PR #51766 完整报告

vllm-project/vllm

[Bugfix][Core] Preserve Mamba running CoW after external hits

合并时间：2026-08-11 18:15

原文链接：http://prhub.com.cn/vllm-project/vllm/pull/51766

执行摘要

- 一句话：修复外部命中后 Mamba running CoW 语义被误判的问题
- 推荐动作：值得精读，尤其是对 vLLM V1 调度器块表所有权和 CoW 语义感兴趣的同学。1 行核心改动背后是 Core/Worker 状态一致性的深刻问题，配套测试完整复现了外部加载序列。建议关注 ywang96 提出的 last_state_block_idx 在投机解码下的分支差异，以及是否补充 num_speculative_blocks > 0 的回归测试。

功能与动机

PR body 指出 Kimi-K3 实测几何为 external state@24,960 → prefill to partial tail@25,472（无 Mamba 块增长）→ continue to prompt end@25,525，生产环境在受影响几何上观察到 3/12 的逐字重复（verbatim repeats）。根因是 allocate_external_computed_blocks() 填充了请求的 Mamba 块表，但当首个续写停留在同一 Mamba 块内时，allocate_new_blocks() 提前返回而未将请求加入 _allocated_block_reqs；随后 partial-tail CoW 被误分类为首次 prefill CoW，Core 用替换块更新表，Worker 却按 running 请求追加返回块 ID，导致 Core 与 Worker 对活 Mamba 状态列产生分歧。

实现拆解

实现拆解如下：

1. 定位提前返回路径：在 vllm/v1/core/single_type_kv_cache_manager.py 的 allocate_new_blocks() 中，align 模式分支存在 if num_required_blocks <= len(req_blocks) and not has_partial_hit: return [] 的提前返回。该路径原本不登记 _allocated_block_reqs，导致后续 partial-tail CoW 走错分支。
2. 单行修复：在提前返回前加入 self._allocated_block_reqs.add(request_id)，将请求标记为“已分配”（running）。这样后续 partial-tail CoW 会走 running-request 分支：活块保留在请求表中，缓存快照移到 CoW 目标块，Worker 不再追加替换块，Core 与 Worker 的状态列保持一致。
3. 新增回归测试：在 tests/v1/core/prefix_cache/test_partial_prefix_cache_hits.py 中新增 test_external_mamba_hit_same_block_uses_running_cow_on_continue，用较小等价几何模拟外部加载序列：external state@10 → prefill to partial tail@14 → continue to prompt end@15。测试断言：续写时 continuation_blocks.get_block_ids()[1] == []（不返回新 Mamba 块）、活块仍留在原表位、partial-tail 哈希迁移到排队 CoW 拷贝目标块。

4. 配套验证：PR 作者在无源码改动时运行新测试失败（断言 [10] == [] 不满足），应用修复后该套件 23 个测试全部通过；pre-commit 对两个变更文件通过。额外说明 macOS 上因 PyTorch MPS teardown 的无关断言而本地禁用了全局 accelerator cleanup，该改动未包含在 PR 中。

关键文件：

- vllm/v1/core/single_type_kv_cache_manager.py（模块 调度器；类别 source；类型 core-logic；符号 allocate_new_blocks）：核心修复文件。在 allocate_new_blocks() 的 no-new-block 提前返回路径上新增 self._allocated_block_reqs.add(request_id)，将外部命中但无需新块的 Mamba 请求标记为已分配，使后续 partial-tail CoW 走 running 分支，修复 Core 与 Worker 的活块表分歧。
- tests/v1/core/prefix_cache/test_partial_prefix_cache_hits.py（模块 前缀缓存；类别 test；类型 test-coverage；符号 test_external_mamba_hit_same_block_uses_running_cow_on_continue）：新增回归测试 test_external_mamba_hit_same_block_uses_running_cow_on_continue，完整复现外部加载后首个续写停留在同一 Mamba 块的序列，断言不返回新块、活块位置不变、partial-tail 哈希迁移到 CoW 目标。该测试在无修复时失败，有效锁定回归。

