

PR #38941 完整报告

sgl-project/sglang

[Fix] Merge adjacent KV-row frees so a mid-page split under DCP cannot double-free

合并时间: 2026-09-14 21:44

原文链接: <http://prhub.com.cn/sgl-project/sglang/pull/38941>

执行摘要

- 一句话: 合并相邻 KV 行释放区间, 修 DCP 下页内接缝双重释放崩溃
- 推荐动作: 值得精读, 尤其推荐给维护 KV cache / radix cache / DCP 的工程师: 这是一个“小而准”的生产崩溃修复, 11 行源码解决整机级故障, 且修复点的选择有讲究。阅读时重点看三处设计决策: 一是为什么在 `free_kv_row` 这一层合并, 而不是让 allocator 放宽断言 (后者会掩盖真实 double free); 二是 `coalesce_ranges` 只合并严格相邻区间的边界处理, 以及它隐含的升序前置条件是否需要显式断言; 三是测试如何把 512/576 的线上几何按 64 倍缩放到 page 8 / 位置 9, 并用第二个用例锁住“有空隙的共享页仍然报错”这条安全线。建议顺手确认其他 `free_kv_row_segments` 调用点是否都已覆盖, 并评估为“保留前缀结束在页内”的接缝情形补一个用例。

功能与动机

PR body 给出了明确的线上故障动机: radix cache 服务在混合 Mamba + MLA 模型 (Kimi-K3、Kimi-Linear) 配 `--dcp-size 8` 时, 在普通负载下崩溃, 报 `mem_cache/allocator/base.py, free_segments` 的 `AssertionError: segment at 576 shares a page with the one ending at 576`, 作者描述为“Every scheduler dies at once”, 在 MMLU 式批量 (大量前缀共享的 logprob 请求同时结束) 下一分钟内必现, agentic 流量下数小时内也会出现。作者还指出断言并非仅“过于严格”: *Without the assertion it would be worse: the second piece would free a page the first piece already released.* 即若无断言就退化为真正的 double free。修复后作者用 Kimi-K3 (`--attention-backend cutedsl_mla --dcp-size 8`, radix cache 与 hierarchical cache 开启、draft-model 投机解码开启) 跑出 210 个带 radix 命中的 prefill batch 无断言, 而同样配置未修复的三台服务在同一负载下一分钟内全部死亡。

实现拆解

1. 入口与根因定位: `BasePrefixCache.free_kv_row` (`python/sglang/srt/mem_cache/base_prefix_cache.py`) 是所有前缀缓存共用的 KV 行归还入口。它把行位置区间切成 `(row[start:end], start)` 段交给 `free_kv_row_segments`, 再由 allocator 的 `free_full_segments` 归还页。DCP 下 allocator 页宽为 `page_size * dcp_size` (默认 $64 \times 8 = 512$), radix 树按该宽度分页, 但 cache-length cap (如 Mamba 组件的 track boundary 576) 仍落在未加宽的 64 token 网格, `cache_finished_req` 于是把请求尾部拆成「页内未对齐尾段」和「延迟截断尾段」两段相接区间, 接缝落在 512 token 页内部,

`free_segments` 因两段共享同一页而断言失败。

2. 新增纯函数 `coalesce_ranges` (`python/sglang/srt/mem_cache/common.py`)。单次线性扫描，仅当 `start == merged[-1][1]` (严格相接) 时把区间合并，遇到空隙则原样保留；放在 `free_kv_row_segments` 等释放工具同侧，保持释放语义内聚。这样 `allocator` 只看到一段页对齐区间，每个页只释放一次。
3. 释放入口接线：`free_kv_row` 内改为 `from ... import coalesce_ranges, free_kv_row_segments`，在构造段列表前先做 `coalesce_ranges(ranges)`。为什么把合并放在调用方而不是 `allocator`：`allocator` 的共享页断言是全局安全网，放松它会掩盖真实 `double free`；而在 `free_kv_row` 这一层，调用方刚好知道这些区间属于同一行、语义上是连续的尾部，是唯一具备“可安全合并”信息的层级。合并后每个区间仍携带自己的 `start`，`free_kv_row_segments` 按 `swa_evicted_seqlen` 切死 / 活两段的逻辑不受影响，所有继承 `BasePrefixCache` 的缓存 (`unified radix cache`、`HiCache` 变体、`chunk cache` 等) 自动受益。
4. 刻意保留的安全边界。`allocator` 侧断言一行未改：两段相邻区间之外的“有空隙但共享页”仍会被拒绝，说明本 PR 只消化“可证明等价”的合并，而不吞掉异常信号。
5. 测试与 CI 配套 (新增 `test/registered/unit/mem_cache/test_free_kv_row_coalesce.py`, 92 行)。用 `PagedTokenToKVPoolAllocator(size=16*8, page_size=8)` 把线上几何按 64 倍缩放 (`512 → 8`, `576 → 位置 9`)，`_make_cache_with_row` 只搭最小 `allocator` + 伪造 `req_to_token` 行，`_RowCache` 继承 `BasePrefixCache` 后除 `free_kv_row` 外全部抛 `NotImplementedError`，避免拉起完整调度栈；用例一验证 `free_kv_row(kv, [(8, 9), (9, 24)])` 后每个页恰好释放一次，用例二验证有空隙的 `[(8, 9), (12, 24)]` 仍抛 `AssertionError`。测试通过 `register_cpu_ci(est_time=10, suite="base-a-test-cpu")` 进入 CPU CI。性能上只是请求完成时对少量区间做一遍线性遍历，作者明确标注“Not applicable”。

