PR #49328 完整报告

vllm-project/vllm

[KV Offload] Fix failed-load livelock by marking the lookup verdict as a miss

合并时间：2026-08-09 15:25

原文链接：http://prhub.com.cn/vllm-project/vllm/pull/49328

执行摘要

- 一句话：修复次级 tier 加载失败导致请求永久卡死的 livelock
- 推荐动作：值得精读。核心看点是：mark_miss + enqueue-once 断言如何用最小 API 面（没有为 tier 新增任何回调）修复一类典型的 " 缓存与真实状态脱节导致 livelock" 问题；performance 驱动的设计取舍（用实测 benchmark 数据否决 stat 方案而非凭直觉）；以及边界情形处理（短读 vs 瞬时错误的删除决策、close 失败语义、部分成功上报）。对 KV offload、异步缓存系统或共享存储 I/O 层设计有借鉴意义。

功能与动机

PR 修复 issue #49176 报告的问题：次级 tier 块加载失败一次后请求永远无法完成，调度器每个调度步重复发起同一个注定失败的 promotion，直至请求被中止。issue 指出 root cause 在 vllm/v1/kv_offload/tiering/ 的调度侧代码，平台无关（无 GPU 即可复现），并强调 "one bad block file converts a request into an infinite promotion loop and a busy-loop of wasted I/O and scheduler work"。PR body 补充说明 "Because the cache is refcounted per block, the stale verdict is served to every request overlapping those blocks, so one bad file wedges more than the request that first touched it, and it's silent"——单个坏文件的影响会扩散到所有重叠该块的请求且静默发生。作者在 Apple Silicon 的 Metal/MLX 后端上复现（文件系统 tier 频繁出现 torn block），但缺陷位于共享 tiering 层，普通 CPU 环境即可复现。

实现拆解

实现按 5 步拆解：

1. 为异步查询缓存引入外部校正原语：vllm/v1/kv_offload/tiering/async_lookup.py 新增 mark_miss(keys)，将缓存中的 verdict 直接翻转为 False（不删除条目，保持 cleanup() 反向索引一致性）；同时 drain_results() 增加 enqueue-once 不变式断言，拒绝迟到的或重复的 worker 结果覆盖已定论的 verdict，防止 stale True 复活。这一层是整个修复的地基，所有 tier 共用。
2. fs tier 在调度线程自报失败：vllm/v1/kv_offload/tiering/fs/manager.py 的 submit_load() 登记 job_id → keys 映射，并把原来的 functools.partial(batch_load_block, ...) 包装为内层 load_task()：捕获 OSError 后从异常读取 num_succeeded（记录失败前已成功加载的块数）存入 _load_progress，再重新抛出让 job 按失败上报。get_finished_jobs() 在调度线程（唯一允许触碰 async cache 的线

程）上弹出 _load_job_keys 与 _load_progress，对失败块调用 mark_miss(failed)，并借助 successful_keys 保留已成功加载的块，避免整批重算。

3. 收窄块文件删除策略：vllm/v1/kv_offload/tiering/fs/io.py 的 _load_block() 与 csrc/fs_io.cpp 的 _load_block() 同步改为：仅在 " 可证明的短读 "（读完成但字节数 < block_size）时删除文件（store 是原子的，过短文件即真实损坏）；open 失败、读错误、close 失败都不动文件，避免把瞬时故障变成永久数据丢失。C 版 batch_load_block 在异常上附加 num_succeeded 属性（PyObject_SetAttrString），Python fallback 用 exc.num_succeeded = i 对齐，实现 C/Python 双路径行为一致。
4. obj tier 同步适配：vllm/v1/kv_offload/tiering/obj/manager.py 使用相同模式，在 get_finished_jobs() 中对失败的 load job 计算 successful_keys 差集后调用 mark_miss；由于 nixl 目前整批上报状态，实际会标记全部 key（作者注明保持 consumer-only，部分上报留作 follow-up，且 mark_miss 读取 successful_keys 保证未来正确性）。
5. 测试与验证配套：tests/v1/kv_offload/tiering/ 下 4 个测试文件新增回归用例，全部 use_c_ext 参数化覆盖 C/Python 双路径，包括：失败加载翻转 verdict 并删除损坏文件、批量部分失败保留已加载块、首个块失败整批标记 miss、瞬时错误（ELOOP）保留文件、mark_miss 不重探 + enqueue-once 断言等；另用 mutation 验证（中和 mark_miss 会使 livelock 测试失败）。测试模拟 open 失败从 chmod 000 改为符号链接循环 ELOOP，以兼容 CI 以 root 运行时绕过文件权限位的差异。

