PR #41366 完整报告

vllm-project/vllm

[KV Offload] Pass ReqContext to touch(), complete_load(), and complete_store()

合并时间：2026-05-10 20:09

原文链接：http://prhub.com.cn/vllm-project/vllm/pull/41366

执行摘要

- 一句话：统一 OffloadingManager 完整接口，为 touch/complete_load/complete_store 添加 ReqContext 参数
- 推荐动作：值得精读，尤其关注抽象基类接口演进方式：先为核心方法引入上下文参数，再逐步扩展到全部回调方法，保持渐进式兼容。此模式适合大型分布式系统的接口标准化。

功能与动机

在 PR #39185 中，OffloadingManager 的三个 " 发起 " 方法（lookup, prepare_load, prepare_store）已经接受 req_context: ReqContext，但三个 " 完成 / 维护 " 方法（touch, complete_load, complete_store）尚未支持。此 PR 填补该空白，使接口完整一致，并允许未来的 manager 实现在完成回调中基于 per-request context 执行操作（例如调整策略、记录统计等）。

实现拆解

1. 更新抽象基类 base.py：修改 OffloadingManager 的 touch、complete_load、complete_store 方法签名，为每个方法新增 req_context: ReqContext 参数，并更新 docstring。这是所有实现必须遵循的契约变化。
2. 更新具体实现 cpu/manager.py 和 reuse_manager.py：
   - CPUOffloadingManager.touch：新增 req_context 参数，但方法体仍只调用 self._policy.touch(keys)，忽略上下文（当前策略不需要）。
   - CPUOffloadingManager.complete_load：新增参数，方法体仍只操作 ref_cnt。
   - CPUOffloadingManager.complete_store：新增参数，方法体处理成功 / 失败逻辑；内部仍忽略 req_context。
   - FilteredReuseManager 的对应委托方法同步更新签名，将所有参数直接传递给 _backing，实现透明转发。
3. 更新分布式调度器 offloading/scheduler.py：
   - _touch 方法调用 self.manager.touch 时，新增第二个参数 req_status.req_context。
   - update_connector_output 方法中调用 complete_store 和 complete_load 时，新增 req_status.req_context 参数。
   - 同时将 req_status 的获取提前，确保在调用前可获得 req_context。

4. 测试配套改动 test_manager.py：所有调用 touch、complete_load、complete_store 的地方均添加 _EMPTY_REQ_CTX 作为 req_context 参数，保持测试与接口一致。

关键文件：

- vllm/v1/kv_offload/base.py（模块 KV 卸载；类别 source；类型 core-logic；符号 touch, complete_load, complete_store）：定义了 OffloadingManager 抽象基类，是接口契约的核心。touch, complete_load, complete_store 方法均新增了 req_context 参数，所有实现必须遵循。
- vllm/v1/kv_offload/cpu/manager.py（模块 KV 卸载；类别 source；类型 core-logic；符号 touch, complete_load, complete_store）：CPUOffloadingManager 是主要的 offloading 实现，此处方法签名同步更新，虽然内部当前仍忽略 req_context 参数。
- vllm/distributed/kv_transfer/kv_connector/v1/offloading/scheduler.py（模块 分布式调度；类别 source；类型 core-logic）：调度器是主要调用方，在 _touch 和 update_connector_output 中传递实际的 req_context，体现了接口的完整使用。
- vllm/v1/kv_offload/reuse_manager.py（模块 KV 卸载；类别 source；类型 core-logic；符号 touch, complete_load, complete_store）：FilteredReuseManager 作为代理管理器，同步更新了委托方法，保持对底层 manager 的透明转发。
- tests/v1/kv_offload/cpu/test_manager.py（模块 测试；类别 test；类型 test-coverage）：测试用例同步更新，传递 _EMPTY_REQ_CTX 占位参数，确保 CI 通过。

关键符号：OffloadingManager.touch, OffloadingManager.complete_load, OffloadingManager.complete_store, CPUOffloadingManager.touch, CPUOffloadingManager.complete_load, CPUOffloadingManager.complete_store, FilteredReuseManager.touch, FilteredReuseManager.complete_load, FilteredReuseManager.complete_store