关键文件：

- `python/sglang/srt/mem_cache/base_prefix_cache.py` (模块 前缀缓存；类别 `source`；类型 `core-logic`；符号 `free_kv_row`, `coalesce_ranges`)：修复的实际接线点：
`BasePrefixCache.free_kv_row` 是先合并区间再构造释放段的地方，也是所有前缀缓存共享的 KV 归还入口，决定了修复的覆盖面与不变量边界。
- `python/sglang/srt/mem_cache/common.py` (模块 缓存公共工具；类别 `source`；类型 `core-logic`；符号 `coalesce_ranges`, `free_kv_row_segments`)：新增核心工具函数 `coalesce_ranges`，是修复的语义核心：只合并严格相邻的半开区间，空档保留，从而在修掉崩溃的同时不削弱 `allocator` 的 `double-free` 断言；该文件同时定义了 `free_kv_row_segments` 的 SWA 水位线切分与页对齐断言。
- `test/registered/unit/mem_cache/test_free_kv_row_coalesce.py` (模块 内存缓存测试；类别 `test`；类型 `test-coverage`；符号 `TestFreeKvRowCoalesce`, `test_adjacent_ranges_split_mid_page_free_every_page_once`, `test_non_adjacent_ranges_sharing_a_page_still_rejected`, `_make_cache_with_row`)：新增的 CPU 单测，把线上 DCP 几何按 64 倍缩放 (`512/576 → 8/9`) 复现崩溃，并同时锁住“合并生效”和“有空隙仍拒绝”两个方向；`_RowCache` 用最小实现隔离 `free_kv_row`，避免拉起完整调度栈。

关键符号: `coalesce_ranges`, `free_kv_row`, `free_kv_row_segments`,
`_make_cache_with_row`, `test_adjacent_ranges_split_mid_page_free_every_page_once`,
`test_non_adjacent_ranges_sharing_a_page_still_rejected`

关键源码片段

`python/sglang/srt/mem_cache/base_prefix_cache.py`

修复的实际接线点: `BasePrefixCache.free_kv_row` 是先合并区间再构造释放段的地方, 也是所有前缀缓存共享的 KV 归还入口, 决定了修复的覆盖面与不变量边界。

```
# python/sglang/srt/mem_cache/base_prefix_cache.py (BasePrefixCache)
def free_kv_row(self, kv: Any, ranges: list[tuple[int, int]]) -> None:
    """Give back ascending, disjoint, half-open row-position ranges
    of the ``kv`` record's row; one call keeps a shared page freed once.
    """

    # 函数内 import, 避免 base_prefix_cache 在模块加载期依赖 mem_cache.common 的运行时符号
    from sglang.srt.mem_cache.common import coalesce_ranges, free_kv_row_segments

    row = self.req_to_token_pool.req_to_token[kv.req_pool_idx]
    # 相邻片段如果在同一个 (被 dcp_size 加宽过的) 页内交接, 这一页会被释放两次;
    # allocator 会直接断言失败, 所以先合并成一个区间再交给释放逻辑。
    free_kv_row_segments(
        self.token_to_kv_pool_allocator,
        # 合并后的区间用自己的 start 重新切片, 仍携带起始位置,
        # 供 free_kv_row_segments 按 swa_evicted_seqlen 把每段切成“已死 / 存活”两半。
        [(row[start:end], start) for start, end in coalesce_ranges(ranges)],
        swa_evicted_seqlen=kv.swa_evicted_seqlen,
    )
```

