PR #46747 完整报告

vllm-project/vllm

[Bugfix][V1][Multimodal] Recover from P0/P1 processor cache drift (#46747)

合并时间：2026-08-12 12:06

原文链接：http://prhub.com.cn/vllm-project/vllm/pull/46747

执行摘要

- 一句话：多模态 P0/P1 缓存漂移改为可自愈重试，不再断言崩溃
- 推荐动作：值得精读。该 PR 展示了两个高价值设计：一是把 " 不可恢复的断言 " 改造成 " 类型化可重试错误 + 批量失效 + 客户端重试自愈 " 的完整恢复链路；二是处理 msgspec array_like 位置化序列化与 Rust 镜像协议同步的工程细节。对于 V1 引擎与多模态 IPC 缓存相关开发，建议重点阅读 vllm/multimodal/cache.py 的异常路径与 EngineCore._handle_mm_cache_miss 的接线方式，并关注 Rust 前端恢复路径的后续落地。

功能与动机

PR body 明确指出：P0 影子命中时转发 data=None 并信任 P1 仍持有条目，但两个进程更新字节预算 LRU 缓存的顺序不同（P0 并发、P1 串行），缓存会漂移，导致 MultiModalReceiverCache.get_and_update_item 命中 assert mm_item is not None 使请求不可恢复失败。BaseMultiModalCache 文档化的 " 镜像驱逐不变量 " 在跨进程并发前端下无法保证，因此漂移应当可恢复而非致命。作者还补充了生产实证：修复在真实图像流量下已稳定运行约 2 周，无崩溃、无引擎重启。

实现拆解

1. P1 接收端：断言改为可重试异常（vllm/multimodal/cache.py）。新增 MultiModalCacheMissError(RuntimeError)，携带 mm_hashes 列表；MultiModalReceiverCache.get_and_update_item 在收到 data=None 且缓存未命中时改为抛出该异常（原先为 assert mm_item is not None）；BaseMultiModalReceiverCache.get_and_update_features 先 touch 全部 key 再逐条更新，聚合一次请求内全部漂移 hash 后统一抛出，避免每条目一次重试；基类 BaseMultiModalProcessorCache 新增默认空实现的 invalidate()，MultiModalProcessorOnlyCache 覆写为从 LRU 中 pop 陈旧影子条目。
2. 输出协议扩展（vllm/v1/engine/__init__.py）。EngineCoreOutput 追加 mm_cache_miss_hashes: list[str] | None = None 字段并置于结构体末尾，使 array_like=True 的位置化 msgpack 序列化保持向后兼容；注意 omit_defaults=True 对 array_like 结构无效，因此该字段平时也会在线缆上编码（值为 None）。
3. EngineCore 接线（vllm/v1/engine/core.py）。process_input_sockets 的 ADD 分支在 preprocess_add_request 上捕获 MultiModalCacheMissError，调用新增

_handle_mm_cache_miss 直接向 output_queue 投递一个 FinishReason.ERROR 的可重试输出，带 mm_cache_miss_hashes，请求不再进入调度器；该分支先于通用 Exception 分支捕获。

4. 前端自愈（vllm/v1/engine/async_llm.py）。_run_output_handler 从 renderer.mm_processor_cache 取得 P0 影子缓存，在输出块处理循环中读取 eco.mm_cache_miss_hashes 并对每个 hash 调用 mm_processor_cache.invalidate()；字段为 None 时热路径零开销。客户端重试时 P0 走 MISS 路径携带数据重发，重新填充 P1。
5. Rust 客户端协议镜像（rust/src/engine-core-client/src/protocol/output.rs 等）。Python 端 array_like 元组变长导致 Rust 解码报 LengthMismatch (expected 13)，因此在 Rust EngineCoreOutput 镜像新增 mm_cache_miss_hashes: Option<Vec<String>>，同步更新 python_compat.py 与 expect-test 快照；多个 Rust 测试构造器按 rust/AGENTS.md 改为 ..Default::default()。Rust 前端尚未实现 mm 处理器缓存，恢复路径保留 TODO。
6. 测试配套（tests/multimodal/test_cache.py）。新增 _StubModelConfig 免 HF 下载直接构造缓存，两个 CPU-only 测试：test_mm_cache_miss_raises_and_recovers 覆盖 " 漂移 → 抛错 → invalidate → 重发 → P1 重新填充 → 后续 hash-only 请求成功 " 全链路；test_mm_cache_miss_batches_all_drifted_hashes 验证一次请求内多 hash 聚合上报且非漂移条目仍被摄入。