关键文件：

- vllm/v1/kv_offload/tiering/fs/manager.py（模块 文件系统；类别 source；类型 core-logic；符号 submit_load, load_task, get_finished_jobs）：修复的核心承载点：submit_load 登记 job keys 并包装 load_task 记录 num_succeeded，get_finished_jobs 在调度线程对失败 key 调用 mark_miss，并借助 successful_keys 实现批量部分成功保留。
- vllm/v1/kv_offload/tiering/async_lookup.py（模块 查询缓存；类别 source；类型 core-logic；符号 mark_miss, drain_results）：新增 mark_miss 原语与 drain_results 的 enqueue-once 断言，是让 " 失败能校正缓存 " 的结构性前提，所有次级 tier 共用。
- vllm/v1/kv_offload/tiering/fs/io.py（模块 文件读写；类别 source；类型 core-logic；符号 _load_block, batch_load_block）：Python 路径的删除策略收窄：仅可证明的短读删除文件，瞬时错误保留文件，并给 batch_load_block 异常附加 num_succeeded。
- csrc/fs_io.cpp（模块 C 扩展；类别 source；类型 core-logic；符号 _load_block, batch_load_block）：C 扩展路径与 Python 路径保持同一契约，是使用 fs_io_C 时修复生效的关键；同时为 batch_load_block 异常附加 num_succeeded。
- tests/v1/kv_offload/tiering/test_fs_tier.py（模块 文件系统；类别 test；类型 test-coverage；符号 test_failed_load_corrects_verdict_and_removes_corrupt_file, test_batched_partial_load_failure_keeps_loaded_blocks, test_batched_load_first_block_fails_marks_whole_batch, test_transient_load_failure_leaves_file）：最全面的回归测试：覆盖失败加载纠正 verdict + 删除损坏文件、批量部分失败保留已加载块、首块失败整批 miss、瞬时错误保留文件四大契约，且全部 C/Python 双路径参数化。
- vllm/v1/kv_offload/tiering/obj/manager.py（模块 对象存储；类别 source；类型 core-logic；符号 submit_load, get_finished_jobs）：obj tier 同步获得同样的失败校正能

力，证明该修复模式可推广到所有次级 tier 实现。

- tests/v1/kv_offload/tiering/test_async_lookup.py（模块 查询缓存；类别 test；类型 test-coverage；符号 test_mark_miss_flips_cached_verdict_without_reprobing, test_enqueue_once_invariant_enforced）：针对 mark_miss 语义与 enqueue-once 不变式的单元验证，是缓存层契约的直接守护。
- tests/v1/kv_offload/tiering/test_tiering_offloading.py（模块 卸载调度；类别 test；类型 test-coverage；符号 test_failed_promotion_finalizes_primary_with_failure, test_successful_promotion_finalizes_primary_with_success）：验证 TieringOffloadingManager 在 promotion 失败 / 成功时仍正确 finalize 主 tier 槽位，确认修复不破坏管理器职责边界。
- tests/v1/kv_offload/tiering/test_obj_tier.py（模块 对象存储；类别 test；类型 test-coverage；符号 test_failed_load_marks_verdict_negative）：obj tier 的 livelock 回归测试，证明同一修复模式在非文件系统 tier 上同样有效。

关键符号：mark_miss (async_lookup.py), drain_results (async_lookup.py), submit_load (fs/manager.py), load_task (fs/manager.py), get_finished_jobs (fs/manager.py), _load_block (fs/io.py), batch_load_block (fs/io.py), _load_block (csrc/fs_io.cpp), batch_load_block (csrc/fs_io.cpp), get_finished_jobs (obj/manager.py)

