PR #44409 完整报告

vllm-project/vllm

[Bugfix] Two-phase KV allocation for cross-group prefix cache hits (supersedes #33775)

合并时间：2026-06-15 22:40

原文链接：http://prhub.com.cn/vllm-project/vllm/pull/44409

执行摘要

- 一句话：修复跨组前缀缓存命中的物理块重复分配问题
- 推荐动作：值得精读。此 PR 展示了一个关键的数据竞争修复，设计决策（将跨组依赖提升到协调器层、两阶段分离、基于 num_cached_block 的守卫）具有良好的启发意义。对于维护 KV cache、调度器或分布式前缀缓存的开发者，理解此变更能帮助他们避免类似的跨组分配问题。

功能与动机

Hybrid models with local prefix hits + external KV could assign the same physical block twice due to interleaved per-group allocation (issue #33775, downstream assert failure in #43884). The original flow allowed one group's external get_new_blocks to evict another group's not-yet-touched cache-hit blocks, causing duplicate block IDs and ref_cnt corruption. This PR implements a two-phase allocation to prevent this race. (PR body, issue #33775, issue #43884)

实现拆解

1. 拆分单类型管理器方法：在 single_type_kv_cache_manager.py 中将原 allocate_new_computed_blocks 拆分为 add_local_computed_blocks（仅处理本地缓存命中块的 touch 和注册）和 allocate_external_computed_blocks（根据总 token 数分配新块用于外部计算 token）。外部 token 的跳过逻辑从前者移除，外部块分配仅在后者中执行。
2. 协调器两阶段调度：在 kv_cache_coordinator.py 的 allocate_new_computed_blocks 中，新增基于 num_cached_block 的快速路径检查（避免重复处理运行的请求）。然后顺序执行两个循环：先对所有组调用 add_local_computed_blocks，再在 num_external_computed_tokens > 0 时对所有组调用 allocate_external_computed_blocks。确保外部块分配时所有组的本地缓存块已被稳固注册。
3. 测试覆盖：在 test_prefix_caching.py 新增回归测试 test_cache_hit_local_and_external、三组场景 test_cache_hit_local_and_external_three_groups 及抢占重分配测试，以及辅助函数 _assert_no_double_allocation。在 test_single_type_kv_cache_manager.py 中更新现有测试以使用新方法名。

关键文件：

- vllm/v1/core/single_type_kv_cache_manager.py（模块 KV 缓存管理器；类别 source；类型 core-logic；符号 allocate_new_computed_blocks, add_local_computed_blocks, allocate_external_computed_blocks）：核心逻辑变更：将原 allocate_new_computed_blocks 拆分为 add_local_computed_blocks 和 allocate_external_computed_blocks，分别处理本地缓存注册和外部块分配。
- vllm/v1/core/kv_cache_coordinator.py（模块 协调器；类别 source；类型 core-logic；符号 allocate_new_computed_blocks）：协调器层实现两阶段分配顺序控制，确保所有组的本地缓存块先注册后才进行外部分配。
- tests/v1/core/test_prefix_caching.py（模块 前缀缓存测试；类别 test；类型 test-coverage；符号 test_cache_hit_local_and_external, _take_free_blocks, _assert_no_double_allocation, _two_phase_block_size）：新增回归测试，覆盖跨组缓存碰撞、三组场景及抢占重分配，确保补丁正确性。
- tests/v1/core/test_single_type_kv_cache_manager.py（模块 缓存管理器测试；类别 test；类型 test-coverage）：更新现有测试以使用新方法名 add_local_computed_blocks，保持兼容性。

关键符号：add_local_computed_blocks, allocate_external_computed_blocks, kv_cache_coordinator.allocate_new_computed_blocks

关键源码片段

vllm/v1/core/single_type_kv_cache_manager.py

核心逻辑变更：将原 allocate_new_computed_blocks 拆分为 add_local_computed_blocks 和 allocate_external_computed_blocks，分别处理本地缓存注册和外部块分配。

```
# vllm/v1/core/single_type_kv_cache_manager.py

def add_local_computed_blocks(
    self,
    request_id: str,
    new_computed_blocks: Sequence[KVCacheBlock],
    num_local_computed_tokens: int,
    num_external_computed_tokens: int,
) -> None:
    """
    添加本地缓存命中的 blocks 到请求中。
    仅为初次分配调用（运行中请求已在协调器层短路），
    因此请求的 blocks 列表为空。
    步骤：处理滑动窗口跳过的 blocks，touch 缓存块，
    将跳过块填充为 null，然后添加剩余计算块。
    """
    req_blocks = self.req_to_blocks[request_id]
    assert len(req_blocks) == 0
    num_total_computed_tokens = (
        num_local_computed_tokens + num_external_computed_tokens
    )
    num_skipped_tokens = self.get_num_skipped_tokens(num_total_computed_tokens)
```

```
        num_skipped_blocks = num_skipped_tokens // self.block_size
        if num_skipped_blocks > 0:
            new_computed_blocks = new_computed_blocks[num_skipped_blocks:]

        # Touch 块以防止被驱逐（当启用缓存时）
        if self.enable_caching:
            self.block_pool.touch(new_computed_blocks)
        else:
            assert not any(new_computed_blocks)

        # 填充滑动窗口跳过的 null 块
        req_blocks.extend([self._null_block] * num_skipped_blocks)
        req_blocks.extend(new_computed_blocks)
        self.num_cached_block[request_id] = len(req_blocks)


    def allocate_external_computed_blocks(
        self,
        request_id: str,
        num_local_computed_tokens: int,
        num_external_computed_tokens: int,
    ) -> None:
        """
        分配外部（KV 连接器）计算 token 的新块。
        必须在所有组的 add_local_computed_blocks 之后调用，
        以避免此组的 get_new_blocks 逐出其他组的缓存块（#33775）。
        仅需要外部分配且对应 token 未被完全跳过时执行。
        """
        num_total_computed_tokens = (
            num_local_computed_tokens + num_external_computed_tokens
        )
        num_skipped_tokens = self.get_num_skipped_tokens(num_total_computed_tokens)
        # 调整外部 token 数：被跳过的部分不计入分配需求
        remaining_external = min(
            num_total_computed_tokens - num_skipped_tokens,
            num_external_computed_tokens,
        )
        if remaining_external > 0:
            num_alloc = cdiv(num_total_computed_tokens, self.block_size) - len(
                self.req_to_blocks[request_id]
            )
            if num_alloc > 0:
                allocated = self.block_pool.get_new_blocks(num_alloc)
                self.req_to_blocks[request_id].extend(allocated)
```

