PR #50507 完整报告

vllm-project/vllm

[KV Offloading] Support partial-tail prefix reuse with fine-grained prefix matching

合并时间：2026-08-05 22:31

原文链接：http://prhub.com.cn/vllm-project/vllm/pull/50507

执行摘要

- 一句话：offloading 支持按 prefix 粒度复用部分物理块，提升混合模型缓存命中率
- 推荐动作：值得精读，尤其是 _lookup 的反向扫描简化设计与 _build_partial_tail_store_jobs 的 CoW 手递模式——它们展示了『在既有完整块命中之上追加小段逻辑』的演进思路，比初版独立计划结构简洁得多。review 中 orozery 与 Change72 的讨论也很有学习价值：前者关于如何避免过度设计，后者关于事件系统与缓存新路径的一致性。建议关注后续是否有 KV Events self-describing 支持及限制放宽的 follow-up PR。

功能与动机

Issue #45702 指出：attention-only 模型的小块 KV cache（16/32 token）给了 prefix caching 细粒度匹配，而 hybrid Attention-Mamba 模型的 full-attention block 必须与 Mamba state block 对齐，导致物理块巨大（如 Qwen3.6 为 784 token），prefix-cache 命中粒度退化为块粒度，chat turns、tool-call 边界等不可复用前缀被浪费。PR body 明确：『Even with a smaller prefix_match_unit, native offloading previously could only store and restore complete physical blocks. As a result, reusable prefix tokens already computed near the end of a block were lost.』本 PR 的目标就是让 offloading 保留并恢复这种『落在物理块内部的 partial tail』。

实现拆解

1. 配置层扩展（scheduler.py 的 SchedulerOffloadConfig / GroupOffloadConfig）：新增 tokens_per_hash、supports_partial_tail 字段，并在 GroupOffloadConfig 中新增 requires_cow_source（Mamba align 模式组为 True），把『哪些组的 partial-tail 数据来自 CoW 手递而非 block table』从调度器 isinstance 判断下沉到配置构建。supports_partial_tail 通过一组保守条件计算：blocks_per_chunk == 1、所有组 block size 一致、存在 Mamba align 且 tokens_per_block > tokens_per_hash 的组、未启用 self-describing KV Events、无 EAGLE 组、无滑动窗口组（或 requires_cow_source）、decode_context_parallel_size == 1。不满足条件时功能自动关闭，退回原有完整块行为。
2. Lookup 路径重构（_lookup 拆分为 _lookup_complete_chunks + _lookup）：先按原有逻辑做完整 chunk 正向扫描得到 complete_hit；若启用 partial tail，则在 complete_boundary 之上、最后一个完整物理块内部按 tokens_per_hash 粒度从高到低反向扫描，对每个候选边界调用 _make_boundary_key（hash_idx = boundary //

tokens_per_hash - 1，直接用 request.block_hashes[hash_idx] 构造 OffloadKey），并要求所有 cache group 的边界 key 全部 HIT 才采纳；存在 HIT_PENDING/RETRY 时标记 pending，命中边界写入 req_status.partial_tail_boundary 供后续 load 消费，若完整命中 为 0 且 pending 则等待异步结果。

3. Store 路径扩展（_build_partial_tail_store_jobs）：消费 scheduler_output.partial_tail_offloads（CoW 手递的 (group_idx, block_id, boundary) 列表），校验各 group 边界一致、落在合法范围后，用 GPULoadStoreSpec 构造单 job，block_ids 取自手递的 CoW 源块（attention 组 + recurrent 组），并注册到 _block_id_to_pending_jobs 防止源块被提前释放；_touch 也同步 touch partial tail 边界 key 以维持缓存活性。
4. 配套改进（字段重命名与测试）：TransferJobStatus 中 sliding_window_block_ids/non_sliding_window_block_ids 更名为语义更准确的 fenced_block_ids/deferred_fence_block_ids（因为 partial-tail 源块不一定是滑动窗口块）；测试新增 _make_partial_tail_scheduler/_make_partial_tail_request 辅助函数，以及 3 个用例验证 store 的 CoW 源选择、lookup 的精确边界返回与 load job 的 dst_spec、以及任一 cache group MISS 时 partial tail 不可用的回退逻辑。

关键文件：

- vllm/distributed/kv_transfer/kv_connector/v1/offloading/scheduler.py（模块 卸载调度；类别 source；类型 core-logic；符号 _lookup, _lookup_complete_chunks, _make_boundary_key, _build_partial_tail_store_jobs）：核心调度器实现：partial-tail 的配置判定、lookup 反向扫描、CoW store job 构建全部在此文件完成，是本次变更的主战场。
- tests/v1/kv_connector/unit/offloading_connector/test_scheduler.py（模块 单元测试；类别 test；类型 test-coverage；符号 _make_partial_tail_scheduler, _make_partial_tail_request, test_partial_tail_store_uses_attention_and_recurrent_cow_sources, test_partial_lookup_returns_exact_boundary_and_group_load_keys）：新增 3 个针对 partial-tail 的单元测试，覆盖 store 的 CoW 源选择、lookup 的精确边界与 load spec、以及任一 group MISS 的回退；同时同步字段重命名对旧测试的影响。

