PR #49204 完整报告

vllm-project/vllm

[Core] Fix internal LB load-balancing

合并时间：2026-07-27 22:00

原文链接：http://prhub.com.cn/vllm-project/vllm/pull/49204

执行摘要

- 一句话：修复内部 DP 负载均衡统计发布遗漏与评分优化
- 推荐动作：值得精读，特别是 Python 与 Rust 端 LB 策略的权衡、KV 缓存压力感知评分的实现细节。对于理解 vLLM DP 架构和负载均衡设计有参考价值。

功能与动机

PR #30739 添加了 DP 支持，但协调器中的引擎队列统计信息未实际发布到客户端，导致负载均衡失效。Issue #48808 报告了 H20 GPU 上长上下文工作负载下 DP 请求分布不平衡。本 PR 修复此遗漏，并进一步优化评分逻辑以提升均衡效果。

实现拆解

1. 修复统计发布遗漏（vllm/v1/engine/core.py）：在 EngineCoreProc._run_busy_loop 的输入处理和 GPU 步骤前后各调用一次 _maybe_publish_request_counts；新增该方法，仅当统计变化时构造 SchedulerStats（含 kv_cache_usage）通过 output_queue 发布。
2. 扩展 EngineState 支持 KV 缓存使用率（vllm/v1/engine/coordinator.py）：EngineState.request_counts 从 [waiting, running] 变为 [waiting, running, kv_cache_usage]；DPCoordinatorProc 更新统计处理逻辑，仅在 enable_wave_coordination 时同步步骤边界。
3. 增加 Scheduler 接口（vllm/v1/core/sched/interface.py, scheduler.py）：新增抽象方法 get_kv_cache_usage 返回 float，默认实现返回 0.0，具体 Scheduler 实现返回实际使用比例。
4. 优化 Python 端客户端 LB 评分（vllm/v1/engine/core_client.py）：DPLBAsyncMPClient 初始化 lb_engines 为三元组；新增 engine_inflight 精确计数；get_core_engine_for_request 以 max(client_count * inflight, waiting + running) 为基础，当等待请求且 KV 缓存使用率 > 0.5 时施加额外惩罚。
5. 简化 Rust 端评分（rust/engine-core-client/src/client/state.rs）：移除 routing_score 中的额外等待权重，仅返回 inflight 与 stats.running+stats.waiting 的最大值。
6. 新增测试覆盖（tests/v1/engine/test_engine_core_client.py）：重构 _make_dplb_client 统一客户端创建；新增四个测试用例验证轮转、背压、KV 压力惩罚和请求释放场景。

关键文件：

- vllm/v1/engine/core.py（模块 负载均衡核心；类别 source；类型 core-logic；符号 _maybe_publish_request_counts）：修复统计发布遗漏的核心，新增 _maybe_publish_request_counts 方法并在 run_busy_loop 中调用
- vllm/v1/engine/core_client.py（模块 客户端调度；类别 source；类型 core-logic；符号 get_core_engine_for_request）：客户端 LB 评分优化，引入 engine_inflight 和 KV 缓存压力惩罚
- vllm/v1/engine/coordinator.py（模块 协调器；类别 source；类型 core-logic；符号 _get_engine_counts）：协调器扩展 EngineState 支持 KV 缓存使用率，调整统计同步逻辑
- tests/v1/engine/test_engine_core_client.py（模块 测试；类别 test；类型 test-coverage；符号 test_dplb_burst_round_robins_despite_snapshot_rebinds, test_dplb_snapshot_backpressure_overrides_inflight, test_dplb_kv_pressure_amplifies_waiting_penalty, test_dplb_finished_requests_release_inflight）：新增四个测试用例覆盖轮转、背压、KV 压力惩罚和请求释放场景
- rust/src/engine-core-client/src/client/state.rs（模块 Rust 客户端；类别 source；类型 core-logic；符号 routing_score_keeps_extra_waiting_penalty, routing_score_counts_waiting_without_extra_penalty）：Rust 端评分简化，移除额外等待权重

关键符号：_maybe_publish_request_counts, get_core_engine_for_request, _get_engine_counts, get_kv_cache_usage, routing_score, _make_dplb_client

