

PR #55650 完整报告

vllm-project/vllm

[Bugfix] Make KV cache and MFU log lines backend-neutral

合并时间: 2026-09-15 02:00

原文链接: <http://prhub.com.cn/vllm-project/vllm/pull/55650>

执行摘要

- 一句话: KV 缓存与 MFU 日志标签按后端中性化, CPU/XPU 可读
- 推荐动作: 值得快速阅读, 作为“小范围可观测性整改”的范例: 重点学习作者对“单位 vs 设备标签”的语义区分, 以及主动把公开指标改名排除在外的范围控制意识。不需要精读, 逻辑直白, 风险极低。

功能与动机

PR body 明确指出: Three log lines hardcode "GPU" regardless of the active backend, which is misleading on the CPU backend (and on XPU/NPU), 对应 issue #55614 的第 1-3 项; 作者还说明这是 #42479 未覆盖的遗漏点。核心诉求是让非 GPU 后端的可观测性日志反映真实设备, 而不是让用户在 CPU 上看到“GPU KV cache size”这类错误信息。

实现拆解

变更入口是三个日志调用点的文案替换, 全部复用 `vllm_config.device_config.device_type` (配置构造时已解析, 无需新增依赖)。

1. `vllm/v1/core/kv_cache_utils.py`: 启动日志设备标签化 `update_kv_cache_capacity()` 中把硬编码的 "GPU KV cache size: ..." 改为 "%s KV cache size: ...", 新增 `device_type = vllm_config.device_config.device_type`, 并用三元表达式 "GPU" if `device_type == "cuda"` else `device_type.upper()` 生成标签。这样 CUDA/ROCm (均上报 `cuda`) 保持原样, CPU/XPU/NPU 显示各自大写设备名。
2. `vllm/v1/metrics/loggers.py`: 周期与迭代日志复用派生标签 `LoggingStatLogger.__init__` 中一次性计算 `self.kv_cache_device`, 避免在每个日志点重复推导; 随后 `_log_iteration_details()` 的逐迭代日志和 `log()` 的汇总日志都把 "GPU KV cache usage" 替换为 "%s KV cache usage", 并传入该字段。把派生逻辑收敛到构造器, 保证两处日志标签一致。
3. `vllm/v1/metrics/perf.py`: MFU 单位中性化 `PerfMetricsLogging.log()` 把 "MFU: %.1f TF/s/GPU %.1f GB/s/GPU" 改为 "TF/s/device ... GB/s/device"。作者论证 MFU 计算除以 `tp_size/pp_size`, 是 per-rank 速率, `/device` 是单位而非平台标签, 因此不套用设备名派生逻辑。
4. 配套与范围控制 无新增测试文件, 作者在 x86 CPU 后端本地构建并对比了 revert/apply 前后的日志输出作为手工验证。Issue #55614 第 4 项 (重命名 Prometheus 指标

vllm:estimated_flops_per_gpu_total 等) 被明确排除, 避免破坏现有 scraper。

关键文件:

- vllm/v1/core/kv_cache_utils.py (模块 KV 缓存; 类别 source; 类型 core-logic; 符号 update_kv_cache_capacity) : 启动时 KV 缓存容量日志 update_kv_cache_capacity() 是本次改动的入口之一, 通过 device_config.device_type 三元表达式派生设备标签, CUDA/ROCm 保持原文案。
- vllm/v1/metrics/loggers.py (模块 统计日志; 类别 source; 类型 core-logic; 符号 LoggingStatLogger.init, LoggingStatLogger._log_iteration_details, LoggingStatLogger.log) : LoggingStatLogger 是周期与迭代 KV 缓存占用日志的宿主, 本次在构造器一次性派生 kv_cache_device 并复用到两处日志, 避免重复计算。
- vllm/v1/metrics/perf.py (模块 性能指标; 类别 source; 类型 core-logic; 符号 PerfMetricsLogging.log) : MFU 日志的单位由 /GPU 改为 /device, 体现 per-rank 速率的语义, 是本次对“单位 vs 设备标签”判断的核心落点。

关键符号: update_kv_cache_capacity, LoggingStatLogger.init, LoggingStatLogger._log_iteration_details, LoggingStatLogger.log, PerfMetricsLogging.log

