# PR #49481 完整报告

vllm-project/vllm

[MooncakeStore] Re-derive full external hits on stored boundaries

合并时间：2026-07-23 18:12

原文链接：http://prhub.com.cn/vllm-project/vllm/pull/49481

## 执行摘要

- 一句话：修复外部 KV cache 全命中时调度器选择错误边界导致重复加载失败
- 推荐动作：值得精读。PR 展示了在分布式缓存系统中如何处理“全命中但最后一个 token 需重算”的边界问题：将边界对齐责任下放给 worker，复用外部存在快照避免二次 RPC，设计简洁且测试充分。

## 功能与动机

PR body 指出：'When an external hit covers the full prompt, the final token must still be recomputed. The scheduler previously rounded that hit down arithmetically, which could select an interior fine-grained boundary that no producer persisted and cause repeated failed loads.' 即全命中时调度器算术向下取整可能选中不存在的内部边界，导致负载重复失败。

## 实现拆解

1. Scheduler 层简化：MooncakeStoreScheduler.get_num_new_matched_tokens 中移除原先的算术向下取整逻辑（token_len = request.num_tokens // self._block_size * self._block_size 及其后续处理），直接传递 request.num_tokens 给 client.lookup，将边界对齐责任下放至 worker。
2. Coordinator 层添加对齐工具：新增 align_lookup_length 方法，基于各 cache group 的 lcm_block_size 将长度向下对齐到公共块边界，确保查找边界对所有 attention 规格均合法。
3. Worker 层重新推导全命中：MooncakeStoreWorker.lookup 参数名从 token_len 改为 num_tokens，入口优先通过 coord.align_lookup_length 对齐。主查找逻辑不变，若结果 >= num_tokens（即全命中），则用 align_lookup_length(num_tokens-1) 作为新上限，复用同一个 cached_block_pool（外部存在快照）再次调用 find_longest_cache_hit，从而在不发起新 RPC 的情况下获得一个真实持久化的边界。
4. 协议层适配：LookupKeyClient._lookup 和 lookup 方法参数名同步改为 num_tokens，确保 wire format 传递原始长度。
5. 测试覆盖：新增 3 个 worker 测试（test_lookup_full_hit_reuses_existing_boundary、test_lookup_full_hit_with_eagle_pops_once_not_twice、test_lookup_full_hit_swa_degrades_when_no_stored_boundary_is_usable），验证正常全命中重新派生、Eagle 场景下不额外弹出、SWA 场景下降级行为；更新 scheduler 测试

以匹配新语义（_StubLookupClient 记录请求长度，模拟返回 32 而非 48）。

关键文件：

- vllm/distributed/kv_transfer/kv_connector/v1/mooncake/store/worker.py（模块 worker；类别 source；类型 core-logic；符号 lookup, _lookup）：核心修复所在，lookup 方法增加了全命中重新派生逻辑
- vllm/distributed/kv_transfer/kv_connector/v1/mooncake/store/scheduler.py（模块 调度器；类别 source；类型 core-logic；符号 get_num_new_matched_tokens）：移除旧的算术取整逻辑，将边界对齐责任下放至 worker
- 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 align_lookup_length）：新增 align_lookup_length 方法用于统一块边界对齐
- tests/v1/kv_connector/unit/test_mooncake_store_worker.py（模块 测试；类别 test；类型 test-coverage；符号 test_lookup_full_hit_reuses_existing_boundary, test_lookup_full_hit_with_eagle_pops_once_not_twice, test_lookup_full_hit_swa_degrades_when_no_stored_boundary_is_usable）：新增 3 个测试覆盖全命中重新派生、Eagle 和 SWA 降级场景
- tests/v1/kv_connector/unit/test_mooncake_store_scheduler.py（模块 测试；类别 test；类型 test-coverage）：更新测试以匹配调度器新语义，验证全命中下 worker 返回对齐的 hit_length

