PR #43516 完整报告

vllm-project/vllm

[KV Connector][Bugfix] MooncakeStore: don't double-apply Eagle prune in load_mask

合并时间：2026-05-26 13:11

原文链接：http://prhub.com.cn/vllm-project/vllm/pull/43516

执行摘要

- 一句话：修复 Eagle 双重剪枝致 MooncakeStore 加载 KV 损坏
- 推荐动作：强烈建议阅读此 PR，尤其关注 coordinator.py 中的 apply_eagle 参数设计：如何通过布尔参数区分 lookup 与 mask 的 Eagle 需求。PR body 的 Walkthrough 部分也非常清晰地描述了 bug 复现步骤和修复逻辑。新增的测试是很好的契约测试范例。

功能与动机

修复启用 Eagle/MTP 时 MooncakeStore 跨实例加载导致的静默 KV 缓存损坏。PR body 描述：The recv-side load_mask was re-applying the last-block pop that client.lookup had already applied, so the trailing block of every cache hit was silently dropped — leaving uninitialized KV in the local pool while the scheduler had already claimed it as cached. 该问题在 MiniMax + MTP/EAGLE-3 交叉实例加载时表现为生成严重损坏（幻觉系统提示、丢失 / 交换用户提示）。

实现拆解

步骤

1. 修改 find_longest_cache_hit 方法：在 coordinator.py 中添加 apply_eagle: bool = True 关键字参数，控制是否应用 Eagle 最后块 pop。该参数透传给 _find_hit_blocks。
2. 修改 _find_hit_blocks 方法：同样添加 apply_eagle 关键字参数，内部使用局部变量 eagle_indices 代替直接引用 self.eagle_attn_group_indices：当 apply_eagle=False 时忽略 Eagle 索引，避免二次 pop。
3. 修改 load_mask 方法：调用 find_longest_cache_hit 时传入 apply_eagle=False，因为 token_len 已是 lookup 返回的 Eagle 剪枝后的命中长度。
4. 新增单元测试：在 test_mooncake_store_coordinator.py 中添加 4 个测试函数，覆盖 Eagle 场景下 lookup 与 load_mask 的往返契约、纯 FullAttn 组、FullAttn+SWA 混合组以及无 Eagle 场景。
5. 文档注释：根据 review 反馈，在关键调用点补充解释性注释，说明为何需要 apply_eagle=False。

关键文件：

- vllm/distributed/kv_transfer/kv_connector/v1/mooncake/store/coordinator.py（模块 协调器；类别 source；类型 core-logic；符号 find_longest_cache_hit, load_mask, _find_hit_blocks）：核心修复文件，添加 apply_eagle 参数控制 Eagle 剪枝，避免 load_mask 重复应用导致 KV 损坏
- tests/v1/kv_connector/unit/test_mooncake_store_coordinator.py（模块 测试；类别 test；类型 test-coverage；符号 test_lookup_with_eagle_pops_last_full_attention_block, test_load_mask_with_eagle_does_not_double_prune_full_attention, test_load_mask_with_eagle_hybrid_full_plus_swa, test_load_mask_without_eagle_unchanged）：新增 4 个回归测试覆盖 Eagle 与 load_mask 交互的关键场景，验证修复正确性

关键符号：find_longest_cache_hit, load_mask, _find_hit_blocks, test_lookup_with_eagle_pops_last_full_attention_block, test_load_mask_with_eagle_does_not_double_prune_full_attention, test_load_mask_with_eagle_hybrid_full_plus_swa, test_load_mask_without_eagle_unchanged

关键源码片段

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

核心修复文件，添加 apply_eagle 参数控制 Eagle 剪枝，避免 load_mask 重复应用导致 KV 损坏

```
# coordinator.py 核心变更片段

def find_longest_cache_hit(
    self,
    block_hashes: list[BlockHash],
    max_length: int,
    cached_block_pool: ExternalCachedBlockPool,
    *,
    apply_eagle: bool = True, # 新增参数：控制是否应用 Eagle 最后块 pop
) -> tuple[tuple[list[bool], ...], int]:
    """
    Returns ``(load_mask_per_group, hit_length)``.
    ``apply_eagle`` 为 True 时，对 Eagle 组应用尾部块 pop（查找调用需要）；
    为 False 时跳过 pop，适用于 load_mask —— 因为 token_len 已经是剪枝后的长度。
    """
    blocks_per_group, hit_length = self._find_hit_blocks(
        block_hashes, max_length, cached_block_pool, apply_eagle=apply_eagle
    )
    masks = tuple(
        [blk is not cached_block_pool.null_block for blk in blocks]
        for blocks in blocks_per_group
    )
    return masks, hit_length
```

```
def load_mask(
    self,
    block_hashes: list[BlockHash],
    token_len: int,
) -> tuple[list[bool], ...]:
    """
    Per-group load masks.
    注意：必须传递 `apply_eagle=False`，因为 `token_len` 已由
    `client.lookup` 剪枝过。如果这里再 pop 一块，会导致掩码比实际
    块数少一，接收线程将静默跳过最后一个块，留下未初始化的 KV。
    """
    masks, _ = self.find_longest_cache_hit(
        block_hashes,
        token_len,
        ExternalCachedBlockPool(),
        apply_eagle=False, # 关键修复：禁止双重 pop
    )
    return masks
```

