PR #51243 完整报告

vllm-project/vllm

[KV Offload] Emit self-describing events for partial recurrent blocks

合并时间：2026-08-09 16:28

原文链接：http://prhub.com.cn/vllm-project/vllm/pull/51243

执行摘要

- 一句话：partial 递归块补发自描述 KV offload 事件，Mamba 组保留占位符
- 推荐动作：值得精读 events.py 中 _record_partial_tail 的 hash 对齐计算与 scheduler.py 的接入点，对自描述 KV 事件消费者有直接价值；tiering 混合状态下的元数据回填缺口可作为后续跟进方向。

功能与动机

PR body 明确要求为 full-attention cache groups 发出包含 hash 对齐元数据的自描述 CPU-offload KV 事件，使外部 KV 事件消费者可以索引 offloaded blocks；Mamba/SSM 组继续保留 legacy 占位符负载（token_ids=[]、block_size=0、无 cache-spec 元数据）。此前 partial recurrent tail 只能以占位符呈现，外部消费者无法定位这些块的 hash 与 token 范围，这是对 KV offload 可观测性的补全。

实现拆解

实现按 4 步推进：

1. 事件元数据构造（events.py）：在 OffloadingEventTracker 中新增 record_partial_store、record_partial_lookup 两个入口，共同落到内部实现 _record_partial_tail。核心是依据 boundary_tokens 反推 chunk 起点（(boundary_tokens - 1) // tokens_per_chunk 的减一处理避免边界恰在块边缘时选中下一物理块），再换算 first_hash_idx / last_hash_idx，对 req.block_hashes 与 req.all_token_ids 切片，构造带 parent_block_hash、block_size=tokens_per_hash 与 kv_event_group_spec 的 _OffloadEventMetadata。record_partial_lookup 仅在 _pending_event_metadata 无记录时回填，避免用 lookup 请求的数据覆盖 store 时保留的原始元数据。
2. 调度器接入（scheduler.py）：_lookup 在某个 boundary 上所有组全部命中（无 MISS、无 HIT_PENDING/RETRY）时，遍历复用 boundary_keys 列表调用 record_partial_lookup；_build_partial_tail_store_jobs 在 prepare_store 实际接受的 key 上调用 record_partial_store，保证事件只描述真正落盘的块。同时从 supports_partial_tail 的条件中删除“启用 self-describing 事件则禁用 partial tail”的互斥分支（原先的 not (...) 段），使两者可同时生效。
3. 测试配套：test_events.py 新增 3 个用例——验证事件精确描述 hash 对齐物理块前缀（block_hashes=[hash4..hash6]、parent=hash3、token_ids=[17,29)、block_size=4）、

lookup 不覆盖 store 元数据、sliding-window 组仍走占位符（参数化覆盖 store/lookup 两个入口）；test_scheduler.py 让 partial tail 调度器默认开启 self-describing（KVEventsConfig(enable_kv_cache_events=True, publisher="null")），并断言 full-attention 组事件带 block_size=4、token_ids=[16,28)、3 个 hash，recurrent 组仍为占位符。

4. 文档更新：docs/features/kv_offloading_usage.md 扩展 self_describing_kv_events 说明，补充“partial recurrent tails 会发出从物理块起点到尾部边界的 hash 对齐部分，其余 sliding-window/SSM chunk 保留占位符”。

关键文件：

- vllm/distributed/kv_transfer/kv_connector/v1/offloading/events.py（模块 事件追踪；类别 source；类型 core-logic；符号 record_partial_store, record_partial_lookup, _record_partial_tail）：核心实现文件：新增 record_partial_store、record_partial_lookup、_record_partial_tail 三个方法，完成 partial recurrent tail 的 hash 对齐元数据构造，是自描述事件能力的关键补充。
- tests/v1/kv_connector/unit/offloading_connector/test_events.py（模块 事件测试；类别 test；类型 test-coverage；符号 test_partial_tail_event_describes_hash_aligned_physical_block_prefix, test_partial_tail_lookup_does_not_overwrite_store_metadata, test_partial_tail_sliding_window_event_uses_placeholder）：新增 3 个针对性测试，覆盖 hash 对齐前缀、lookup 不覆盖 store 元数据、sliding-window 占位符，是验证核心逻辑的关键配套。
- vllm/distributed/kv_transfer/kv_connector/v1/offloading/scheduler.py（模块 调度器；类别 source；类型 core-logic；符号 _lookup, _build_partial_tail_store_jobs）：调度器接入点：在 _lookup 命中回填与 _build_partial_tail_store_jobs 落盘路径调用事件记录，并移除 supports_partial_tail 对自描述事件的禁用条件。
- tests/v1/kv_connector/unit/offloading_connector/test_scheduler.py（模块 调度测试；类别 test；类型 test-coverage）：将 partial tail 测试默认配置切换为启用 self-describing 事件，并新增对 full-attention 与 recurrent 组事件内容的断言，验证调度器与 tracker 的集成。
- docs/features/kv_offloading_usage.md（模块 功能文档；类别 docs；类型 documentation）：同步更新 self_describing_kv_events 配置说明，补充 partial recurrent tail 的事件语义，保证文档与实现一致。

