后续改进修复；PR 被批准合并。

风险与影响

- 风险：

1. 核心路径变更：KV 缓存管理器、单类型管理器、调度器均被修改，涉及前缀缓存的核心逻辑，修改若存在边界条件疏漏可能导致缓存状态不一致或死锁。
2. 内存引用竞争：CoW 目标块脱离请求块表后被 pin 住，若 pin 释放时机与 connector 读取之间出现竞态，可能导致 use-after-free。现有测试覆盖了 pin 生存周期，但生产环境高并发下仍需验证。
3. 压力门限退避：卸载 put 失败时阻塞常规保存，可能放大尾部延迟。但设计上只影响同一请求的后续扫描，且压力清除后恢复，风险可控。
4. 兼容性：新增 hash_block_size 和 enable_partial_hash_hits 配置，默认关闭，不影响现有用户。但若同时使用旧版 connector，需确保双方配置对齐。

- 影响：影响范围：

- 用户：启用 MooncakeStore 的混合模型部署可显著提升前缀缓存命中率，减少预填充计算，降低端到端延迟。
- 系统：新增 _partial_tail_pins 字典和多个待处理列表，内存开销极低；卸载路径增加少量 CPU 计算和网络 I/O。
- 团队：该 PR 是系列 [3/N] 中的一环，为后续更精细的缓存策略奠定基础。
- 风险标记：核心路径变更，内存引用竞争，压力门限退避

关联脉络

- PR #49392 [Bugfix] Normalize sparse MLA warmup compression ratios: 同样涉及 DeepSeek/Mamba 模型的前缀缓存压缩比修正，部分逻辑与此 PR 中 partial-tail 的持久化策略相关。
- PR #46116 [Core][KV-transfer] MoRIIO: heterogeneous TP<->DP prefill/decode read routing: 同为 KV transfer 模块的异构路由功能，与本 PR 的 partial-tail offload 同属 KV connector 演进路线。