PR #29432 完整报告

sgl-project/sglang

Fix bookkeeping fields not encapsulated with real allocations in normal alloc, PD pre-alloc, DFlash and EAGLE

合并时间：2026-07-15 14:52

原文链接：http://prhub.com.cn/sgl-project/sglang/pull/29432

执行摘要

- 一句话：封装 KV 分配记账到分配函数
- 推荐动作：值得精读。此 PR 演示了如何在大型代码库中进行安全的大规模重构：使用机械验证确保纯移动正确，用等价性审查确认语义变更，通过多步提交渐进式推进。对于关注代码架构和重构技巧的工程师，这是很好的学习案例。

功能与动机

消除记账与分配分离导致的不一致风险。PR body 指出：'Encapsulate the req.kv.kv_allocated_len bookkeeping with the real allocation in every path'，这是 req_pool_idx / cache / owned-KV 解耦重构链的一部分，为后续全面 owned-KV 生命周期管理创造条件。

实现拆解

1. 移动 Triton 辅助函数：将 assign_req_to_token_pool 及其调度函数从 cache_locs.py 搬迁至 allocation.py，并添加 CPU 分支（字节级验证）。
2. 准备 decode.py：对 _pre_alloc 进行去 self 化重写，将 req.kv 的初始化与物理分配放在同一处。
3. 提取前分配函数：从 DecodePreallocQueue 中提取 alloc_for_decode_prealloc 和 alloc_for_decode_prealloc_hisparse，封装 HiSparse 路径的分配和记账。
4. 统一 spec 解码分配：将 EAGLE 和 DFlash 内联的分配逻辑重写为标准形式，提取公共 alloc_for_spec_decode，供 eagle_utils.py 和 dflash_info_v2.py 调用，并删除分散的各处 kv_allocated_len 手工更新。
5. 封装剩余记账：在 alloc_for_extend 和 alloc_for_decode 末尾添加 req.kv.kv_allocated_len 更新；从 release_kv_cache 中提取 _release_overallocated_kv_indices 辅助函数。

关键文件：

- python/sglang/srt/mem_cache/allocation.py（模块 内存缓存；类别 source；类型 core-logic；符号 assign_req_to_token_pool, assign_req_to_token_pool_func, alloc_for_spec_decode）：核心变更文件：新增 assign_req_to_token_pool、assign_req_to_token_pool_func（从 cache_locs.py 移入）、alloc_for_spec_decode；在

alloc_for_extend 和 alloc_for_decode 末尾添加记账封装。

- python/sglang/srt/disaggregation/decode.py（模块 分离解码；类别 source；类型 core-logic；符号 alloc_for_decode_prealloc, alloc_for_decode_prealloc_hisparse）：提取了 alloc_for_decode_prealloc 和 alloc_for_decode_prealloc_hisparse 两个前分配函数，将物理分配和 req.kv 初始化封装在一起。
- python/sglang/srt/mem_cache/common.py（模块 内存缓存；类别 source；类型 core-logic；符号 _release_overallocated_kv_indices）：提取了 _release_overallocated_kv_indices 函数，将超额 KV 释放逻辑从 release_kv_cache 中分离。
- python/sglang/srt/speculative/eagle_utils.py（模块 投机解码；类别 source；类型 dependency-wiring）：改为调用统一的 alloc_for_spec_decode，删除原有的内联分配逻辑和 ALLOC_EXTEND_FUNCS 字典。
- python/sglang/srt/speculative/dflash_info_v2.py（模块 投机解码；类别 source；类型 dependency-wiring）：同样改为调用 alloc_for_spec_decode，删除内联分配和记账更新。
- python/sglang/kernels/ops/speculative/cache_locs.py（模块 kernel 层；类别 infra；类型 infrastructure；符号 assign_req_to_token_pool, assign_req_to_token_pool_func）：移除了 assign_req_to_token_pool 和 assign_req_to_token_pool_func 的定义（搬迁至 allocation.py）。
- test/registered/unit/spec/test_decode_bookkeeping_ownership.py（模块 单元测试；类别 test；类型 test-coverage）：测试文件，验证解码记账的所有权逻辑，适应新的记账封装。
- test/registered/unit/mem_cache/test_unified_radix_cache_unittest.py（模块 单元测试；类别 test；类型 test-coverage）：调整测试以适应记账迁移。
- test/registered/unit/mem_cache/test_hisparse_allocator.py（模块 单元测试；类别 test；类型 test-coverage）：调整 HiSparse 分配器测试。