`python/sglang/srt/mem_cache/common.py`

新增核心工具函数 `coalesce_ranges`, 是修复的语义核心: 只合并严格相邻的半开区间, 空档保留, 从而在修掉崩溃的同时不削弱 allocator 的 double-free 断言; 该文件同时定义了 `free_kv_row_segments` 的 SWA 水位线切分与页对齐断言。

```
# python/sglang/srt/mem_cache/common.py
def coalesce_ranges(ranges: list[tuple[int, int]]) -> list[tuple[int, int]]:
    """Merge adjacent half-open ranges so a split that falls mid-page frees that page once."""
    merged: list[tuple[int, int]] = []
    # 单次线性扫描; 依赖调用方保证 ranges 升序且互不相交 (见 free_kv_row 的 docstring)。
    for start, end in ranges:
        # 只有「严格相接」才合并: start 恰等于上一段的 end, 说明两段之间的接缝
        # 落在页内部, 分开释放会把同一个 (DCP 加宽后的) 页释放两次。
        if merged and start == merged[-1][1]:
            merged[-1] = (merged[-1][0], end)
        else:
            # 中间存在空隙时保持原样: 此时共享页确实可能被释放两次,
            # 交给 allocator 的页不相交断言继续拦截, 而不是在这里静默吞掉。
            merged.append((start, end))
```

```
return merged
```

test/registered/unit/mem_cache/test_free_kv_row_coalesce.py

新增的 CPU 单测，把线上 DCP 几何按 64 倍缩放（512/576 → 8/9）复现崩溃，并同时锁住“合并生效”和“有空隙仍拒绝”两个方向；_RowCache 用最小实现隔离 free_kv_row，避免拉起完整调度栈。

```
# test/registered/unit/mem_cache/test_free_kv_row_coalesce.py
# 线上几何：64 token 页被 dcp_size 8 加宽成 512；这里按 64 倍缩小 —— page 8,
# Mamba track boundary 576 落到位置 9。
PAGE_SIZE = 8
NUM_PAGES = 16
```

```
def _make_cache_with_row(num_tokens):
    # 只搭最小的分页 allocator + 伪造的 req_to_token 行，不拉起完整调度栈
    alloc = PagedTokenToKVPoolAllocator(
        size=NUM_PAGES * PAGE_SIZE,
        page_size=PAGE_SIZE,
        dtype=torch.float16,
        device="cpu",
        kvcache=None,
        need_sort=False,
    )
    row = alloc.alloc(num_tokens)
    req_to_token = torch.zeros((1, num_tokens), dtype=row.dtype)
    req_to_token[0] = row
    kv = SimpleNamespace(req_pool_idx=0, swa_evicted_seqlen=0)
    return _RowCache(alloc, req_to_token), alloc, row, kv
```

```
class TestFreeKvRowCoalesce(unittest.TestCase):
    def test_adjacent_ranges_split_mid_page_free_every_page_once(self):
        cache, alloc, row, kv = _make_cache_with_row(3 * PAGE_SIZE)
        before = len(alloc.free_pages)
        # 保留前缀 [0, 8)；页内未对齐尾段 [8, 9) 与延迟截断尾段 [9, 24) 在 9 处相接，
        # 9 落在 page 1 内部 —— 修复前这里会抛共享页断言。
        cache.free_kv_row(kv, [(8, 9), (9, 3 * PAGE_SIZE)])
        freed = alloc.free_pages[: len(alloc.free_pages) - before]
        # 每个页必须且只能被释放一次：[8, 24) 恰好整页覆盖 page 1 与 page 2
        reference = torch.unique(row[8:] // PAGE_SIZE)
        self.assertTrue(torch.equal(torch.sort(freed)[0], reference))

    def test_non_adjacent_ranges_sharing_a_page_still_rejected(self):
        cache, _, _, kv = _make_cache_with_row(3 * PAGE_SIZE)
        # 两段之间留有缺口（[8, 9) 与 [12, 24)），共享页是真的会被释放两次，
        # 断言必须保留 —— 这是本 PR 刻意不放松的安全网。
        with self.assertRaises(AssertionError):
            cache.free_kv_row(kv, [(8, 9), (12, 3 * PAGE_SIZE)])
```

评论区精华

评审非常轻：唯一的 review 是 ispobock 的空正文 **APPROVED**，issue 下唯一评论是 **ispobock: cc @hnyls2002**，没有任何针对实现细节的追问；实质性分析全部由作者写在 PR body 里，包括根因链和“去掉断言会更糟”的判断。合并由 ispobock 执行，PR 带 **bypass-fastfail**、**run-ci-extra** 标签。PR body 的 CI 状态里“Latest PR Test (AMD ROCm 10)”为红 (Run #34733362396)，讨论中没有解释或跟进。

- DCP 加宽页与未加宽 cap 网格的接缝导致 double free (correctness): 在 `free_kv_row` 中用 `coalesce_ranges` 先把相邻区间并成一段再交给 allocator，每个页只释放一次，崩溃消失；作者以 Kimi-K3 + `--dcp-size 8` 的 210 个 prefill batch 实测无断言作为验证。
- 合并相邻区间是否削弱 allocator 的 double-free 保护 (design): 安全网保留，非相邻共页仍然报错；合并范围被限制在可证明等价（语义连续、页覆盖不变）的情形。
- 评审强度与 CI 覆盖是否足够 (testing): 合入依据主要是作者自述的 Kimi-K3 实测与新增 CPU 单测，第三方复核有限；AMD ROCm 10 失败原因未在讨论中澄清，属于遗留的可见性缺口。