关键符号：record_partial_store, record_partial_lookup, _record_partial_tail, _lookup, _build_partial_tail_store_jobs

关键源码片段

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

核心实现文件：新增 record_partial_store、record_partial_lookup、_record_partial_tail 三个方法，完成 partial recurrent tail 的 hash 对齐元数据构造，是自描述事件能力的关键补充。

```
def record_partial_store(
    self,
```

```
    req: Request,
    group_config: "GroupOffloadConfig",
    boundary_tokens: int,
    offload_key: OffloadKey,
) -> None:
    """Snapshot metadata for a newly stored partial recurrent tail."""
    # 与 record_store 保持一致：sliding-window 组不使用自描述事件，
    # 继续走 legacy 占位符负载
    if group_config.sliding_window_size_in_chunks is not None:
        return
    self._record_partial_tail(req, group_config, boundary_tokens, offload_key)


def record_partial_lookup(
    self,
    req: Request,
    group_config: "GroupOffloadConfig",
    boundary_tokens: int,
    offload_key: OffloadKey,
) -> None:
    """Backfill metadata for a partial recurrent tail lookup hit."""
    if group_config.sliding_window_size_in_chunks is not None:
        return
    # 只回填缺失项：store 已记录的元数据优先，避免 lookup 请求覆盖原始数据
    if offload_key not in self._pending_event_metadata:
        self._record_partial_tail(req, group_config, boundary_tokens, offload_key)


def _record_partial_tail(
    self,
    req: Request,
    group_config: "GroupOffloadConfig",
    boundary_tokens: int,
    offload_key: OffloadKey,
) -> None:
    """Build metadata for the valid prefix of one physical cache block.

    partial recurrent tail 结束在 hash 边界上、但物理块尚未写满，
    事件只描述物理块起点到合法边界之间的 hash 与 token，不描述剩余空洞。
    """
    if not self.self_describing_enabled:
        return

    tokens_per_hash = group_config.tokens_per_chunk // group_config.hashes_per_chunk
    # 先减一再取整：当 boundary 恰好落在块边缘时，boundary 自身 token
    # 不应驱动选取下一个物理块，因此用 (boundary_tokens - 1) 定位 chunk 起点
    chunk_start = (
        (boundary_tokens - 1) // group_config.tokens_per_chunk
    ) * group_config.tokens_per_chunk
```

```
    first_hash_idx = chunk_start // tokens_per_hash
    last_hash_idx = boundary_tokens // tokens_per_hash
    # 以下断言依赖调用方保证 boundary 为 hash 对齐边界
    assert chunk_start < boundary_tokens
    assert boundary_tokens % tokens_per_hash == 0
    assert last_hash_idx <= len(req.block_hashes)

    # 与完整 chunk 不同，partial tail 只取物理块起点到合法边界之间的 hash
    maybe_block_hashes = req.block_hashes[first_hash_idx:last_hash_idx]
    block_hashes = tuple(
        block_hash for block_hash in maybe_block_hashes if block_hash is not None
    )
    assert block_hashes and len(block_hashes) == len(maybe_block_hashes)
    # 父 hash 是有效区间前一个 hash，供消费者重建 hash 链
    parent_block_hash = (
        req.block_hashes[first_hash_idx - 1] if first_hash_idx > 0 else None
    )
    assert first_hash_idx == 0 or parent_block_hash is not None

    lora_id = req.lora_request.adapter_id if req.lora_request is not None else None
    lora_name = req.lora_request.name if req.lora_request is not None else None
    self._pending_event_metadata[offload_key] = _OffloadEventMetadata(
        block_hashes=block_hashes,
        parent_block_hash=parent_block_hash,
        token_ids=tuple(req.all_token_ids[chunk_start:boundary_tokens]),
        block_size=tokens_per_hash, # 事件按 hash 粒度描述，块大小即 tokens_per_hash
        lora_id=lora_id,
        lora_name=lora_name,
        extra_keys=None,
        group_idx=group_config.group_idx,
        kv_cache_spec=group_config.kv_event_group_spec,
    )
```