关键符号：alloc_for_extend, alloc_for_decode, alloc_for_spec_decode, alloc_for_decode_prealloc, alloc_for_decode_prealloc_hisparse, _release_overallocated_kv_indices, assign_req_to_token_pool_func, assign_req_to_token_pool

关键源码片段

python/sglang/srt/mem_cache/allocation.py

核心变更文件：新增 assign_req_to_token_pool、assign_req_to_token_pool_func（从 cache_locs.py 移入）、alloc_for_spec_decode；在 alloc_for_extend 和 alloc_for_decode 末尾添加记账封装。

```
# alloc_for_spec_decode: 合并 EAGLE 和 DFlash 的 spec 解码分配路径
# 接收所有必要参数，单点处理物理分配和 req_to_token 更新

from sglang.srt.managers.schedule_batch import ReqKvInfo

@triton.jit
```

```
def assign_req_to_token_pool(
    req_pool_indices, req_to_token,
    start_offset, end_offset, out_cache_loc,
    pool_len: tl.constexpr, bs_upper: tl.constexpr,
):
    BLOCK_SIZE: tl.constexpr = 32
    pid = tl.program_id(axis=0)
    # 计算每个请求的起始偏移
    length_offset = tl.arange(0, bs_upper)
    start = tl.load(start_offset + length_offset, mask=length_offset < pid, other=0)
    end = tl.load(end_offset + length_offset, mask=length_offset < pid, other=0)
    out_offset = tl.sum(end - start, axis=0)
    # 循环写入 token pool
    save_offset = tl.arange(0, BLOCK_SIZE) + tl.load(start_offset + pid)
    load_offset = tl.arange(0, BLOCK_SIZE)
    num_loop = tl.cdiv(tl.load(end_offset + pid) - tl.load(start_offset + pid), BLOCK_SIZE)
    for _ in range(num_loop):
        mask = save_offset < tl.load(end_offset + pid)
        data = tl.load(out_cache_loc + out_offset + load_offset, mask=mask)
        tl.store(req_to_token + tl.load(req_pool_indices + pid) * pool_len + save_offset, data, mask=
        mask)
        save_offset += BLOCK_SIZE
        load_offset += BLOCK_SIZE

def assign_req_to_token_pool_func(
    req_pool_indices, req_to_token, start_offset, end_offset,
    out_cache_loc, batch_size,
):
    """调度函数：根据设备类型分发到 CPU 或 GPU 核"""
    if _is_cpu:
        from sgl_kernel import assign_req_to_token_pool_cpu
        assign_req_to_token_pool_cpu(
            req_pool_indices, req_to_token, start_offset, end_offset,
            out_cache_loc, req_to_token.shape[1],
        )
        return
    assign_req_to_token_pool[(batch_size,)](
        req_pool_indices, req_to_token, start_offset, end_offset, out_cache_loc,
        req_to_token.shape[1], next_power_of_2(batch_size),
    )

def alloc_for_spec_decode(
    tree_cache, req_to_token_pool, *,
    reqs, req_pool_indices,
    cur_kv_lens, cur_kv_lens_cpu,
    nxt_kv_lens, nxt_kv_lens_cpu,
    num_needed_tokens, batch=None,
):
    """统一的 spec 解码分配：evict、alloc、assign_req_to_token、更新 kv_allocated_len"""
```

```
    if num_needed_tokens <= 0:
        return
    evict_from_tree_cache(tree_cache, num_needed_tokens * req_to_token_pool.page_size)
    last_loc = get_last_loc(
        req_to_token_pool.req_to_token, req_pool_indices, cur_kv_lens,
    )
    out_cache_loc = alloc_paged_token_slots_extend(
        tree_cache, cur_kv_lens, cur_kv_lens_cpu,
        nxt_kv_lens, nxt_kv_lens_cpu, last_loc, num_needed_tokens,
        req_pool_indices=req_pool_indices, batch=batch,
    )
    assign_req_to_token_pool_func(
        req_pool_indices, req_to_token_pool.req_to_token,
        cur_kv_lens, nxt_kv_lens, out_cache_loc, len(reqs),
    )
    # 更新每个请求的 kv_allocated_len 记账
    for i, req in enumerate(reqs):
        req.kv.kv_allocated_len = int(nxt_kv_lens_cpu[i])
    return out_cache_loc
```