关键源码片段

vllm/v1/engine/core.py

修复统计发布遗漏的核心，新增 _maybe_publish_request_counts 方法并在 run_busy_loop 中调用

```
# vllm/v1/engine/core.py (head)

def _maybe_publish_request_counts(self):
    # 仅当内部 LB 启用时才发布统计
    if not self.publish_dp_lb_stats:
        return

    # 获取当前请求计数，若变化则发布 SchedulerStats
    counts = self.scheduler.get_request_counts()
    if counts != self.last_counts:
        self.last_counts = counts
        # SchedulerStats 现在包含 3 个字段：
        # (num_waiting_reqs, num_running_reqs, kv_cache_usage)
        stats = SchedulerStats(
            *counts,
            kv_cache_usage=self.scheduler.get_kv_cache_usage()
        )
        # 使用 -1 标识来自该引擎的统计消息
```

```
            self.output_queue.put_nowait(
                (-1, EngineCoreOutputs(scheduler_stats=stats))
            )

    # 在 run_busy_loop 中的调用位置：
    # while self._handle_shutdown():
    # self._process_input_queue()
    # self._maybe_publish_request_counts() # 步骤前发布
    # self._process_engine_step()
    # self._maybe_publish_request_counts() # 步骤后发布
```

vllm/v1/engine/core_client.py

客户端 LB 评分优化，引入 engine_inflight 和 KV 缓存压力惩罚

```
    # vllm/v1/engine/core_client.py (head) - DPLBAsyncMPClient.get_core_engine_for_request
    核心评分循环

    # lb_engines 现在为 [waiting, running, kv_cache_usage] 三元组
    current_counts = self.lb_engines
    num_engines = len(current_counts)
    min_score: float = sys.maxsize
    eng_index = 0
    for i in range(num_engines):
        # 从 client_index 偏移开始，帮助空引擎平衡
        idx = (self.eng_start_index + i) % num_engines
        waiting, running, kv_cache_usage = current_counts[idx]

        # 精确的进行中请求数，不会被 coordinator snapshot 重置
        inflight = self.engine_inflight[self.core_engines[idx]]

        # 基础评分：取 coordinator 统计与精确 inflight 的最大值
        # inflight 乘以 client_count 以与其他客户端等比例估计
        score: float = max(self.client_count * inflight, waiting + running)

        if waiting:
            # 当 KV 缓存使用率 > 50% 时，等待请求施加额外惩罚
            # 惩罚从 0（使用率 <=50%）线性增加到 3x waiting（使用率 =100%）
            score += waiting * 6.0 * max(0.0, kv_cache_usage - 0.5)

        if score < min_score:
            min_score = score
            eng_index = idx
```

评论区精华

BugenZhao 在 Review 中指出 Python 端新增的额外评分逻辑在 Rust 端未实现。njhill 回应 Rust 端基于精确 inflight 请求计数的评分已足够，在不同工作负载测试中表现与 Python 端改进后的逻辑一致，因此无需同步 Rust 变更。该讨论揭示了 Python 与 Rust 客户端在 LB 策略上的一致性要求，以及各自环境下权衡的实现差异。

- Python 端额外评分逻辑是否应在 Rust 端同步 (design): njhill 表示 Rust 端基于精确 inflight 的评分已足够，测试表明无需额外等待权重，因此不修改 Rust。

风险与影响

- 风险：核心路径变更（DP 负载均衡）涉及多个模块的协调，但作者进行了充分的实验和测试验证。主要风险： 1) coordinator.py 中条件判断逻辑调整可能影响锁步 DP 同步； 2) 新评分逻辑可能改变负载分布，但测试确认改善； 3) 新增统计发布调用可能引入微小性能开销，但通过仅变化时发布控制。整体风险可控。
  - 影响：对使用内部 DP 负载均衡（无外部 LB）的用户有显著正面影响，解决请求不平衡问题，提高资源利用率。对已有外部 LB 或非 DP 配置的用户无影响。影响范围限于 v1 引擎的 DP 部署形态。
  - 风险标记：核心路径变更，跨模块协调变动，Python/Rust 评分差异

关联脉络

- PR #30739 Add support for efficient DP without EP: 本 PR 修复了 30739 中遗漏的统计发布问题，是其后续改进。
- PR #48808 DP request distribution becomes imbalanced under long-context workload: 关联 Issue，本 PR 修复的 bug 报告。