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

调度器接入点：在 _lookup 命中回填与 _build_partial_tail_store_jobs 落盘路径调用事件记录，并移除 supports_partial_tail 对自描述事件的禁用条件。

```
  def _lookup(self, req_status: RequestOffloadState) -> int | None:
      complete_hit = self._lookup_complete_chunks(req_status)
      req_status.partial_tail_boundary = None
      if complete_hit is None or not self.config.supports_partial_tail:
          return complete_hit

      local_tokens = req_status.num_locally_computed_tokens
      complete_boundary = local_tokens + complete_hit
      tokens_per_hash = self.config.tokens_per_hash
      block_end = complete_boundary + self._partial_tail_block_size
      # 只回溯一个物理块跨度内的 hash 边界
      max_boundary = round_down(
```

```
        min(req_status.req.num_prompt_tokens - 1, block_end - 1), tokens_per_hash
    )
    if max_boundary <= complete_boundary:
        return complete_hit

    pending = False
    # 从最远的合法边界向前搜索，直到找到所有组都命中的边界
    for boundary in range(max_boundary, complete_boundary, -tokens_per_hash):
        boundary_pending = False
        boundary_missed = False
        boundary_keys = []
        for group_config in self.config.kv_group_configs:
            key = self._make_boundary_key(
                req_status.req, group_config.group_idx, boundary
            )
            boundary_keys.append(key)
            result = self.manager.lookup(key, req_status.req_context)
            if result is LookupResult.MISS:
                boundary_missed = True
                break
            if result in (LookupResult.HIT_PENDING, LookupResult.RETRY):
                boundary_pending = True

        pending |= boundary_pending
        if not boundary_missed and not boundary_pending:
            # 只有所有组都在该边界命中时才回填自描述元数据，
            # 保证后续 BlockStored/BlockRemoved 事件携带 hash 与 token 信息
            for group_config, key in zip(
                self.config.kv_group_configs, boundary_keys
            ):
                self._events_tracker.record_partial_lookup(
                    req_status.req, group_config, boundary, key
                )
            req_status.partial_tail_boundary = boundary
            return boundary - local_tokens

    if pending and complete_hit == 0:
        return None
    return complete_hit
```

评论区精华

Review 中有三处有价值的交锋：

1. 防护位置选择：作者主动指出 _record_partial_tail 内部没有像 record_store 那样的 sliding-window 检查——"I didn't add the same check as record_store does"；orozery 以 suggestion 形式建议把检查放在 record_partial_store / record_partial_lookup 外层，最终采纳，_record_partial_tail 只保留 self_describing_enabled 守卫。

2. boundary key 复用：orozery 提示 "We already create the boundary key a few lines above. We can save it to a list and re-use here"，避免 _lookup 在命中回填时重复构造 key；作者以 boundary_keys 列表落实。
3. tiering 混合状态边缘情况：Change72 用 Codex 发现——若一组 HIT 而另一组 MISS/RETRY，ready 组的元数据不会被 backfill，其 promotion/removal 事件仍可能带占位符负载；该问题不影响缓存与模型正确性，建议在 lookup 循环内记录每个组 HIT 或作为 follow-up。作者回应 "It feels more natural to only record self-describing metadata when all groups are HIT"，非阻塞保留。

- sliding-window 防护应放在哪一层 (correctness): 在外层两个入口添加 sliding_window_size_in_chunks 检查，内部保留 self_describing_enabled 守卫。
- lookup 命中时复用已构造的 boundary key (performance): 作者用 boundary_keys 列表落实，命名循环内收集、命中后直接遍历。
- tiering 混合 HIT/MISS 时 ready 组元数据不回填 (design): 作者认为只在所有组都 HIT 时记录自描述元数据更自然，作为非阻塞项保留，未在本次修复。

风险与影响

- 风险：主要风险集中在三点：

1. 断言脆性：_record_partial_tail 内置三条 assert（boundary_tokens % tokens_per_hash == 0、last_hash_idx <= len(req.block_hashes)、first_hash_idx == 0 or parent_block_hash is not None），依赖调用方保证 hash 边界对齐。当前 supports_partial_tail 约束了 blocks_per_chunk == 1 与单一块大小，未来若放开这些约束（如 EAGLE/DCP 支持、多块大小混合），可能运行期触发断言。
2. tiering 边缘情况：Change72 指出的混合 HIT/MISS 场景下，ready 组元数据不回填，相关 promotion/removal 事件仍是占位符——不影响缓存与模型正确性，但外部消费者可能暂时看不到 ready 组的自描述事件。
3. 热路径开销：_lookup 是每个请求的关键路径，新增 per-boundary 的 key 收集、zip 遍历与 tracker 写入（dict 写入 + 切片）。均为 CPU 侧小操作，但极端配置下 boundary 循环变长时存在放大效应。 - 影响：对使用 OffloadingConnector + self-describing KV events 的部署，外部事件消费者现在能索引完整注意力组的 partial tail 块（hash 列表、token 范围、parent hash、block_size）；Mamba/SSM 组行为不变，保持占位符，不破坏现有消费者。调度器测试默认配置翻转（启用 self-describing）表明该组合已是第一等支持场景。改动集中在 offloading connector 内部，不影响其他 connector 与核心调度路径。 - 风险标记：核心路径变更，边界条件依赖断言，已知边缘情况遗留

关联脉络

- PR #51161 [Bugfix][KV Offload] Handle chunked local attention in offloading scheduler: 同一 offloading scheduler.py 的修复系列，共同支撑 partial tail 与 chunked attention 场景。
- PR #49328 [KV Offload] Fix failed-load livelock by marking the lookup verdict as a miss: KV offload tiering 可靠性系列，涉及 lookup 判定与元数据状态机，与本 PR 的 lookup 回填路径同域。

- PR #50344 [BugFix] Scope divergent hybrid cache hits to capable connectors: v1 scheduler + kv-connector 前缀缓存命中系列，与本 PR 的混合组命中边界处理相关。