关键符号：allocate_new_blocks

关键源码片段

vllm/v1/core/single_type_kv_cache_manager.py

核心修复文件。在 allocate_new_blocks() 的 no-new-block 提前返回路径上新增 self._allocated_block_reqs.add(request_id)，将外部命中但无需新块的 Mamba 请求标记为已分配，使后续 partial-tail CoW 走 running 分支，修复 Core 与 Worker 的活块表分歧。

```
def allocate_new_blocks(
    self, request_id: str, num_tokens: int, num_tokens_main_model: int
) -> list[KVCacheBlock]:
    assert isinstance(self.kv_cache_spec, MambaSpec)
    if self.mamba_cache_mode != "align":
        # 非 align 模式：为投机解码（MTP/EAGLE）多分配 speculative blocks
        if self.num_speculative_blocks > 0:
            num_tokens += self.block_size * self.num_speculative_blocks
        return super().allocate_new_blocks(
            request_id, num_tokens, num_tokens_main_model
        )
    else:
        # align 模式：忽略 lookahead tokens，避免破坏块对齐
        num_tokens = num_tokens_main_model
        req_blocks: list[KVCacheBlock] = self.req_to_blocks[request_id]
        # num_tokens 可能包含后续被拒绝的 draft tokens，因此这是过估
        num_required_blocks = (
            cdiv(num_tokens, self.block_size) + self.num_speculative_blocks
        )
        partial_hit = self._partial_hit_reqs.get(request_id)
```

```
        has_partial_hit = partial_hit is not None
        # 若所需块数不超过已有块数且无 partial hit，无需分配新块。
        # 修复：即使不分配新块，也要把请求标记为 allocated，
        # 这样后续 partial-tail CoW 会走 running 分支，
        # 活块保留在请求表内，缓存快照移到 CoW 目标块，
        # Core 与 Worker 的活列保持一致。
        if num_required_blocks <= len(req_blocks) and not has_partial_hit:
            self._allocated_block_reqs.add(request_id)
            return []
        else:
            prev_block_len = len(req_blocks)
            blocks_allocated = request_id in self._allocated_block_reqs
            # 记录最后状态块：running 时取倒数第 (1+num_speculative_blocks) 块，
            # 首次命中 prefix cache 时取最后一块。
            if blocks_allocated:
                self.last_state_block_idx[request_id] = (
                    prev_block_len - 1 - self.num_speculative_blocks
                )
            elif prev_block_len > 0:
                self.last_state_block_idx[request_id] = prev_block_len - 1
            # ... 后续分配逻辑（null blocks、speculative block 复用等）
```

tests/v1/core/prefix_cache/test_partial_prefix_cache_hits.py

新增回归测试 test_external_mamba_hit_same_block_uses_running_cow_on_continue，完整复现外部加载后首个续写停留在同一 Mamba 块的序列，断言不返回新块、活块位置不变、partial-tail 哈希迁移到 CoW 目标。该测试在无修复时失败，有效锁定回归。

```
def test_external_mamba_hit_same_block_uses_running_cow_on_continue():
    """外部 mid-block 命中后，即使首个续写不需要新 Mamba 块，
    请求也必须成为 running 请求，以保证后续 partial-tail CoW 走正确的分支。"""
    hash_block_size = 2
    mamba_block_size = 4 * hash_block_size
    kv_cache_config = KVCacheConfig(
        num_blocks=32,
        kv_cache_tensors=[],
        kv_cache_groups=[
            KVCacheGroupSpec(
                ["full"],
                FullAttentionSpec(
                    block_size=hash_block_size,
                    num_kv_heads=1,
                    head_size=1,
                    dtype=torch.float32,
                ),
            ),
            KVCacheGroupSpec(
                ["mamba"],
                MambaSpec(
                    block_size=mamba_block_size,
```

```
                shapes=(1, 1),
                dtypes=(torch.float32,),
                mamba_cache_mode="align",
            ),
        ),
    ],
)
manager = make_kv_cache_manager(
    kv_cache_config=kv_cache_config,
    max_model_len=8192,
    enable_caching=True,
    hash_block_size=hash_block_size,
)
# 模拟外部加载：状态 @10，随后 prefill 到 partial tail@14，
# 再续写 1 个 token 到 @15。
request = make_request("0", [0] * 15, hash_block_size, sha256)
loaded_blocks = manager.allocate_slots(
    request, num_new_tokens=0,
    num_external_computed_tokens=10, delay_cache_blocks=True,
)
assert loaded_blocks is not None

request.num_computed_tokens = 10
first_step_blocks = manager.allocate_slots(request, num_new_tokens=4)
assert first_step_blocks is not None

source_block_id = manager.get_blocks("0").get_block_ids()[1][1]
partial_hash = request.block_hashes[14 // hash_block_size - 1]
partial_block = manager.block_pool.get_cached_block(
    partial_hash, kv_cache_group_ids=[1]
)
assert partial_block is not None
assert partial_block[0].block_id == source_block_id

# 续写阶段：停留在同一 Mamba 块，不返回新块，活块留在原表位。
request.num_computed_tokens = 14
continuation_blocks = manager.allocate_slots(request, num_new_tokens=1)
assert continuation_blocks is not None
assert continuation_blocks.get_block_ids()[1] == []
assert manager.get_blocks("0").get_block_ids()[1][1] == source_block_id

# partial-tail 缓存快照应迁移到排队 CoW 拷贝的目标块，
# 而不是覆盖活块本身。
copies, _ = manager.take_kv_cache_block_copies()
cow_copy = next(c for c in copies if c.src_block_id == source_block_id)
assert cow_copy.dst_block_id != source_block_id

moved = manager.block_pool.get_cached_block(
    partial_hash, kv_cache_group_ids=[1]
```

```
    )
    assert moved is not None
    assert moved[0].block_id == cow_copy.dst_block_id
```