关键源码片段

vllm/v1/kv_offload/tiering/fs/manager.py

修复的核心承载点：submit_load 登记 job keys 并包装 load_task 记录 num_succeeded, get_finished_jobs 在调度线程对失败 key 调用 mark_miss，并借助 successful_keys 实现批量部分成功保留。

```
@override
def submit_load(self, job_metadata: JobMetadata) -> None:
    job_id = job_metadata.job_id
    # 记录本次 load 的 keys，失败的 promotion 才能在 get_finished_jobs
    # 里精确地把失败 key 标记为 miss，而不是整批处理。
    keys = list(job_metadata.keys)
    self._load_job_keys[job_id] = keys
    paths = [self.file_mapper.get_file_name(key) for key in keys]
    offsets = [int(bid) * self._block_size for bid in job_metadata.block_ids]

    def load_task() -> None:
        try:
            batch_load_block(
                paths,
                self._primary_kv_view,
                offsets,
                self._block_size,
                self._use_o_direct,
            )
        except OSError as exc:
```

```
            # 在池工作线程上执行。记录失败前已成功加载的块数，供
            # get_finished_jobs 保留这些块；该写发生在 task_done 发布
            # 之前，调度线程在 GIL 下读到已完成的 job 时必然可见。
            num_succeeded = getattr(exc, "num_succeeded", 0)
            self._load_progress[job_id] = num_succeeded
            logger.debug(
                "Load of %d blocks for job %s failed at block %d: %s",
                len(paths), job_id, num_succeeded, exc,
            )
            raise

    self._pool.enqueue_load(job_id, 1, [load_task])


@override
def get_finished_jobs(self) -> Iterable[JobResult]:
    """Collect finished jobs; a failed promotion marks only its failed keys
    as a miss here (scheduler thread)."""
    results = []
    for job_id, success in self._pool.get_finished():
        if self.events is not None:
            keys = self._store_job_keys.pop(job_id, None)
            if success and keys:
                self.events.append(OffloadingEvent(
                    keys=keys, medium=self.medium,
                    removed=False, locality=self.locality,
                ))
        load_keys = self._load_job_keys.pop(job_id, None)
        num_succeeded = self._load_progress.pop(job_id, 0)
        if load_keys is not None and not success:
            # 批量 load 在第一个坏块处停止，此前加载成功的块保留在主 tier
            # （经 successful_keys 上报）；只有失败块及其后续块被标记 miss，
            # 该请求转而在 GPU 上重算这些块。
            successful = load_keys[:num_succeeded]
            failed = load_keys[num_succeeded:]
            self._lookup_manager.mark_miss(failed)
            results.append(JobResult(
                job_id=job_id,
                success=False,
                successful_keys=tuple(successful) if successful else None,
            ))
        else:
            results.append(JobResult(job_id=job_id, success=success))
    return results
```

vllm/v1/kv_offload/tiering/async_lookup.py

新增 mark_miss 原语与 drain_results 的 enqueue-once 断言，是让 " 失败能校正缓存 " 的结构性前提，所有次级 tier 共用。

```
  def drain_results(self) -> None:
      """Apply pending worker results to _lookup_state.

      Called from lookup() before checking state.
      """
      while True:
          try:
              batch = self._pending_results.get_nowait()
          except queue.Empty:
              break
          for key, result in batch:
              state = self._lookup_state.get(key)
              if state is not None:
                  # 一个 key 只会入队探测一次，所以已定论的 verdict 不应再收到
                  # 第二个结果。强制该不变式可防止迟到的 / 重复的结果复活一个
                  # 已修正的 stale True，从而重开 failed-load livelock。
                  assert state.result is None, (
                      "cached key received a second lookup result; the "
                      "enqueue-once invariant is broken and could reopen the "
                      "failed-load livelock"
                  )
                  state.result = result


  def mark_miss(self, keys: Collection[OffloadKey]) -> None:
      """Force the cached verdict for ``keys`` to False after a failed load, so
      the scheduler stops re-issuing the doomed promotion (livelock, #49176).
      Keys with no cached entry are skipped."""
      for key in keys:
          state = self._lookup_state.get(key)
          if state is not None:
              # 将缓存的 True 翻转为 False：后续 lookup 直接从缓存返回 MISS，
              # 不再发起新的 batch 探测（避免重探），请求在 GPU 上重算即可，
              # 结构上不可能再进入 promotion 循环。条目保留而非删除，保证
              # cleanup() 的反向索引一致。
              state.result = False
```

