PR #47679 完整报告

vllm-project/vllm

[KV Offload] Split tiering_lookup_delay into sync/async histograms

合并时间：2026-07-17 01:03

原文链接：http://prhub.com.cn/vllm-project/vllm/pull/47679

执行摘要

- 一句话：拆分层级查找延迟为同步 / 异步直方图
- 推荐动作：值得精读，尤其是同步 / 异步延迟拆分设计、观察时机的决策（allocation vs finish）如何与调度器生命周期对齐。团队其他成员可参考相同模式在其他模块应用。

功能与动机

为了更精确地区分层级 KV 卸载中查找操作的同步阻塞开销和跨步长 deferred 等待时间，便于运维人员定位性能瓶颈。PR body 明确指出拆分后与连接器级同步 / 异步延迟指标（LOOKUP_SYNC_DELAY / LOOKUP_ASYNC_DELAY）对齐。

实现拆解

1. 在 tiering/base.py 中新增 TieringOffloadingMetrics 类，定义 LOOKUP_SYNC_DELAY 和 LOOKUP_ASYNC_DELAY 两个指标名。
2. 在 tiering/spec.py 的 build_metric_definitions 中注册这两个直方图，设置各自的 bucket 边界和文档。
3. 在 tiering/manager.py 中：
   - 在 RequestState 添加 sync_lookup_delay 和 secondary_lookup_start_time 字段；
   - 在 __init__中添加 _stats 缓冲区；
   - 修改 lookup()：记录开始时间，在命中二级层并触发 promotion 时累积同步延迟，若首次 deferred 则设置异步开始时间；
   - 新增 _accumulate_lookup_sync_delay, _maybe_observe_lookup_sync_delay, _maybe_observe_lookup_async_delay 方法；
   - 在 on_schedule_end 中对新增请求（context.new_req_ids）观察同步和异步延迟，对可能 finalize 的逻辑也观察异步延迟。
4. 测试文件扩展 _simulate_on_schedule_end 以支持传入 new_req_ids，新增三个测试用例覆盖同步延迟在分配时上报、异步延迟在 promotion 后上报、以及请求结束时上报异步延迟。

关键文件：

- vllm/v1/kv_offload/tiering/manager.py（模块 层级管理器；类别 source；类型 core-logic；符号 _accumulate_lookup_sync_delay, _maybe_observe_lookup_sync_delay, _maybe_observe_lookup_async_delay, lookup）：核心实现文件；修改 lookup() 和

on_schedule_end，新增延迟累积和观察方法

- vllm/v1/kv_offload/tiering/base.py（模块 层级基类；类别 source；类型 core-logic；符号 TieringOffloadingMetrics）：新增 TieringOffloadingMetrics 类定义指标名
- vllm/v1/kv_offload/tiering/spec.py（模块 层级配置；类别 source；类型 dependency-wiring；符号 build_metric_definitions）：注册两个新的直方图指标到度量定义
- tests/v1/kv_offload/tiering/test_tiering_offloading.py（模块 层级测试；类别 test；类型 test-coverage；符号 test_lookup_reports_sync_delay_for_resolved_lookups, test_lookup_reports_async_delay_across_promotion, test_lookup_reports_async_delay_on_request_finish, _simulate_on_schedule_end）：新增三个延迟测试用例

关键符号：lookup, on_schedule_end, _accumulate_lookup_sync_delay, _maybe_observe_lookup_sync_delay, _maybe_observe_lookup_async_delay, build_metric_definitions

