PR #45261 完整报告

vllm-project/vllm

[Core][KV events] Report prefix-cache-reused blocks in full report mode

合并时间：2026-07-10 13:46

原文链接：http://prhub.com.cn/vllm-project/vllm/pull/45261

执行摘要

- 一句话：新增 full 模式报告 prefix 缓存复用 block 事件
- 推荐动作：这是一个设计干净、讨论充分的特性 PR。推荐后端和基础设施开发者精读，尤其是 _build_block_stored_event 的共享设计和 resolve_block_hashes 的提取过程，展示了良好的代码演进模式。对于使用 KV events 的外部项目（如 Dynamo、llm-d），建议关注此 PR 并测试 full 模式。

功能与动机

原有 KV cache 事件机制仅在 cache_full_blocks() 中新分配的 block 上发射 BlockStored 事件，而复用的 prefix cache block 从未被报告。外部消费者（如路由网关）因此无法感知复用前缀，导致其镜像前缀树随时间退化，前缀匹配深度降为零。此 PR 增加了可选的 kv_cache_report_mode="full" 模式，也为复用 block 发射事件，使外部消费者能观察到完整的请求前缀链。

实现拆解

1. 参数解析：在 vllm/v1/request.py 中，从采样参数的 extra_args 读取 kv_cache_report_mode，默认为 incremental。
2. 缓存命中触发：在 vllm/v1/core/kv_cache_manager.py 的 get_computed_blocks() 方法中，当满足条件（enable_kv_cache_events=True，kv_cache_report_mode="full"，且 num_new_computed_tokens > 0）时，遍历各组调用 block_pool.emit_cached_block_events()。
3. 事件构建与发射：在 vllm/v1/core/block_pool.py 中，新增 emit_cached_block_events() 方法，通过共享的 _build_block_stored_event() 构造 BlockStored 事件，不修改 block 状态。同时将原有 cache_full_blocks() 中的事件构造也改用该方法，确保一致性。
4. 辅助函数抽取：响应 review 反馈，将 block hash 解析逻辑提取为独立函数 resolve_block_hashes() 并移入 vllm/v1/core/kv_cache_utils.py，防止重复代码。
5. 测试覆盖：在 tests/v1/core/test_prefix_caching.py 中新增三个测试用例，分别验证正常发射、禁用时不发射、零缓存不发射。

关键文件：

- vllm/v1/core/block_pool.py（模块 block 池；类别 source；类型 core-logic；符号 _build_block_stored_event, emit_cached_block_events）：核心变更文件。新增

emit_cached_block_events 方法负责为复用 block 发射事件，并重构 cache_full_blocks使用共享的 _build_block_stored_event，保证事件形状一致。

- vllm/v1/core/kv_cache_manager.py（模块 缓存管理；类别 source；类型 core-logic）：触发点：在 get_computed_blocks 中根据 kv_cache_report_mode 调用事件发射，连接请求参数与 block pool。
- vllm/v1/core/kv_cache_utils.py（模块 缓存工具；类别 source；类型 core-logic；符号 resolve_block_hashes）：提取的共享辅助函数 resolve_block_hashes，封装了 block hash 在不同 block size 下的解析逻辑，消除重复代码。
- vllm/v1/request.py（模块 请求；类别 source；类型 core-logic）：新增 kv_cache_report_mode 属性解析，使得用户可通过 vllm_xargs 控制报告模式。
- tests/v1/core/test_prefix_caching.py（模块 测试；类别 test；类型 test-coverage；符号 test_emit_cached_block_events, test_emit_cached_block_events_disabled, test_emit_cached_block_events_zero_cached）：三个测试用例覆盖了 emit_cached_block_events 的主要场景，确保功能正确。

关键符号：emit_cached_block_events, _build_block_stored_event, resolve_block_hashes

关键源码片段

vllm/v1/core/block_pool.py

核心变更文件。新增 emit_cached_block_events 方法负责为复用 block 发射事件，并重构 cache_full_blocks 使用共享的 _build_block_stored_event，保证事件形状一致。

```
# vllm/v1/core/block_pool.py ( 新增共享事件构建和发射方法 )

def _build_block_stored_event(
    self,
    request: Request,
    block_hashes: list[ExternalBlockHash] | None,
    parent_block_hash: ExternalBlockHash | None,
    start_token_idx: int,
    end_token_idx: int,
    block_size: int,
    kv_cache_group_id: int,
    extra_keys_list: list[tuple[Any, ...] | None],
) -> BlockStored:
    """
    构建 BlockStored 事件，由 cache_full_blocks 和 emit_cached_block_events 共享。
    两者发射的事件形状保持一致，确保下游消费者可以统一处理。
    """
    return BlockStored(
        block_hashes=block_hashes,
        parent_block_hash=parent_block_hash,
        token_ids=request.all_token_ids[start_token_idx:end_token_idx],
        block_size=block_size,
        lora_id=request.lora_request.adapter_id if request.lora_request else None,
        medium=MEDIUM_GPU,
```

```
            lora_name=request.lora_request.name if request.lora_request else None,
            extra_keys=extra_keys_list if extra_keys_list else None,
            group_idx=kv_cache_group_id,
        )


    def emit_cached_block_events(
        self,
        request: Request,
        num_cached_blocks: int,
        block_size: int,
        kv_cache_group_id: int,
    ) -> None:
        """
        为从 prefix cache 复用的 block 生成 BlockStored 事件。
        与 cache_full_blocks 不同，此方法不修改 block 状态——block 已经缓存。
        它仅仅生成事件，使得外部消费者（如 gateway）可以了解复用的前缀信息。
        """
        if not self.enable_kv_cache_events:
            return
        if num_cached_blocks == 0:
            return

        # 使用共享的 resolve_block_hashes 解析 block 哈希
        block_hashes = resolve_block_hashes(request, self.hash_block_size, block_size)
        assert len(block_hashes) >= num_cached_blocks

        # 复用前缀总是从 block 0 开始连续，因此 parent_block_hash 为 None
        parent_block_hash = None
        new_hashes: list[ExternalBlockHash] | None = [
            maybe_convert_block_hash(block_hashes[i]) for i in range(num_cached_blocks)
        ]

        self.kv_event_queue.append(
            self._build_block_stored_event(
                request,
                block_hashes=new_hashes,
                parent_block_hash=parent_block_hash,
                start_token_idx=0,
                end_token_idx=num_cached_blocks * block_size,
                block_size=block_size,
                kv_cache_group_id=kv_cache_group_id,
                extra_keys_list=[],
            )
        )
```