vllm/v1/kv_offload/tiering/fs/io.py

Python 路径的删除策略收窄：仅可证明的短读删除文件，瞬时错误保留文件，并给 batch_load_block 异常附加 num_succeeded。

```
  def _load_block(
      source_path: str,
      view: memoryview,
      offset: int,
      block_size: int,
      use_o_direct: bool = True,
  ) -> None:
      """Read one KV block from disk; remove the file only on a provable short
```

```
    read (a too-short file is genuine corruption) and leave it untouched on any
    other error."""
    fd: int | None = None
    view_slice = view.cast("B")[offset : offset + block_size]
    o_direct = O_DIRECT if use_o_direct else 0

    try:
        fd = os.open(source_path, os.O_RDONLY | o_direct)
        bytes_read = os.readv(fd, [view_slice])
        if bytes_read < block_size:
            # 短读 = 文件确实损坏（store 是原子的，正常写入不可能产生
            # 过短文件），删除后后续请求 lookup 即为 miss；删除失败不能
            # 掩盖原始短读错误，所以单独捕获清理异常。
            try:
                os.remove(source_path)
            except OSError as cleanup_exc:
                logger.warning(
                    "Failed to remove short-read file %s: %s",
                    source_path,
                    cleanup_exc,
                )
            raise OSError(
                f"Short read: expected {block_size} bytes, read {bytes_read}")
    finally:
        if fd is not None:
            os.close(fd)
```

评论区精华

评审讨论围绕 5 个核心交锋展开：

- tier API 是否新增回调：orozery 反对早期方案中的 on_load_failed() 管理器回调："The tier should already have the information of failed load jobs... He does not need the tiering manager to report it back to him." 作者接受并移除该 API，改为 tier 在 get_finished_jobs() 自报。
- lookup 是否做 size 校验（stat vs access）：varun-sundar-rabindranath 质疑热点路径开销 "lookups are called very frequently and we should keep it light"；作者给出实测对比（Linux ext4 上 HIT 场景 access 约 420 ns vs stat 约 490 ns，约 1.16x；macOS/APFS 约 1.35x），最终放弃 size 校验，回归 access 存在性检查，并论证 " 防止 bit-rot 靠 size 检测本就不可靠 "。
- 失败后缓存语义：标记 False 还是删除条目：varun 提出 "disk-offloading should be best-effort... The policy could be that the request gets single try, and if it fails then it just recomputes on the GPU"；orozery 担心删条目会导致冗余重探。结论是标记 False（单次尝试、无重探），作者还补充了 drain_results 的 enqueue-once 断言作为硬化。
- 防御性判断 vs 依赖不变式：varun 问 state.result is None 是否为防御性检查，作者确认后 varun 建议 "Add an assert to check that it is enforced"，最终改为显式断言。

- close 失败语义与测试健壮性：orozery 指出 C 路径 close 失败不应让加载失败（数据已入缓冲区），触发 mark_miss 是误报；作者修正 C 路径。测试中 chmod 000 触发 EACCES 在 root 运行的 CI 下失效（root 无视权限位），作者改用 ELOOP 符号链接循环模拟瞬时 open 失败。
  - tier API 是否新增 on_load_failed 回调 (design): 移除该回调，tier 在 get_finished_jobs() 中自行校正自己的 lookup 缓存，API 面为零新增。
  - lookup 热路径 stat vs access 的性能权衡 (performance): 放弃 size 校验，回归 access 存在性检查；同时确认 size 校验对 bit-rot 本就不可靠。
  - 失败后缓存语义：标记 False vs 删除条目 (design): 标记 False 且保留条目：缓存 MISS 直接返回、不重探，结构上不可能循环；请求结束后条目清理，完整块对其他请求仍 HIT。
  - mark_negative 命名与防御性检查的处理 (style): 重命名为 mark_miss；防御性跳过改为显式 assert 强化 enqueue-once 不变式。
  - close 失败是否视为加载失败 (correctness): C 路径改为：全量读后的 close 失败不视为加载失败，数据已入缓冲区即成功。
  - 50321 合并后适配 successful_keys 部分结果 (design): fs tier 实现部分成功保留；obj tier 因 check_xfer_state 只能整批上报，保持整批标记并留作 follow-up。
  - CI root 权限导致测试失败：EACCES 改为 ELOOP (testing): 改用符号链接循环产生 ELOOP，对所有用户表现一致，测试与代码路径均不受影响。