关键文件：

- vllm/multimodal/cache.py（模块 缓存层；类别 source；类型 core-logic；符号 MultiModalCacheMissError, BaseMultiModalProcessorCache.invalidate, MultiModalProcessorOnlyCache.invalidate, BaseMultiModalReceiverCache.get_and_update_features）：核心修复所在：新增 MultiModalCacheMissError、将断言改为可重试异常、聚合整批漂移 hash、新增 invalidate 失效机制。
- vllm/v1/engine/core.py（模块 引擎核心；类别 source；类型 dependency-wiring；符号 EngineCore.process_input_sockets, EngineCore._handle_mm_cache_miss）：引擎核心输入路径接线：捕获 MultiModalCacheMissError，并以带 mm_cache_miss_hashes 的输出返回可重试响应。
- tests/multimodal/test_cache.py（模块 单元测试；类别 test；类型 test-coverage；符号 _StubModelConfig, test_mm_cache_miss_raises_and_recovers, test_mm_cache_miss_batches_all_drifted_hashes）：新增两个 CPU-only 单元测试覆盖漂移恢复全链路与批量聚合，并用 _StubModelConfig 免网络构造缓存。
- vllm/v1/engine/__init__.py（模块 引擎协议；类别 source；类型 data-contract；符号 EngineCoreOutput）：EngineCoreOutput 新增 mm_cache_miss_hashes 字段并置于末尾，保持 array_like 位置化序列化向后兼容。
- vllm/v1/engine/async_llm.py（模块 前端引擎；类别 source；类型 core-logic；符号 AsyncLLM._run_output_handler）：前端自愈：输出处理器根据 mm_cache_miss_hashes 对 P0 影子缓存执行 invalidate，热路径零开销。

- rust/src/engine-core-client/src/protocol/output.rs（模块 协议层；类别 source；类型 data-contract；符号 EngineCoreOutput）：Rust 客户端镜像新增 mm_cache_miss_hashes 字段，修复 array_like 元组变长导致的解码失败；恢复路径保留 TODO。
- rust/src/engine-core-client/src/tests/python_compat.py（模块 兼容测试；类别 test；类型 test-coverage）：保持 Python/Rust 编码 fixture 一致，防止未来 schema 变更再次导致 LengthMismatch。

关键符号：MultiModalCacheMissError.init, BaseMultiModalProcessorCache.invalidate, MultiModalProcessorOnlyCache.invalidate, MultiModalReceiverCache.get_and_update_item, BaseMultiModalReceiverCache.get_and_update_features, EngineCore.process_input_sockets, EngineCore._handle_mm_cache_miss, AsyncLLM._run_output_handler, test_mm_cache_miss_raises_and_recovers, test_mm_cache_miss_batches_all_drifted_hashes