python/sglang/srt/disaggregation/decode.py

提取了 alloc_for_decode_prealloc 和 alloc_for_decode_prealloc_hisparse 两个前分配函数，将物理分配和 req.kv 初始化封装在一起。

```
# alloc_for_decode_prealloc_hisparse: HiSparse 路径的前分配
# 封装了逻辑索引分配和 kv_allocated_len 同步

def alloc_for_decode_prealloc_hisparse(
    allocator: BaseTokenToKVPoolAllocator, *,
    req: Req, fill_len: int,
    uses_swa_tail: bool, swa_tail_len: int,
) -> torch.Tensor:
    """为请求预分配 HiSparse 逻辑索引，并同步 kv_allocated_len"""
    # 初始化或更新 kv_allocated_len
    if req.kv is None:
        req.kv = ReqKvInfo(kv_allocated_len=fill_len, swa_evicted_seqlen=0)
    else:
        req.kv.kv_allocated_len = fill_len

    device = allocator.device
    prefix_lens = torch.tensor([0], dtype=torch.int64, device=device)
    seq_lens = torch.tensor([fill_len], dtype=torch.int64, device=device)
    last_loc = torch.tensor([-1], dtype=torch.int64, device=device)

    if uses_swa_tail:
        kv_loc = allocator.alloc_extend_swa_tail(
            prefix_lens=prefix_lens, seq_lens=seq_lens, last_loc=last_loc,
            extend_num_tokens=fill_len, swa_tail_len=swa_tail_len,
        )
```

```
            req.swa_evicted_seqlen = fill_len - swa_tail_len
        else:
            kv_loc = allocator.alloc_logical_only(
                prefix_lens=prefix_lens, seq_lens=seq_lens, last_loc=last_loc,
                extend_num_tokens=fill_len,
            )
        return kv_loc

    def alloc_for_decode_prealloc(
        allocator: BaseTokenToKVPoolAllocator, *,
        req: Req, prefix_indices, prefix_len, total_prefix_len, fill_len,
    ) -> torch.Tensor:
        # 类似封装，处理正常路径分配和记账
        ...
```

评论区精华

Review 中 Gemini Code Assist 提出了两个潜在安全问题：

- alloc_for_spec_decode 中直接访问 req.kv.kv_allocated_len 但 req.kv 可能为 None，建议添加防御性初始化（未在最终代码中采纳）。
- alloc_for_decode_prealloc 中对 prefix_indices 切片时未检查是否为 None，建议添加条件检查（未采纳）。 此外，PR 作者在 CI triage 中严格区分了机械移动和语义变更，并通过字节级验证脚本确保了移动的正确性。
- alloc_for_spec_decode 中 req.kv 可能为 None 导致 AttributeError (correctness): 未采纳（最终代码中仍直接访问），可能因为调用路径保证 req.kv 已初始化。
- alloc_for_decode_prealloc 中 prefix_indices 切片前未检查 None (correctness): 未采纳，但原代码逻辑依赖 prefix_len > 0 时 prefix_indices 必然非空。
- CI 失败皆为基础设施抖动，非本 PR 回归 (other): 确认所有 CUDA 失败均为 flake，非 PR 引入。

风险与影响

- 风险：主要风险是回归：修改了多条分配路径的记账位置。但由于以下因素风险可控：
  1. 所有纯移动提交均通过字节级验证脚本确认；
  2. 非移动的语义变更经过等价性审查；
  3. 测试覆盖率（包括单元测试和 CI）覆盖了主要路径。 次要风险：DSV4 NPU 上 DFlash 路径行为改变（从不可能执行到使用预留路径），若该路径被意外激活可能触发新问题。 无性能和安全风险。
- 影响：对用户：无行为变化，完全透明。 对开发者：分配集中的记账逻辑更易维护，为后续 owned-KV 完整生命周期管理奠定基础。 对系统：重构后 allocation.py 成为 KV 分配的中心，spec_utils 等模块的依赖关系减少。 影响范围：中，涉及多个模块但不影响用户可见行为。
- 风险标记：核心路径变更，DSV4 NPU 死代码激活，机械验证覆盖，缺少防御性检查

关联脉络

- PR #29431 op40 (req.kv container infrastructure): 此 PR 基于 op40（#29431），op40 引入了 req.kv 容器，此 PR 在此基础上完成记账封装。