风险与影响

- 风险：具体风险点如下：
- 运行时断言引入崩溃面：async_lookup.drain_results() 的 enqueue-once 断言是生产代码中的主动崩溃点。当前逻辑保证每个 key 只入队一次，若未来改动破坏该不变式（例如引入重试探测），vLLM 会直接 AssertionError。作者有意为之且 mutation 验证过，但团队需要知晓这是 " 宁可崩溃也不静默出错 " 的取舍。
- 磁盘持续故障下的重复单次失败：mark_miss 是请求级、单次尝试策略。若磁盘长期处于坏块状态，每个新请求都会对同一坏块再发起一次（注定失败的）promotion 再转 GPU 重算，形成 " 每请求一次 I/O 浪费 + 重算 " 的恒定开销，虽不会死循环但会在故障期放大负载。
- 删除策略收窄后的磁盘残留：相比 #49152 的 " 任何失败都删文件 "，现在只有短读才删除。瞬时错误（EMFILE/EIO/ENOENT）留下的文件会继续占据磁盘空间；连接器查找仍会 HIT 它们，后续请求可能重复遭遇瞬时失败并重算。
- 共享文件系统假设：短读删除策略依赖 "store 原子性 "（临时文件 + os.replace），且 _load_block 的删除与 get_finished_jobs 的 mark_miss 分处不同线程，依赖 GIL 与 task_done 发布顺序保证 _load_progress 可见性——这些假设在文档注释中已写明，但属于需要维护者共同知晓的隐式契约。
- obj tier 整批误标 miss：obj/manager.py 因 check_xfer_state 只能整批上报状态，成功块也会被标记 miss 并重算。正确性无碍，但批量场景下会浪费已到达的数据，作者已说明留作 follow-up。

- C/Python 双路径一致性：num_succeeded 在 C 路径经 PyObject_SetAttrString 附加，Python 路径直接赋值属性，两处逻辑已对齐；但 C 路径若未来改动错误处理分支，需同步维护该契约。
- 影响：对用户的影响集中在启用 KV offload（fs/obj 次级 tier）的环境：文件系统不稳定时（Apple Silicon Metal 后端、网络盘、易出现 torn block 的场景）请求不再永久卡死，损坏块会在首次失败后被删除并在该请求内以 GPU 重算兜底；对完整块，由于 verdict 是请求级且随请求结束清理，其他请求仍可正常 HIT，blast radius 被有效控制。对系统而言，调度器不再为坏块反复做无效 promotion，节省了无效 I/O 与调度开销，这正是该 bug 被定义为 "busy-loop of wasted I/O" 的原因。对团队而言，本 PR 确立了两个值得固化的规则：次级 tier 必须自行校正自己的 lookup 缓存（无需新增 manager 回调 API），以及 " 单次尝试 + GPU 重算 + 仅确证损坏才删除 " 的 best-effort offload 哲学，后续新增 tier 实现需要遵循同样的缓存校正约定。
- 风险标记：核心路径变更，C/Python 双路径一致性，新增运行时断言，存储删除策略调整，obj tier 整批误标 miss

关联脉络

- PR #49152 [KV-offload][FS]：Batch store/load_block in C: 本 PR 直接构建在其 C 批处理 store/load 之上（fs/io.py 与 csrc/fs_io.cpp 共享），并修正其 ' 任何失败都删除块文件 ' 的过宽策略为仅短读删除。
- PR #50321 （讨论中提及的 PR，标题未在上下文中提供）：orozery 在 review 中要求适配其引入的 JobResult.successful_keys，本 PR 的批量部分成功保留逻辑依赖该能力。
- PR #49176 [Bug][KV Offload]: failed secondary-tier load livelocks the request — async lookup cache is never invalidated on load failure: 本 PR 修复的关联 issue，完整描述了 livelock 的逐步成因与复现脚本。
- PR #51161 [Bugfix][KV Offload] Handle chunked local attention in offloading scheduler: 近期合入的 KV offload 调度器 bugfix，与本次修复同属 vllm/v1 调度侧 tiering 稳定化演进线。