关键源码片段

vllm/multimodal/cache.py

核心修复所在：新增 MultiModalCacheMissError、将断言改为可重试异常、聚合整批漂移 hash、新增 invalidate 失效机制。

```
class MultiModalCacheMissError(RuntimeError):
    """P1 接收端缓存未命中时抛出的可重试异常。

    P0（前端）只保留 P1（引擎）缓存的元数据影子副本，
    影子命中时发送 `data=None`。由于两端 LRU 更新顺序不同
    （P0 并发、P1 串行），缓存可能漂移——P0 引用的条目已被 P1 驱逐。
    用异常替代 assert，让引擎端返回可重试响应，
    P0 通过 `invalidate()` 丢弃陈旧条目后客户端携带数据重发。
    """

    def __init__(self, mm_hashes: list[str]) -> None:
        super().__init__(
            f"Multi-modal items {mm_hashes} are not in the receiver (P1) cache and "
            "no data was provided to recompute them (P0/P1 cache drift); the request "
            "should be retried with the multi-modal data attached."
        )
        self.mm_hashes = mm_hashes


class BaseMultiModalReceiverCache(
    BaseMultiModalCache[MultiModalKwargsItem | None, MultiModalKwargsItem]
):
    def get_and_update_features(
        self, mm_features: list["MultiModalFeatureSpec"]
    ) -> list["MultiModalFeatureSpec"]:
        # 先 touch 全部缓存 key，避免更新过程中 LRU 驱逐正在使用的条目
        for feature in mm_features:
```

```
            cache_key = feature.mm_hash or feature.identifier
            self.touch_receiver_cache_item(cache_key, feature.data)

        # 聚合本次请求内所有漂移 hash，统一抛一次异常；
        # 否则 k 个漂移条目需要 k 次客户端重试（每次只上报一个 hash）
        missing_mm_hashes: list[str] = []
        for feature in mm_features:
            cache_key = feature.mm_hash or feature.identifier
            try:
                feature.data = self.get_and_update_item(feature.data, cache_key)
            except MultiModalCacheMissError as e:
                missing_mm_hashes.extend(e.mm_hashes)
        if missing_mm_hashes:
            raise MultiModalCacheMissError(missing_mm_hashes)
        return mm_features


class MultiModalReceiverCache(BaseMultiModalReceiverCache):
    @override
    def get_and_update_item(
        self, mm_item: MultiModalKwargsItem | None, mm_hash: str
    ) -> MultiModalKwargsItem:
        if (cached_item := self._cache.get(mm_hash)) is not None:
            return cached_item

        # 无数据且未命中：P0 依据影子缓存发送了 `data=None`，但两端缓存已漂移。
        # 抛出可重试异常（而非 assert），引擎端据此通知 P0 失效并让客户端重发。
        if mm_item is None:
            raise MultiModalCacheMissError([mm_hash])

        self._cache[mm_hash] = mm_item
        return mm_item


class MultiModalProcessorOnlyCache(BaseMultiModalProcessorCache):
    @override
    def invalidate(self, mm_hash: str) -> None:
        # 丢弃陈旧影子条目：下次请求该 hash 时重新走 MISS 路径，
        # 客户端携带数据重发即可重新填充 P1（配合 `MultiModalCacheMissError`）。
        self._cache.pop(mm_hash, None)
```

vllm/v1/engine/core.py

引擎核心输入路径接线：捕获 MultiModalCacheMissError，并以带 mm_cache_miss_hashes 的输出返回可重试响应。

```
    # process_input_sockets 的 ADD 分支：preprocess 阶段捕获漂移异常
    if request_type == EngineCoreRequestType.ADD:
        req: EngineCoreRequest = add_request_decoder.decode(data_frames)
        try:
```

```
            request = self.preprocess_add_request(req)
        except MultiModalCacheMissError as e:
            # P0/P1 影子缓存漂移：返回可重试信号，P0 丢弃陈旧条目，
            # 客户端随后携带数据重发；请求不进入调度器。
            self._handle_mm_cache_miss(req, e)
            continue
        except Exception:
            self._handle_request_preproc_error(req)
            continue

    def _handle_mm_cache_miss(
        self, request: EngineCoreRequest, err: MultiModalCacheMissError
    ) -> None:
        """为 P0/P1 缓存漂移未命中返回可重试响应。

        通过 `EngineCoreOutput.mm_cache_miss_hashes` 上报所有漂移 hash，
        前端据此失效 sender cache 中的陈旧条目，客户端携带数据重发。
        使用 warning 而非 exception 记录：该错误是预期内的、可自愈的。
        """
        logger.warning(
            "Multi-modal cache miss for request %s (mm_hashes=%s): P0/P1 cache "
            "drift; returning a retryable response so the items are resent with data.",
            request.request_id,
            err.mm_hashes,
        )
        self.output_queue.put_nowait(
            (
                request.client_index,
                EngineCoreOutputs(
                    engine_index=self.engine_index,
                    finished_requests={request.request_id},
                    outputs=[
                        EngineCoreOutput(
                            request_id=request.request_id,
                            new_token_ids=[],
                            finish_reason=FinishReason.ERROR,
                            mm_cache_miss_hashes=err.mm_hashes,
                        )
                    ],
                ),
            )
        )
```