关键源码片段

vllm/v1/kv_offload/cpu/manager.py

CPUOffloadingManager 是主要的 offloading 实现，此处方法签名同步更新，虽然内部当前仍忽略 req_context 参数。

```
# vllm/v1/kv_offload/cpu/manager.py ( 变更后片段 )

class CPUOffloadingManager(OffloadingManager):
    def touch(self, keys: Collection[OffloadKey], req_context: ReqContext) -> None:
        # 当前忽略 req_context，直接调用底层策略的 touch
        self._policy.touch(keys)

    def complete_load(
        self, keys: Collection[OffloadKey], req_context: ReqContext
    ) -> None:
        # 方法体仅操作引用计数，不依赖请求上下文
        for key in keys:
            block = self._policy.get(key)
            assert block is not None, f"Block {key!r} not found"
```

```
            assert block.ref_cnt > 0, f"Block {key!r} ref_cnt is already 0"
            block.ref_cnt -= 1

    def complete_store(
        self,
        keys: Collection[OffloadKey],
        req_context: ReqContext,
        success: bool = True,
    ) -> None:
        # 存储完成逻辑：成功则标记 ready，失败则移除
        stored_keys: list[OffloadKey] = []
        if success:
            for key in keys:
                block = self._policy.get(key)
                if block is not None and not block.is_ready:
                    block.ref_cnt = 0
                    stored_keys.append(key)
        else:
            for key in keys:
                block = self._policy.get(key)
                if block is not None and not block.is_ready:
                    self._policy.remove(key)
                    self._free_block(block)
        # ... events 处理略
```

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

调度器是主要调用方，在 _touch 和 update_connector_output 中传递实际的 req_context，体现了接口的完整使用。

```
# vllm/distributed/.../offloading/scheduler.py（变更后片段）

def _touch(self, req_status: RequestOffloadState):
    for group_config, group_state in zip(
        self.config.kv_group_configs, req_status.group_states
    ):
        if group_config.sliding_window_size_in_blocks is None:
            # 新增第二参数：req_context
            self.manager.touch(group_state.offload_keys, req_status.req_context)
        else:
            blocks_to_skip = max(
                0,
                group_state.num_hit_blocks
                - group_config.sliding_window_size_in_blocks,
            )
            self.manager.touch(
                group_state.offload_keys[blocks_to_skip:],
                req_status.req_context,
            )
```

```
def update_connector_output(self, connector_output: KVConnectorOutput):
    # ... 在获取 req_status 后传递 req_context
    req_status = self._req_status[job_status.req_id]
    if job_status.is_store:
        self.manager.complete_store(job_status.keys, req_status.req_context)
    else:
        self.manager.complete_load(job_status.keys, req_status.req_context)
```

评论区精华

无实质技术讨论。orozery 快速批准，gemini-code-assist[bot] 自动回复确认变更。仅有的评论来自 mergify[bot] 关于 pre-commit 失败的信息，但已解决。

- PR 审核总结 (other): 变更被接受，合并到 main。

风险与影响

- 风险：低风险。
- 当前所有实现均忽略新参数 req_context，因此不会改变既有的运行行为。
- 接口变更属于向后兼容的扩展（添加默认值？但没有默认值，所有调用点必须更新），但本 PR 统一更新了所有调用点，编译 / 运行时不会出现遗漏。
- 若未来有第三方自定义 OffloadingManager 未同步更新，则会导致类型错误。需同步发布变更。
- 影响：### 对用户 无功能性影响，KV offload 行为完全保持不变。

对系统

接口统一为所有方法提供 per-request context 能力，后续可以开发更智能的 offloading 策略（如按请求优先级调整 eviction）。

对团队

降低了维护多个方法签名不一致的成本；后续添加新 manager 实现时必须遵循此签名规范。

- 风险标记：所有实现均忽略新参数，调用点已全部更新，低风险

关联脉络

- PR #39185 [KV Offload] Introduce ReqContext and pass it to lookup, prepare_load, prepare_store: 该 PR 是 #39185 的后续，完成了接口的完全统一。