评论区精华

核心 review 讨论来自 ywang96（#51763 原修复作者）：

- 肯定方向：ywang96 认为本 PR 方向优于 #51763——#51763 通过抑制 mid-block resume 后的 partial-tail stop 来避开该过渡，但牺牲了一个有效缓存条目；本 PR 保留细粒度 partial-tail 缓存，并修复了所有权簿记，对 Core/Worker 活 Mamba 状态列分歧的解释也比他原来的"切割落在块网格外"更完整。
- 对 `_allocated_block_reqs` 其他读者的质疑：ywang96 提出该一行改动会影响 `last_state_block_idx` 在投机解码下的分支选择（single_type_kv_cache_manager.py:1565-1576）：标记后 `blocks_allocated` 为真时走 `prev_block_len - 1 - num_speculative_blocks`，否则走 `prev_block_len - 1`。两个公式仅在 `num_speculative_blocks == 0` 时一致，因此投机解码场景需要确认语义是否仍然正确。该讨论线程未在 PR 中闭合（review_comments 为空），属于未完全解决的疑虑。
  - 标记 allocated 对 last_state_block_idx 在投机解码下的影响 (design): 该问题被提出后未被进一步讨论或显式解决，PR 已由 ZJY0516 批准合并，但 review_comments 为空，属于遗留的未闭合疑虑。
  - 与 #51763 方案对比：保留 partial-tail 缓存 vs 抑制 stop (design): 结论明确：采纳本 PR 方向，保留 partial-tail 缓存并修复 running CoW 所有权。

风险与影响

- 风险：主要风险集中在 `last_state_block_idx` 的语义变化：
- 投机解码场景：在 `vllm/v1/core/single_type_kv_cache_manager.py` 中，`blocks_allocated` 由 False 变 True 后，`last_state_block_idx` 的公式从 `prev_block_len - 1` 变为 `prev_block_len - 1 - num_speculative_blocks`。当 `num_speculative_blocks > 0` 时，两者不再一致，可能影响 MTP/EAGLE 草稿块的状态保存位置。虽然后续 else 分支的块复用逻辑（reuse previous speculative blocks in this step）依赖 `blocks_allocated`，但新增测试仅覆盖 `num_speculative_blocks == 0` 的几何，未覆盖投机解码组合。
- 外部加载与 partial-tail 缓存交互：改动把外部命中后但未新增块的请求标记为 allocated，可能影响后续 free 或 offload 路径对 `_allocated_block_reqs` 的读取（如 #51358 涉及的 connector handoff 生命周期），PR body 已声明与 #51358 范围不同，但未做交叉验证。
- 回归覆盖面：源码仅 1 行，但行为分支变化影响面主要在 Mamba align 模式 + 外部前缀加载场景；常规模型不受影响。
- 影响：影响范围限定在 vLLM V1 调度器核心的 Mamba align 缓存模式：
- 用户侧：修复了 Kimi-K3 等 Mamba 模型在外部 KV 加载 + partial-tail 缓存场景下的状态列错乱，可消除 3/12 的逐字重复生成；对普通 Transformer 模型无影响。
- 系统侧：Core 与 Worker 的块表所有权语义被修正，后续 CoW 行为与 running 请求一致；保留了 partial-tail 缓存命中，避免 #51763 那样的缓存收益损失。

- 团队侧：该 PR 是 Mamba 外部命中正确性链条中的一环，与 #51763、#51358 形成系列修复；commits 仅 1 个，代码量小，review 已通过。
- 风险标记：投机解码分支未覆盖，1 行核心逻辑变更，外部加载与 CoW 交互复杂，讨论遗留未闭合问题

关联脉络

- PR #51763 Suppress partial-tail stop after mid-block Mamba resume: 同一 Mamba 外部命中 /partial-tail 问题线的替代方案；本 PR 明确与其“materially different”，保留 partial-tail 缓存而非抑制 stop。
- PR #51358 Fix Mamba boundary-state persistence and connector handoff lifetime: 处理 Mamba 边界状态持久化和 connector handoff 生命周期；PR body 声明本 PR 与其范围不同，不标记外部填充请求为 allocated，属于相关但独立的修复。