关键符号：lookup, _lookup, get_num_new_matched_tokens, align_lookup_length

## 关键源码片段

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

核心修复所在，lookup 方法增加了全命中重新派生逻辑

```
def lookup(self, num_tokens: int, block_hashes: Sequence[BlockHash]) -> int:
    """检查 store 中有多少前缀 token 可用。

    Checks across all rank-specific key namespaces that may be loaded. A
    hit covering all ``num_tokens`` is re-derived below the request end so
    the last token is recomputed for sampling.
    """
    # Step 1: 将请求长度对齐到公共块边界
    token_len = self.coord.align_lookup_length(num_tokens)
    if not block_hashes or token_len <= 0:
        return 0

    # Step 2: 构建候选键（省略，与之前相同）
    # ...

    # Step 3: 使用 batch_is_exist 获取外部存在快照
    # ...

    # Step 4: 组装存在集合，按 group+hash 去重
```

```
        cached_block_pool = ExternalCachedBlockPool(
            self.hash_block_size,
            exists_set,
        )
        # 第一次查找：使用对齐后的完整长度
        _masks, hit_length = self.coord.find_longest_cache_hit(
            block_hashes,
            token_len,
            cached_block_pool,
        )
        # Step 5: 若全命中，则在请求结束下方重新派生一个实际存储的边界
        if hit_length >= num_tokens:
            usable_length = self.coord.align_lookup_length(num_tokens - 1)
            if usable_length <= 0:
                return 0
            # 复用 cached_block_pool（不发起新 RPC）
            _masks, hit_length = self.coord.find_longest_cache_hit(
                block_hashes,
                usable_length,
                cached_block_pool,
            )
        return hit_length
```

## vllm/distributed/kv_transfer/kv_connector/v1/mooncake/store/scheduler.py

移除旧的算术取整逻辑，将边界对齐责任下放至 worker

```
    def get_num_new_matched_tokens(
        self,
        request: Request,
        num_computed_tokens: int,
    ) -> tuple[int | None, bool]:
        """检查外部 KV cache 命中。

        Returns ``(None, False)`` when an async lookup is still in flight,
        signaling the scheduler to retry this request on a later step.
        """
        # 不再对请求长度做算术取整，直接传递原始长度
        if request.num_tokens < self._block_size:
            return 0, False

        num_external_hit_tokens = self.client.lookup(
            request.request_id,
            request.num_tokens, # 直接传递原始长度
            request.block_hashes,
            non_block=self.lookup_async,
        )
        if num_external_hit_tokens is None:
            return None, False

        # 不再有全命中特殊修正（已下放至 worker）
```

```
if num_external_hit_tokens < num_computed_tokens:
    need_to_allocate = 0
else:
    need_to_allocate = num_external_hit_tokens - num_computed_tokens
# ...
```

## 评论区精华

审查中无实质性讨论；ivanium 批准，claude[bot] 自动评论（因来自 fork 跳过审查）。

- 暂无高价值评论线程

## 风险与影响

- 风险：
  1. 回归风险（worker.py lookup 重入路径）：若 align_lookup_length(num_tokens-1) 返回 0 或负值，函数直接返回 0，可能导致本应部分命中的请求退化为 0 命中，但已有边界检查防止空输入。
  2. Eagle 场景（worker.py lookup 第 1537-1545 行）：Eagle 已自动为 drafter 保留最后一块，重新派生本不该触发。测试覆盖了全命中时正确弹出一次（返回 48），但若未来 Eagle 逻辑改变可能导致二次弹出。
  3. SWA 降级（worker.py lookup SWA 路径）：当只有尾部窗口持久化时，重新派生会彻底回退到 0 以避免 livelock，这是预期行为，但会损失缓存优势。
  4. 性能影响：全命中场景多一次 find_longest_cache_hit 调用，但复用 cached_block_pool（仅查询内存集合），开销可忽略。 - 影响：只影响 MooncakeStore KV connector 组件，用户无感知。修复了在特定竞态（调度器算术取整与生产者存储边界不匹配）下的重复加载失败，提升了外部 KV cache 的命中率和可靠性。测试覆盖了新路径，降低回归风险。 - 风险标记：核心路径变更，边界条件复杂，Eagle/SWA 退化场景需谨慎

## 关联脉络

- PR #48425 [BugFix] Handle per-group prefix-hit divergence for hybrid models with KV connector: 同一个 KV connector 模块的 prefix-hit 发散 bugfix，涉及协调器和调度器的交互逻辑
- PR #48399 [Core] Simplify KVBlockZeroer index tensor handling: 同模块重构，简化 KVBlockZeroer 索引处理，减少循环缓冲区预分配