关键源码片段

vllm/v1/kv_offload/tiering/manager.py

核心实现文件；修改 lookup() 和 on_schedule_end，新增延迟累积和观察方法

```
# 在 RequestState 中新增的延迟字段
@dataclass(slots=True)
class RequestState:
    # ... 原有字段
    sync_lookup_delay: float = 0.0 # 累积的同步查找延迟，等待观察时清零
    # time.monotonic() 时间戳，记录本请求首次 deferred 二次查找的时间；
    # None 表示没有待处理的异步延迟 .
    secondary_lookup_start_time: float | None = None


class TieringOffloadingManager:
    # 在 lookup() 中累积同步延迟并可能开启异步计时
    def lookup(self, key, req_context, exclude_tier=None):
        # ... 首先尝试主层
        req_state = self._req_state.get(req_context.req_id)
        # ... 若主层 miss 则遍历二级层
        lookup_start = time.monotonic()
        any_retry = False
        for tier in self.secondary_tiers:
            if tier is exclude_tier:
                continue
            result = tier.lookup(key, req_context)
            if result is LookupResult.HIT:
                promoted = self._initiate_promotion(tier, key, req_context)
                self._accumulate_lookup_sync_delay(req_state, lookup_start)
                # 若此次为首次 deferred，记录异步开始时间
```

```
            if (req_state is not None and promoted
                    and req_state.secondary_lookup_start_time is None):
                req_state.secondary_lookup_start_time = lookup_start
            return LookupResult.RETRY if promoted else LookupResult.MISS
        if result is LookupResult.RETRY:
            any_retry = True
            self._accumulate_lookup_sync_delay(req_state, lookup_start)
    if any_retry:
        if (req_state is not None
                and req_state.secondary_lookup_start_time is None):
            req_state.secondary_lookup_start_time = lookup_start
        return LookupResult.RETRY
    return LookupResult.MISS

def _accumulate_lookup_sync_delay(
    self, req_state: RequestState | None, start_time: float
) -> None:
    """累积单次二级层查找耗时到请求状态中."""
    if req_state is not None:
        req_state.sync_lookup_delay += time.monotonic() - start_time

def _maybe_observe_lookup_sync_delay(self, req_state: RequestState) -> None:
    """将已累积的同步延迟上报到统计直方图并清零."""
    delay = req_state.sync_lookup_delay
    if delay == 0:
        return
    req_state.sync_lookup_delay = 0.0
    self._stats.observe_histogram(
        TieringOffloadingMetrics.LOOKUP_SYNC_DELAY, delay
    )

def _maybe_observe_lookup_async_delay(self, req_state: RequestState) -> None:
    """若请求有异步开始时间，则计算延迟并上报，然后清除开始时间."""
    start = req_state.secondary_lookup_start_time
    if start is None:
        return
    req_state.secondary_lookup_start_time = None
    delay = time.monotonic() - start
    self._stats.observe_histogram(
        TieringOffloadingMetrics.LOOKUP_ASYNC_DELAY, delay
    )
```

评论区精华

审核人 orozery 提出三点关键修改：

- 同步延迟也应像异步一样在 on_schedule_end 观察而非在 lookup 直接观察（减少计算点）；
- 函数应重命名为 _maybe_observe_lookup_sync_delay 以体现条件性；

- 在 on_schedule_end 中只迭代 context.new_req_ids 而非全部 _req_state 以避免过早上报。所有建议均被采纳。
- 同步延迟观察时机应在 on_schedule_end (design): 同意并修改为在 on_schedule_end 中针对 new_req_ids 调用 _maybe_observe_lookup_sync_delay。
- 函数重命名为 _maybe_observe_lookup_sync_delay (style): 采纳新名称。
- 直方图文档字符串调整 (documentation): 接受建议，更新了字符串。

风险与影响

- 风险：主要风险在于同步延迟累积逻辑在请求状态中，若 RequestState 生命周期管理有误可能导致延迟累积不重置或丢失；on_schedule_end 中观察时机若未覆盖所有情况可能漏报异步延迟（例如请求提前结束）。但已通过三个新增测试场景覆盖正常分配、promotion 后、请求结束时三种路径。
- 影响：用户层面无行为变更；运维人员获得两个新的 Prometheus 直方图指标，可用于分析层级卸载延迟；系统性能略增（每个 lookup 多做一次 time.monotonic() 和加法），影响可忽略。
- 风险标记：核心路径变更，指标累积逻辑可能遗漏

关联脉络

- PR #45958 Introduce connector-level sync/async lookup delay metrics: 本 PR 构建于 #45958 之上，镜像其同步 / 异步拆分模式到 TieringOffloadingManager 层级。