vllm/v1/core/kv_cache_manager.py

触发点：在 get_computed_blocks 中根据 kv_cache_report_mode 调用事件发射，连接请求参数与 block pool。

```
# vllm/v1/core/kv_cache_manager.py ( 在 get_computed_blocks 中新增触发逻辑 )

# 原有查找缓存命中逻辑之后 ...
computed_blocks, num_new_computed_tokens = (
    self.coordinator.find_longest_cache_hit(
        request.block_hashes, max_cache_hit_length
    )
)

# 当 kv_cache_report_mode 为 "full" 时，为复用的 prefix cache block 发射事件
if (
    num_new_computed_tokens > 0
    and self.enable_kv_cache_events
    and getattr(request, "kv_cache_report_mode", "incremental") == "full"
):
    for group_idx, group_blocks in enumerate(computed_blocks):
        num_blocks = len(group_blocks)
        if num_blocks > 0:
            group = self.kv_cache_config.kv_cache_groups[group_idx]
            block_size = group.kv_cache_spec.block_size
            self.block_pool.emit_cached_block_events(
                request,
                num_blocks,
                block_size,
                group_idx,
            )
```

评论区精华

讨论亮点

1. block_size 一致性检查（MengqingCao）：在 emit_cached_block_events 中最初缺少对 block_size 和 hash_block_size 关系的校验。作者随后提取了 resolve_block_hashes 共享函数，其中包含了 assert block_size % hash_block_size == 0，同时应用于两个调用方。
2. parent_block_hash 是否可省略（MengqingCao）：认为若 parent_block_hash 始终为 None 则不需要报告。作者解释保留 None 以维持 BlockStored 事件的统一形状，下游消费者依赖该字段重建前缀链。
3. 辅助函数提取至 kv_cache_utils（ivanium）：建议将 _resolve_block_hashes（当时是 BlockPool 的方法）提升为模块级函数，便于更通用的使用。作者照做，得到 resolve_block_hashes()。
4. 复用前缀起始注释优化（MengqingCao）：指出 emit_cached_block_events 的注释可能暗示可以从非 0 开始，需要澄清。作者更新注释明确 prefix-cache hits 总是形成从 block 0 开始的连续前缀。

- block_size 与 hash_block_size 的一致性校验 (correctness): 作者提取了 resolve_block_hashes 共享函数，在函数内部通过 assert 确保两者兼容，并应用到 cache_full_blocks 和 emit_cached_block_events。
- parent_block_hash 字段保留 None 的必要性 (design): 作者解释保留 None 以维持 BlockStored 事件形状统一，下游消费者依赖该字段重建前缀链，因此不能省略。
- 将 resolve_block_hashes 提取为模块级函数 (design): 作者照做，创建了独立的 resolve_block_hashes 函数，供两个方法调用。
- 复用前缀起始注释优化 (documentation): 作者更新注释，明确 prefix-cache hits 总是形成从 block 0 开始的连续前缀，因此 parent_block_hash 为 None。

风险与影响

- 风险：该 PR 新增了一条可选的事件发射路径，但默认行为完全不变（incremental 模式）。启用 full 模式时，每个请求在首次调度时可能额外发射一些 BlockStored 事件，数量等于复用 block 的组数（通常 1-2 组）。事件数量可控，不会显著增加事件队列压力。主要风险在于外部消费者若不正确处理幂等性（IDem Potence），可能观察到重复事件导致状态错误，但 PR body 已明确消费者应按幂等性理解事件。此外，由于事件发射位于 get_computed_blocks 路径，不会阻塞主调度循环。代码中使用了 assert，可通过 -O 消除潜在影响。
- 影响：用户层面：提供了一种新的可选参数 kv_cache_report_mode，默认无影响。KV events 消费者可通过设置该参数获得完整前缀链，提升路由准确度。

系统层面：增加少量事件开销，仅在启用且请求命中缓存时产生。不改变现有数据路径。

团队层面：该 PR 建立了一种扩展 KV events 事件类型的模式（共享 _build_block_stored_event 和 resolve_block_hashes），后续功能可复用。

- 风险标记：新增事件发射路径，full 模式事件数量增加，依赖外部消费者幂等处理

关联脉络

- PR #47923 [kv_offload] Emit tier-owned BlockStored events from FS/OBJ secondary tiers: 同样扩展 KV events 机制，增加新的 BlockStored 事件来源