关键源码片段

vllm/v1/core/kv_cache_utils.py

启动时 KV 缓存容量日志 update_kv_cache_capacity() 是本次改动的入口之一, 通过 device_config.device_type 三元表达式派生设备标签, CUDA/ROCm 保持原文案。

```
def update_kv_cache_capacity(
    vllm_config: VllmConfig, kv_cache_config: KVCacheConfig
) -> None:
    """记录并输出解析后的 KV cache 容量。"""
    # 计算可容纳 token 数与最大并发度, 并保存到 cache_config 供调度器使用
    num_tokens, max_concurrency = get_kv_cache_capacity(vllm_config, kv_cache_config)
    vllm_config.cache_config.kv_cache_size_tokens = num_tokens
    vllm_config.cache_config.kv_cache_max_concurrency = max_concurrency
    max_model_len = vllm_config.model_config.max_model_len

    # 设备标签从 device_config 派生: CUDA/ROCm 均上报 "cuda", 保持 "GPU" 原文案;
    # CPU/XPU/NPU 等后端则输出各自大写设备名 (cpu -> CPU), 避免日志误导
    device_type = vllm_config.device_config.device_type
    device_label = "GPU" if device_type == "cuda" else device_type.upper()

    logger.info_once(
        "%s KV cache size: %s tokens, "
        "Maximum concurrency for %s tokens per request: %.2fx",
        device_label,
        f"{num_tokens:}",
        f"{max_model_len:}",
        max_concurrency,
```

)

vllm/v1/metrics/loggers.py

`LoggingStatLogger` 是周期与迭代 KV 缓存占用日志的宿主，本次在构造器一次性派生 `kv_cache_device` 并复用到两处日志，避免重复计算。

```
class LoggingStatLogger(StatLoggerBase):
    def __init__(self, vllm_config: VllmConfig, engine_index: int = 0):
        self.engine_index = engine_index
        self.vllm_config = vllm_config

        # 构造时一次性派生设备标签，供迭代日志与周期汇总日志复用；
        # ROCm 同样上报 device_type == "cuda", 因此三元表达式保持 "GPU" 原文案，
        # 非 CUDA 后端（CPU/XPU/NPU）输出各自大写设备名
        device_type = vllm_config.device_config.device_type
        self.kv_cache_device = "GPU" if device_type == "cuda" else device_type.upper()
        self._reset(time.monotonic())
        self.last_scheduler_stats = SchedulerStats()
        # ... 其余指标对象初始化保持不变 ...

    def _log_iteration_details(self, scheduler_stats, engine_idx):
        # 逐迭代日志：把硬编码 "GPU" 换成 self.kv_cache_device
        logger.info(
            "%sIteration(%d): %d context requests, ... "
            "iteration elapsed time: %.2f ms%s, "
            "%s KV cache usage: %.1f%%s",
            self._log_prefix_for_engine(engine_idx),
            details.iteration_index,
            details.num_ctx_requests,
            # ...
            " (dummy)" if details.is_dummy else "",
            self.kv_cache_device,
            scheduler_stats.kv_cache_usage * 100,
            encoder_msg,
        )

    def log(self, log_fn=logger.info, log_prefix: str = ""):
        # 周期汇总日志：同样复用构造时算好的标签，保证两处输出一致
        log_parts.extend(
            ["%s KV cache usage: %.1f%%", "Prefix cache hit rate: %.1f%%"]
        )
        log_args.extend(
            [
                self.kv_cache_device,
                self.last_scheduler_stats.kv_cache_usage * 100,
                self.prefix_caching_metrics.hit_rate * 100,
            ]
        )
```

vllm/v1/metrics/perf.py

MFU 日志的单位由 `/GPU` 改为 `/device`，体现 per-rank 速率的语义，是本次对“单位 vs 设备标签”判断的核心落点。

```
def log(self, log_fn=logger.info, log_prefix: str = "") -> None:
    if not (
        self.total_num_flops_per_gpu
        or self.total_read_bytes_per_gpu
        or self.total_write_bytes_per_gpu
    ):
        return

    now = time.monotonic()
    delta_time = now - self.last_log_time

    # MFU 计算按 tp_size/pp_size 分摊后统计，属于 per-rank 速率；
    # 因此这里改的是“单位”而非“设备标签”，用中性的 /device
    # 而不是平台派生名（如 CPU/XPU），避免又造出新的误导
    if delta_time <= 0.0:
        avg_tflops_per_gpu = 0.0
        avg_gbps_per_gpu = 0.0
    else:
        avg_tflops_per_gpu = self.total_num_flops_per_gpu / delta_time / 1e12
        avg_gbps_per_gpu = (
            (self.total_read_bytes_per_gpu + self.total_write_bytes_per_gpu)
            / delta_time
            / 1e9
        )

    log_fn(
        "%sMFU: %.1f TF/s/device %.1f GB/s/device",
        log_prefix,
        avg_tflops_per_gpu,
        avg_gbps_per_gpu,
    )
```