风险与影响

- 风险：
 1. 覆盖范围仅在 `BasePrefixCache.free_kv_row`：若存在其它直接调用 `free_kv_row_segments` 的路径，或子类重写 `free_kv_row` 后自行构造段列表，传入相邻共页区间仍会命中同一断言。材料未提供完整调用图，建议以 `grep` 确认所有调用点是否都经过合并。
 2. `coalesce_ranges` 的前置条件未做校验：它只按 `start == merged[-1][1]` 合并，隐含依赖输入升序且互不相交（该保证只写在 `free_kv_row` 的 docstring 里）。若未来调用方传入乱序区间，函数会静默退化为不合并（回到崩溃行为）；传入重叠区间则会原样追加，可能产生重叠段。建议补断言或排序兜底。
 3. 合并在页覆盖上的等价性边界：用例覆盖的是“保留前缀正好在页边界结束（ $[0, 8)$ 之后接 $[8, 9) + [9, 24)$ ”的情形，合并后的并集恰好整页覆盖。若接缝两侧合起来并不覆盖整页（例如保留前缀结束在页内），释放语义是否仍与两次独立释放等价，缺少测试覆盖，需结合 allocator `free_full_segments` 的页粒度语义确认。
 4. 根因仍在，只是被绕开：DCP 下“页按 512 对齐、cache-length cap 按 64 网格”的不一致没有消除，`free_kv_row_segments` 中 `swa_evicted_seqlen % allocator.page_size == 0` 这类基于页宽的断言在相近配置（SWA 水位线落在页内）下仍可能被触发。
 5. 回归面较小：非 DCP 或只有单个区间的场景下 `coalesce_ranges` 是恒等变换，行为变化只发生在“相邻且接缝在页内”的情形，而该情形修复前是必崩，因此引入新回归的风险低。
 6. CI 信号：AMD ROCm 10 一项为红且讨论无结论；PR 依赖 `bypass-fastfail` 由维护者人工放行合入。
 - 影响：用户侧：使用 Kimi-K3 / Kimi-Linear 等 hybrid Mamba + MLA 模型并开启 `--dcp-size 8`（叠加 radix cache、HiCache、投机解码）的部署，从“正常负载下一分钟内整机崩溃、所有 scheduler 同时断言”恢复为可稳定服务，属于线上可用性级别的修复。系统侧：改动落在所有前缀缓存共享的 KV 归还入口

BasePrefixCache.free_kv_row, 处于内存释放核心路径, 但只在区间相邻时改变行为, 非 DCP 场景等价于无变化, 页级释放的并发与锁语义未触及。团队侧: 新增的 CPU 单测给出了一套把线上 DCP 页宽问题按 64 倍缩放到纯 CPU 复现的模板, 后续同类“加宽页 vs 未加宽网格”问题可直接复用; 同时暴露出 DCP 页几何与其它仍按 64 网格计算的组件之间可能存在系统性不一致, 值得作为专项排查。- 风险标记: 核心释放路径变更, 仅覆盖 free_kv_row 入口, DCP 页宽根因未消除, coalesce_ranges 依赖升序前置条件, 缺少端到端 DCP 回归测试, AMD ROCm 10 CI 红

关联脉络

- PR #37425 HiCache: Add @rank_consensus to various functions: 同属沿 radix cache / HiCache 不变式用断言守边并纳入 CI 的路线: 该 PR 给 unified_radix_cache 的关键函数加一致性校验, 本 PR 则选择保留 allocator 的共享页断言、只在调用方消化可证明等价的合并, 二者是同一“断言式防线”思路的不同切面。
- PR #37506 [unified-memory] PD disaggregation for every unified pool shape: 共享 mem_cache/allocator 的页级寻址与释放路径 (unified_hybrid_swa、unified_sub_pool、kv_cache_configurator), 说明 allocator 的页几何与 free 语义是近期多特性交叉的高风险面, 本 PR 正是在同一层面修的释放侧不变量。
- PR #37418 [unified-memory] Enable prefill cuda-graph capture: 同样改动 mem_cache allocator / kv cache 配置 (unified_hybrid_swa、kv_cache_configurator、kv_cache_hook), 与本 PR 的页几何、分配器约束属于同一条内存池演进线, 可作为理解页宽假设的上下文。
- PR #38925 [HiCache] Keep the file backend temp file name within NAME_MAX: 同属“生产环境崩溃 / 写失败 → 定点小修 + 单测”的 KV cache 线修法: 改动同样集中在单个 mem_cache 文件并附带可复现测试, 可与本 PR 对照其风险定位与验证方式。