tests/v1/kv_connector/unit/test_mooncake_store_coordinator.py

新增 4 个回归测试覆盖 Eagle 与 load_mask 交互的关键场景，验证修复正确性

```
# 回归测试：验证 load_mask 不会双重剪枝

def test_load_mask_with_eagle_does_not_double_prune_full_attention():
    """Regression for silent KV corruption with MTP/EAGLE-3.

    接收侧调用 ``load_mask(block_hashes, token_len)`` 时，
    ``token_len`` 已经是 lookup 阶段 Eagle 剪枝后的命中长度。
    之前 load_mask 内部再次 pop 一块，导致掩码少一，
    而 ``process_tokens`` 仍然会产出那块对应的 chunk，
    最终该 chunk 被静默跳过，本地 KV 中对应块未初始化。
    """
    groups = [KVCacheGroupSpec(["L0"], _full(16))]
    coord = _make_coord(groups, hash_block_size=16, use_eagle=True)
    hs = _hashes(4) # 4 个哈希，代表 4 个块（64 tokens）
    cmap = ExternalCachedBlockPool({(0, bytes(h)) for h in hs})

    # 第一步：lookup 触发 Eagle 剪枝，命中 3 块（48 tokens）
    _masks, hit = coord.find_longest_cache_hit(
        hs, max_length=64, cached_block_pool=cmap
    )
    assert hit == 48 # 原始 4 块 - Eagle pop 1 块 = 3 块

    # 第二步：load_mask 必须使用剪枝后的 hit（48），
    # 且不能再 pop。期望得到 3 个 True（对应 3 个块）。
    masks = coord.load_mask(hs, token_len=hit)
    assert masks[0] == [True, True, True]
```

评论区精华

审核者 zhewenl 在审查 coordinator.py 的 load_mask 调用处要求添加注释，解释为何使用 apply_eagle=False（let's add a comment to explain why we need to use apply_eagle=False?）。作者在后续提交中为 find_longest_cache_hit、load_mask 和 _find_hit_blocks 都添加了详细注释，说明 apply_eagle 的作用和双重 pop 的风险。

- 解释 apply_eagle=False 的原因 (documentation): 作者在 find_longest_cache_hit 和 load_mask 中添加了详细注释，说明 token_len 已剪枝和避免双重 pop 的原因。

风险与影响

- 风险：变更主要集中在协调器内部方法签名和调用方式，默认 apply_eagle=True 保持向后兼容。主要风险在于未来有新的 find_longest_cache_hit 或 load_mask 调用者若忘记传递 apply_eagle 参数，可能误用默认值导致类似 bug。但当前所有内部调用路径已显式传入（lookup 不传默认为 True，load_mask 传 False），且新增的测试覆盖了关键路径，回归风险较低。
- 影响：直接影响：使用 MooncakeStore + Eagle/MTP 推测解码的用户将消除随机生成损坏，系统恢复正确的 KV 缓存加载行为。间接影响：其他未使用 Eagle 的功能不受影响（apply_eagle 默认 True 且非 Eagle 路径无 pop 操作）。团队需了解 apply_eagle 参数的存在，以便未来维护。测试框架新增 4 个专用测试，防止此类回归。
- 风险标记：核心路径变更，Eagle 特定逻辑

关联脉络

- PR #43281 [KV Connector] Handle Mooncake finish after preemption: 同一组件（MooncakeStore）的并发处理 bugfix，涉及 scheduler 和 data 模块，与本 PR 在相同代码区域，相互独立但共同提升稳定性。
- PR #42788 [KV Connector] Propagate MooncakeStore load failures: 同一组件的另一个 MooncakeStore 修复，暴露加载失败的 block ID，与本 PR 的加载路径相关。