vllm/v1/core/kv_cache_coordinator.py

协调器层实现两阶段分配顺序控制，确保所有组的本地缓存块先注册后才进行外部分配。

```
    # vllm/v1/core/kv_cache_coordinator.py
```

```
def allocate_new_computed_blocks(
    self,
    request_id: str,
    new_computed_blocks: tuple[Sequence[KVCacheBlock], ...],
    num_local_computed_tokens: int,
    num_external_computed_tokens: int,
) -> None:
    """
    两阶段分配：先注册所有组的本地缓存块，再分配外部块。
    运行中请求（已存在于任一 manager.num_cached_block）
    不会有新的前缀缓存命中，提前返回。
    """
    # 运行中请求的快速路径
    if any(
        request_id in manager.num_cached_block
        for manager in self.single_type_managers
    ):
        assert all(len(blocks) == 0 for blocks in new_computed_blocks)
        return

    # 第一阶段：所有组处理本地缓存命中（touch + 注册）
    for i, manager in enumerate(self.single_type_managers):
        manager.add_local_computed_blocks(
            request_id,
            new_computed_blocks[i],
            num_local_computed_tokens,
            num_external_computed_tokens,
        )

    # 第二阶段：统一分配外部块（仅在需要外部 token 时）
    if num_external_computed_tokens > 0:
        for manager in self.single_type_managers:
            manager.allocate_external_computed_blocks(
                request_id,
                num_local_computed_tokens,
                num_external_computed_tokens,
            )
```

评论区精华

Review 中 ivanium 与作者进行了多轮深入讨论：

- 初始阶段 ivanium 质疑外部分配守卫 num_external_computed_tokens > 0 是否多余，作者解释了在拆分后 add_local_computed_blocks 仍有快速路径，而 allocate_external_computed_blocks 需要此守卫避免重复分配。
- ivanium 建议使用 request_id in manager.num_cached_block 作为新请求的判断依据（而非 len(req_to_blocks) == 0），以与快速路径信号一致，并提升部分逻辑到协调器层。作者采纳并重构，最终内联为 if any(request_id in manager.num_cached_block ...): return，

简化了代码。

- ivanium 指出 add_local_computed_blocks 的注释与协调器层冗余，作者修剪了注释。 最终两位 reviewer（ivanium 和 youkaichao）均批准。
- 外部分配守卫的必要性及判断条件改进 (design): 作者采纳并重构，统一使用 num_cached_block 检测，并将快速路径提升到协调器层，最终内联为一行检查。
- 统一快速路径到协调器层 (design): 作者在第二次提交中实现了此建议，移除了单个 manager 内的快速路径，统一在协调器处理。
- 注释冗余与简化 (style): 作者修剪注释，移除了重复说明，保持文档简洁。

风险与影响

- 风险：
  1. 回归风险：核心分配路径被完全重写（合并为一个新协调器方法），任何未考虑到的边缘情况可能导致双重分配或 ref_cnt 错误。测试覆盖了基本本地 + 外部、三组、抢占重分配等场景，但仍可能遗漏其他复杂交互（如连贯的多步 token 生成）。
  2. 性能影响：两阶段循环引入两次遍历所有组的开销，但操作总数不变（touch + extend + 外部 alloc），因此性能影响可忽略。
  3. 接口兼容性：allocate_new_computed_blocks 方法签名保持不变（仍在协调器），但底层单类型管理器的方法被拆分。任何直接调用 single_type_kv_cache_manager.allocate_new_computed_blocks 的代码（目前仅通过协调器调用）会中断。经审计，所有调用均已更新。
  4. 集成风险：num_cached_block 快速路径依赖该字典的正确状态；如果任何路径未正确更新或初始化，可能导致请求被错误跳过。 - 影响：此修复直接影响所有使用前缀缓存和 KV connector（外部 KV）的混合注意力模型（如 Gemma-4）。解决了 #43884 中断言失败导致的 EngineCore 崩溃问题。对于不使用外部 KV 的场景，行为不变（因为外部阶段被 if num_external_computed_tokens > 0 守卫）。修复有效降低了 KV 调度器的并发复杂性和潜在数据竞争。测试覆盖全面，风险可控。 - 风险标记：核心分配路径变更，跨组依赖，旧接口移除，回归风险

关联脉络

- PR #33775 [KVConnector] Fix data race when we have both local and external cache hit: 本 PR 是 #33775 的继任者，实现了其原始意图但基于 review 反馈重构；#33775 已因冲突和过时被替代。
- PR #43884 [Bug]: EngineCore crash: AssertionError in offloading_connector during update_state_after_alloc: 本 PR 修复了 #43884 报告的根本原因（双重块分配导致断言失败），但不包含 #44329 中的临时缓解措施。
- PR #44329 [Bugfix] Clamp offloading load path to hash-backed GPU blocks: 独立的修复尝试（解决 #43884 下游现象），与本 PR 修复不同；本 PR 不包含 #44329 的修改。