关键符号：_lookup, _lookup_complete_chunks, _make_boundary_key, _build_partial_tail_store_jobs, _touch, _make_partial_tail_scheduler, _make_partial_tail_request

关键源码片段

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

核心调度器实现：partial-tail 的配置判定、lookup 反向扫描、CoW store job 构建全部在此文件完成，是本次变更的主战场。

```
def _make_boundary_key(
    self, request: Request, group_idx: int, boundary_tokens: int
) -> OffloadKey:
    # boundary_tokens 是已对齐到 tokens_per_hash 的边界 token 数，
    # 其对应的最后一个 hash 块索引为 boundary_tokens // tokens_per_hash - 1。
    hash_idx = boundary_tokens // self.config.tokens_per_hash - 1
```

```
    return make_offload_key(request.block_hashes[hash_idx], group_idx)


def _lookup(self, req_status: RequestOffloadState) -> int | None:
    # 先按完整 offload chunk 正向扫描，得到完整命中 token 数。
    complete_hit = self._lookup_complete_chunks(req_status)
    # 每次 lookup 重新评估 partial tail，先清掉上次残留的边界。
    req_status.partial_tail_boundary = None
    # 不启用 partial tail 或完整命中失败时，直接返回完整命中结果。
    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
    # partial tail 只能落在“最后一个完整块”内部：
    # 再往上需要缺失 chunk 的 token，正向扫描已确认不可达。
    block_end = complete_boundary + self._partial_tail_block_size
    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
    # 在块内按 tokens_per_hash 粒度从高到低反向扫描，
    # 只有所有 cache group 的边界 key 全部 HIT 才采纳该边界，
    # 保证 Attention KV 与 Mamba recurrent state 都能恢复。
    for boundary in range(max_boundary, complete_boundary, -tokens_per_hash):
        boundary_pending = False
        boundary_missed = False
        for group_config in self.config.kv_group_configs:
            key = self._make_boundary_key(
                req_status.req, group_config.group_idx, boundary
            )
            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:
            # 找到可复用的 partial tail 边界，记录到请求状态，
            # 供后续 update_state_after_alloc 安排 CoW 加载。
            req_status.partial_tail_boundary = boundary
            return boundary - local_tokens
```

```
    # 若有 pending 命中且完整命中为 0，则等待异步结果；
    # 否则退回完整命中结果。
    if pending and complete_hit == 0:
        return None
    return complete_hit
```

tests/v1/kv_connector/unit/offloading_connector/test_scheduler.py

新增 3 个针对 partial-tail 的单元测试，覆盖 store 的 CoW 源选择、lookup 的精确边界与 load spec、以及任一 group MISS 的回退；同时同步字段重命名对旧测试的影响。

```
def test_partial_lookup_returns_exact_boundary_and_group_load_keys():
    scheduler = _make_partial_tail_scheduler()
    request = _make_partial_tail_request(scheduler)
    req_status = scheduler._req_status["req"]
    req_status.num_locally_computed_tokens = 0
    req_status.update_offload_keys()

    # 所有 cache group 的边界 key 都返回 HIT，
    # 期望 _lookup 返回 28 个外部可加载 token（30 - 前 2 个已计算 token）。
    scheduler.manager.lookup.return_value = LookupResult.HIT
    assert scheduler._lookup(req_status) == 28
    assert req_status.partial_tail_boundary == 28

    # 分配目标块后，load job 的 dst_spec 应包含
    # attention 组 2 个块 + recurrent CoW 组 1 个块的来源。
    scheduler.update_state_after_alloc(
        request,
        KVCacheBlocks(
            (
                [KVCacheBlock(31), KVCacheBlock(32)],
                [KVCacheBlock(0, is_null=True), KVCacheBlock(41)],
            )
        ),
        num_external_tokens=28,
    )
    [load_job] = scheduler._current_batch_load_jobs.values()
    dst_spec = load_job.dst_spec
    assert isinstance(dst_spec, GPULoadStoreSpec)
    assert dst_spec.block_ids.tolist() == [31, 32, 41]
    assert dst_spec.group_sizes == [2, 1]
    assert dst_spec.block_indices == [0, 1]
    # 边界已被消费，请求状态复位。
    assert req_status.partial_tail_boundary is None
```

评论区精华

核心交锋有三处。第一，orozery 在首轮 review 提出整体简化意见：『The partial-tail lookup doesn't need a separate code path — after the existing forward scan confirms chunks 0..N-1 are HITs, boundaries above block N are unreachable (they need the