评论区精华

Review 核心讨论有两点。其一是 DarkLight1337 在 tests/multimodal/test_cache.py 的 review 中建议 "It would be better to test this E2E"（E2E 测试更好）；作者 WillZZZy 回应称该修复已在内部生产环境用真实图像流量运行约 2 周，无 assert mm_item is not None 崩

溃、无引擎重启，并给出 MMMU-Pro 前后对比（+0.1pp 与 +0.5pp，McNemar p ≈ 0.95 与 0.47，均在 ~0.5pp 噪声范围内），评审者表示认可。其二是 Rust 侧 schema 问题：EngineCoreOutput 为 array_like=True，Python 端加字段拉长 msgpack 元组导致 Rust 解码失败，且 omit_defaults=True 对 array_like 无效；作者镜像新增 Rust 字段并同步 python_compat.py，同时将测试构造器改为 ..Default::default()，Rust 前端恢复路径因暂不支持 mm 缓存而保留 TODO。

- E2E 测试建议与生产验证回应 (testing): 评审者接受单元测试 + 生产验证的组合（"Thanks for confirming this!"），未强制要求补 E2E。
- array_like 输出协议变更导致 Rust 解码失败 (design): 通过在 Rust EngineCoreOutput 镜像新增 mm_cache_miss_hashes 字段并同步 python_compat.py 解决；测试构造器改用 ..Default::default()；Rust 前端恢复逻辑保留 TODO。

风险与影响

- 风险：
  - 跨语言 wire 协议风险（rust/src/engine-core-client/src/protocol/output.rs）：EngineCoreOutput 是 array_like 位置化 msgpack，任何 Python 端字段增删都会改变元组长度，Rust 端未同步镜像将整体解码失败（本次已出现 expected 13 的 LengthMismatch）。python_msgpack_fixtures_match_rust_encoding 提供了一定保护，但未来 schema 演进仍需 Python/Rust 双端同步。
  - 依赖客户端重试行为：恢复路径要求客户端收到 FinishReason.ERROR 后携带数据重发。若客户端不重试（例如 Rust 前端，当前保留 TODO），该请求仍以错误结束——只是从 " 引擎崩溃 + 请求不可恢复 " 降级为 " 可重试错误 "，对 API 用户属于行为语义变化。
  - 并发窗口：P0 的 invalidate() 与并发进行中的其他请求可能竞争，同一 hash 可能被多次重传（重复处理），但 pop(mm_hash, None) 幂等，不会破坏正确性。
  - 核心输入路径回归面：process_input_sockets 是 V1 引擎核心输入路径，新增的 except 分支改变了异常处理顺序（MultiModalCacheMissError 先于通用 Exception 捕获），需确认 preprocess 阶段其他异常不受影响；PR 未提供覆盖 EngineCore 层的 E2E 测试，核心恢复路径仅有单元测试覆盖。
- 影响：
  - 用户 / 服务方：开启多模态 IPC 缓存（--mm-processor-cache-*）时，闲置后仅按 hash 重发缓存的客户端不再触发 assert mm_item is not None 崩溃；一次重试即可整批恢复，显著降低长尾失败率。
  - 系统：新增 EngineCoreOutput.mm_cache_miss_hashes 会随所有 V1 输出编码（array_like 无法省略），带来极小带宽开销；happy path 无额外逻辑开销。
  - 团队 / 工程：涉及 Python 引擎核心、多模态缓存与 Rust engine-core-client 三块，跨语言协议变更需双端同步；Rust 前端后续若引入 mm processor cache，可直接复用该恢复协议。
  - 风险标记：跨语言协议变更，核心输入路径变更，依赖客户端重试行为，Rust 恢复路径 TODO，缺少 E2E 覆盖

关联脉络

- 暂无明显关联 PR