

PR #52755 完整报告

vllm-project/vllm

[Rust Frontend] Record Mooncake/NIXL KV-connector metrics

合并时间: 2026-09-04 14:21

原文链接: <http://prhub.com.cn/vllm-project/vllm/pull/52755>

执行摘要

- 一句话: 为 Rust 前端添加 Mooncake/NIXL 连接器指标记录, 修复指标丢失问题。
- 推荐动作: 本 PR 修复了 Rust 前端的关键监控缺失, 实现思路清晰, 测试覆盖充分。建议关注其跨语言数据结构设计 (如 KvConnectorStats 枚举和 to_dict 扁平化), 这对维护 Rust/Python 双前端协议一致性有参考价值。由于涉及序列化格式变更, 升级时需确保监控系统适配新的指标名称。

功能与动机

Rust 前端 (VLLM_USE_RUST_FRONTEND=1) 在解码来自 Python EngineCore 的 kv_connector_stats 时, 无法识别 Mooncake 和 NIXL 连接器的具体结构, 导致这些指标被静默丢弃。这影响了用户对分布式 KV 缓存传输性能的监控能力。本 PR 旨在补齐这一功能缺口, 使 Rust 前端能够记录并暴露与 Python 前端一致的 Mooncake 和 NIXL 指标。

实现拆解

- 定义 Rust 端指标类型与协议: 在 rust/src/metrics/src/scheduler.rs 中添加 Mooncake 和 NIXL 连接器的直方图、计数器定义及标签结构 (MooncakeOperationLabels 等), 并注册到 Prometheus 注册表。在 rust/src/engine-core-client/src/protocol/stats.rs 中新增 kv_connector 模块, 定义 NixlStats、MooncakeStats、MultiConnectorStats 等数据结构, 用于从 JSON 反序列化 kv_connector_stats。
- 实现指标记录逻辑: 修改 rust/src/engine-core-client/src/metrics.rs, 在 SchedulerStatsHandles 中新增对应指标句柄, 并在 record_scheduler_stats_with_handles 函数中解析 kv_connector_stats, 调用 record_nixl_stats、record_mooncake_stats、record_multi_connector_stats 函数将数据写入指标。
- Python 端适配: 在 vllm/distributed/kv_transfer/kv_connector/v1/metrics.py 中为 KVConnectorStats 基类添加 to_dict 方法; 修改 vllm/distributed/kv_transfer/kv_connector/v1/multi_connector.py 中 MultiKVConnectorStats.to_dict 和 MultiKVConnectorPromMetrics.observe, 以输出扁平化的字典格式, 并调整 build_kv_connector_stats 去除多余的 "data" 层级, 确保与 Rust 端的反序列化结构匹配。
- 测试与兼容性验证: 更新 tests/v1/kv_connector/unit/test_multi_connector.py 中的测试用例, 验证 to_dict 输出和扁平化观察逻辑; 在 rust/src/engine-core-client/src/tests/client.rs 和 rust/src/engine-core-client/src/tests/python_compat.py 中增加 Python 与 Rust 编码兼容性测试, 确保 NIXL、Mooncake 和多连接器统计能被正确序列化和反序列化。

关键文件：

- rust/src/engine-core-client/src/metrics.rs（模块 指标记录；类别 source；类型 core-logic；符号 record_multi_connector_stats, record_mooncake_stats, record_nixl_stats）：核心记录逻辑，新增了 Mooncake 和 NIXL 指标的记录函数，将统计信息写入 Prometheus 指标。
- rust/src/metrics/src/scheduler.rs（模块 指标定义；类别 source；类型 core-logic；符号 mooncake_operation_time_histogram, nixl_xfer_time_histogram, nixl_post_time_histogram, nixl_bytes_transferred_histogram）：定义 Mooncake 和 NIXL 连接器的指标结构、标签和桶，注册到 Prometheus 注册表。
- rust/src/engine-core-client/src/protocol/stats.rs（模块 协议定义；类别 source；类型 dependency-wiring；符号 NixlStats, MooncakeOperation, MooncakeStatus, MooncakeRecord）：定义从 Python 反序列化的数据结构，将 kv_connector_stats 从不透明的 OpaqueValue 变为强类型枚举。
- vllm/distributed/kv_transfer/kv_connector/v1/multi_connector.py（模块 多连接器统计；类别 source；类型 core-logic；符号 to_dict, observe, build_kv_connector_stats）：Python 端适配，修改 MultiKVConnectorStats.to_dict 和 MultiKVConnectorPromMetrics.observe 以输出扁平结构，并调整 build_kv_connector_stats 去除 data 层级。
- tests/v1/kv_connector/unit/test_multi_connector.py（模块 连接器测试；类别 test；类型 test-coverage；符号 test_build_kv_connector_stats_with_multiple_connectors, test_prom_metrics_passes_flat_data_to_children）：更新测试用例以验证 to_dict 输出和扁平化观察逻辑，确保与 Rust 端兼容。