评论区精华

PR 没有实质性的 reviewer 评论，核心权衡都写在 PR 描述里：

- coderabbitai 自动审查：No actionable comments were generated in the recent review.，未提出需要跟进的问题。
- 范围决策（作者自述）：MFU 行是 per-rank 速率，属于“单位”而非“设备标签”，因此用 `/device` 而不是平台派生名；同时刻意不重命名 Prometheus 公开指标，因为“renaming them breaks existing scrapers and needs a maintainer decision on a deprecation/alias path”，可后续单独跟进。
- 合入过程：维护者 markmc 触发 `/ci run` 后直接 APPROVED，只回复 `Thank you`，说明改动风险被认定为极低。

- MFU 行单位选择 /device 而非平台名 (design): 采用 /device 作为 MFU 单位; CUDA/ROCm 用户看到 TF/s/device 而非 TF/s/GPU, 属于有意的语义修正。
- 刻意排除 Prometheus 指标重命名 (design): 本期不处理, 作者表示可后续单独跟进; 属于明确的 API 兼容性边界决策。
- CUDA/ROCm 输出不变性保证 (correctness): CUDA 与 ROCm 日志文案保持原样, 无回归风险。

风险与影响

- 风险:
 - 缺少测试覆盖: 纯日志文案改动没有配套单测, `device_type.upper()` 的派生逻辑和两处日志格式未受断言保护; 若未来 `device_type` 语义变化 (例如新增非 `cuda` 的 GPU 别名), 可能悄悄回归。
 - 日志解析兼容风险: GPU KV cache、TF/s/GPU、GB/s/GPU 等字符串被部分运维脚本 `grep` 或解析, 本次改为 CPU ...、/device 后这类脚本会失效; 虽然日志不是稳定 API, 但对 CPU/XPU/NPU 用户属可见变化。
 - `device_type.upper()` 的显示假设: 逻辑假设 `device_type` 是简单标识符 (如 `cpu` → `CPU`), 若未来出现复合小写值, 输出可能不符合预期; 当前 ROCm 仍走 `cuda` 分支, 不受影响。
 - 影响面有限: 不涉及数据面、调度或性能逻辑, CUDA 路径按构造保持原文案, 回归风险低。
- 影响:
 - 用户侧: CPU/XPU/NPU 后端的日志可读性显著提升, 不再出现“GPU 缓存”误导; CUDA/ROCm 用户完全无感知。
 - 系统侧: 仅日志输出层变更, 无性能、内存或行为变化。
 - 团队侧: 为 issue #55614 后续第 4 项 (Prometheus 指标改名) 划清了边界, 明确了哪些是安全可改的日志文案、哪些是需走弃用流程的公开契约。
 - 风险标记: 缺少测试覆盖, 日志解析兼容风险, 可观测性文案变更

关联脉络

- PR #55624 [V1][Metrics] Support Sliding Window Attention (SWA) and hybrid layers in MFU/MBU estimation: 同改 `vllm/v1/metrics/perf.py`, 围绕 MFU 计算与日志指标做增强; 本 PR 将 MFU 单位中性化, 两者同属 MFU 可观测性改进线。
- PR #56791 [Bugfix] Trim stale consequence claims from unannotated-eagle warning: 同改 `vllm/v1/core/kv_cache_utils.py`, 属于同一“日志信息正确性”整改方向, 均涉及 `kv_cache_utils` 中日志输出。
- PR #50894 [Bugfix] Scale KV page size for hidden states extraction with TP: 同改 `vllm/v1/core/kv_cache_utils.py` 的 KV 缓存容量相关逻辑区域, 属于该文件的持续演进脉络。