MISS chunk), so just append a short backward scan within block N checking one key per group. Store the hit as a single partial_tail_boundary: int on the request state』，作者据此把初版的 ExternalLoadPlan/GroupLoadPlan/PartialStoreCandidate 多结构设计收敛为单个字段 + 反向扫描。第二，Change72 指出 self-describing KV Events 路径未覆盖：『the new partial-tail store and lookup paths do not call OffloadingEventsTracker.record_store() or record_lookup(), so these events fall back to placeholder payloads with block_size=0, empty token_ids』，作者回应『I've added a gate so partial-tail reuse is disabled when self-describing KV Events are enabled』，以保守方式解决了兼容性。第三，多轮评论推动把 _partial_tail_enabled/_mamba_group_ids 移入 SchedulerOffloadConfig.from_spec 作为 supports_partial_tail，并用 requires_cow_source 泛化 Mamba 识别（Claude 建议：future recurrent specs (RWKV, RetNet, etc.) just flip the flag），同时把防御性 logger.warning + continue 改成 assert。

- partial-tail lookup 设计简化 (design): 作者采纳，最终实现为 _lookup_complete_chunks + 块内反向扫描 + 单字段 partial_tail_boundary，代码复杂度大幅降低。
- KV Events self-describing 兼容性 (correctness): 作者在 supports_partial_tail 计算中加入 not (enable_kv_cache_events and self_describing_kv_events) 门控，self-describing 开启时禁用 partial-tail 复用，事件支持留作 follow-up。
- supports_partial_tail 计算下沉到配置构建 (design): 作者落实，supports_partial_tail 成为 SchedulerOffloadConfig 的字段，调度器只读 self.config.supports_partial_tail。
- requires_cow_source 泛化替代 _mamba_group_ids (design): 作者采纳，新增 requires_cow_source 字段并在配置构建中为 Mamba align 组设置，调度器用 frozenset 聚合。
- 防御逻辑与字段命名 (style): 作者全部落实，测试同步更新字段引用。

风险与影响

- 风险：
  1. 功能受限与静默回退：supports_partial_tail 有大量限制条件（blocks_per_chunk == 1、单一 block size、无 EAGLE、decode_context_parallel_size == 1、非 self-describing KV Events），在这些配置组合下 partial-tail 静默不生效，用户可能误以为启用成功，建议文档化这些约束。
  2. CoW 源块生命周期：_build_partial_tail_store_jobs 依赖调度器手递的源块 ID，若上游（如 store 侧）与 consumer 版本不一致或 hand-off 缺失，assert self._cow_source_groups.issubset(cow_blocks) 会直接抛错——这是有意的 fail-fast，但跨版本混布时需要关注。
  3. KV Events 兼容性：self-describing 事件路径已 gate，但 legacy key-only 事件路径虽兼容，partial-tail 的 block_size=0 占位 payload 问题在 gate 之外仍可能影响后续事件消费者。
  4. 查找开销：块内反向扫描的查询次数为 (block_size - tokens_per_hash) / tokens_per_hash 轮 × 组数，Mamba 大块场景（如 784/8≈98 轮）会放大 manager.lookup 调用量，可能影响调度延迟。

5. 正确性依赖多组同时命中：test_partial_lookup_requires_every_cache_group 已覆盖单一 group MISS 的回退，但真实多节点下 HIT_PENDING/RETRY 交错状态仍需要端到端验证。 - 影响：影响范围集中在 KV OffloadingConnector（v1）的调度器，不改动 cache manager、其他连接器或公共 API。对 hybrid Attention-Mamba 模型（Qwen3.6-27B 等）在 CPU offload + prefix caching 场景收益显著：示例中缓存命中 token 从 784 提升到 896，TTFT 降低约 53%；对纯 attention 模型无行为变化（supports_partial_tail 需要 recurrent 组）。对团队而言，该 PR 为 Issue #45702 提出的『partial cache hits for hybrid models』提供了 offloading 侧的落地实现，并沉淀了 CoW 手递与细粒度边界查找的模式，但当前实现刻意保守（大量限制条件），后续可逐步放宽。 - 风险标记：核心调度路径变更，功能受多个配置门控限制，KV Events self-describing 暂不支持，块内反向扫描放大 lookup 调用量，依赖 CoW 源块 hand-off 正确性

关联脉络

- PR #50358 [Bugfix] Fail fast with a clear error when CPU offload region exceeds available space: 同为 OffloadingConnector / KV Offload CPU 路径的健壮性改进，与本 PR 共享 offloading 调度与 CPU 缓存语义。
- PR #51007 [KV Offload] Support out-of-tree secondary tier managers via module_path: KV offload tiering 工厂扩展，与本次 partial-tail 支持同属 KV Offload 模块能力演进。
- PR #51180 [CI bug] Fix Each KV cache group's real block_size must be divisible by has h_block_size: 涉及 Mamba 多 KV 组与 hash 块大小（hash_block_size）对齐语义，与本 PR 的 tokens_per_hash 与 block_size 约束直接相关。