关键符号：record_multi_connector_stats, record_mooncake_stats, record_nixl_stats, to_dict, build_kv_connector_stats

关键源码片段

rust/src/engine-core-client/src/metrics.rs

核心记录逻辑，新增了 Mooncake 和 NIXL 指标的记录函数，将统计信息写入 Prometheus 指标。

```
// 从 kv_connector_stats 中解析并记录 NIXL 统计信息到 Prometheus 直方图和计数器。
fn record_nixl_stats(handles: &SchedulerStatsHandles, stats: &NixlStats) {
    // 记录传输持续时间直方图
    for value in &stats.transfer_duration {
        handles.nixl_xfer_time_seconds.observe(*value);
    }
    // 记录发布后时间直方图
    for value in &stats.post_duration {
        handles.nixl_post_time_seconds.observe(*value);
    }
    // 记录传输字节数直方图（将 u64 转为 f64）
    for value in &stats.bytes_transferred {
        handles.nixl_bytes_transferred.observe(*value as f64);
    }
}
```

```

}
// 记录描述符数量直方图
for value in &stats.num_descriptors {
    handles.nixl_num_descriptors.observe(*value as f64);
}
// 累加失败传输计数
handles
    .nixl_num_failed_transfers
    .inc_by(stats.num_failed_transfers.iter().sum());
// 累加失败通知计数
handles
    .nixl_num_failed_notifications
    .inc_by(stats.num_failed_notifications.iter().sum());
// 累加 KV 过期请求数
handles.nixl_num_kv_expired_reqs.inc_by(stats.num_kv_expired_reqs.iter().sum());
}

```

rust/src/metrics/src/scheduler.rs

定义 Mooncake 和 NIXL 连接器的指标结构、标签和桶，注册到 Prometheus 注册表。

```

// Mooncake 操作时间直方图桶，与 Python 端 MooncakeStorePromMetrics 一致。
const MOONCAKE_OPERATION_TIME_BUCKETS: [f64; 15] = [
    1e-3, 5e-3, 1e-2, 5e-2, 1e-1, 2e-1, 3e-1, 4e-1, 5e-1, 7.5e-1, 1.0, 1.5, 2.0, 3.0, 4.0,
];

// NIXL 传输时间直方图桶。
const NIXL_XFER_TIME_BUCKETS: [f64; 12] = [
    0.005, 0.01, 0.025, 0.05, 0.075, 0.1, 0.2, 0.3, 0.5, 0.75, 1.0, 5.0,
];

// Mooncake 操作标签，包含动态的 operation 和 status。
#[derive(Clone, Debug, Hash, PartialEq, Eq, EncodeLabelSet)]
pub struct MooncakeOperationLabels {
    pub model_name: String,
    pub engine: u32,
    pub operation: String,
    pub status: String,
}

// 类型别名，用于 Family 的构造。
pub type MooncakeOperationHistogramFamily =
    Family<MooncakeOperationLabels, Histogram, fn() -> Histogram>;
pub type MooncakeOperationCounterFamily = Family<MooncakeOperationLabels, U64Counter>;

```

rust/src/engine-core-client/src/protocol/stats.rs

定义从 Python 反序列化的数据结构，将 kv_connector_stats 从不透明的 OpaqueValue 变为强类型枚举。

```

// NIXL 连接器传输统计信息。

```

```

#[derive(Debug, Clone, Default, PartialEq, Serialize, Deserialize)]
#[serde(deny_unknown_fields)]
pub struct NixlStats {
    pub transfer_duration: Vec<f64>,
    pub post_duration: Vec<f64>,
    pub bytes_transferred: Vec<u64>,
    pub num_descriptors: Vec<u64>,
    pub num_failed_transfers: Vec<u64>,
    pub num_failed_notifications: Vec<u64>,
    pub num_kv_expired_reqs: Vec<u64>,
}

// Mooncake 存储操作名称。
#[derive(Debug, Clone, Copy, PartialEq, Eq, PartialOrd, Ord, Serialize, Deserialize)]
#[serde(rename_all = "snake_case")]
pub enum MooncakeOperation {
    SaveExists,
    SavePut,
    LoadGet,
    LookupExists,
}

// 连接器特定的调度器遥测。
#[derive(Debug, Clone, PartialEq, Serialize, Deserialize)]
#[serde(untagged)]
pub enum KvConnectorStats {
    Nixl(NixlStats),
    Mooncake(MooncakeStats),
    Multi(Box<MultiConnectorStats>),
    Other(BTreeMap<String, OpaqueValue>),
}

```

评论区精华

1. 强类型化重构: BugenZhao 指出初始实现未将统计信息强类型化, 并提供了修改建议 (commit bce428f), 要求作者 cherry-pick。这促使统计结构从 `BTreeMap<String, OpaqueValue>` 转变为显式的枚举 `KvConnectorStats`, 提高了类型安全性和可维护性。
 2. Serde 属性调整: coderabbitai[bot] 建议将 `NixlStats` 的 `#[serde(deny_unknown_fields)]` 改为 `#[serde(default)]`, 以允许缺失或额外字段, 避免因 Python 端字段变化导致反序列化失败。尽管此建议未被采纳 (作者保留了严格模式), 但体现了对跨语言兼容性的关注。
 3. 数据格式对齐: 讨论中强调了 Python 和 Rust 之间序列化格式的一致性, 特别是去除了旧代码中嵌套的 "data" 层级, 改为扁平结构, 以简化解析并避免双重嵌套问题。
- 强类型化重构 (design): 统计结构从不透明的 `BTreeMap` 转变为显式的枚举 `KvConnectorStats`, 提高了类型安全性。

- Serde 属性调整 (correctness): 作者未采纳, 保留了严格模式, 以确保与 Python 端的一致性。
- 数据格式对齐 (design): 修改了 Python 端的 to_dict 和 observe 方法, 确保输出扁平字典, 简化了 Rust 端的解析。

风险与影响

- 风险:
 1. 跨语言序列化兼容性: Rust 端的 KvConnectorStats 使用 serde(untagged) 枚举反序列化, 依赖于 Python 端 to_dict 输出的精确结构。若后续 Python 端修改统计字段, 可能导致 Rust 端反序列化失败或静默忽略。
 2. 严格反序列化模式: NixlStats 和 MooncakeRecord 使用了 `#[serde(deny_unknown_fields)]`, 若 Python 端意外添加新字段, 会引发错误。这有利于保持一致性, 但也增加了维护负担。
 3. 指标命名差异: Rust 端注册的 Mooncake 计数器名称 (如 `vllm:mooncake_store_operation`) 与 Python 端 (`vllm:mooncake_store_operation_total`) 略有不同, 这是因为 Rust 的 `prometheus-client` 库会自动追加 `_total` 后缀。需确保监控面板适配此差异。
- 影响:
 1. 用户影响: 使用 Rust 前端并配置了 Mooncake 或 NIXL 连接器的用户将能通过 Prometheus 端点获得详细的连接器性能指标 (如操作延迟、传输字节数、失败次数等), 从而改善对分布式 KV 缓存传输的监控和故障排查能力。
 2. 系统影响: 变更涉及 Rust 和 Python 跨层协议, 影响了统计信息的序列化格式。原有依赖 `kv_connector_stats` 字段的代码需适配新的扁平结构。
 3. 团队影响: 为 Rust 前端补齐了与 Python 前端对齐的指标能力, 减少了前端实现的功能差异, 有利于双前端的统一运维。 - 风险标记: 跨语言兼容性, 类型定义变更, 严格反序列化

关联脉络

- PR #47941 [EC Connector] P2P NIXL + CPU EC Connector: 同为 KV 连接器扩展, 涉及 NIXL 传输, 可能使用了类似的指标收集机制。
- PR #54878 [Bugfix][KV Connector] Fix DecodeBenchConnector prefix block selection: 同属 KV 连接器模块, 修复了连接器逻辑, 可能影响统计信息收集。
- PR #51381 [Core][KV Events] Echo session_id on GPU BlockStored events: 为 KV 缓存事件添加会话标识, 与本 PR 的指标记录同属监控增强。