PR #37085 完整报告

sgl-project/sglang

[mem_cache] Settle extend `kv_committed_len` inside `alloc_for_extend`

合并时间：2026-08-30 14:14

原文链接：http://prhub.com.cn/sgl-project/sglang/pull/37085

执行摘要

- 一句话：将 kv_committed_len 写入收拢进 KV 分配函数
- 推荐动作：值得快速阅读，不必精读。重点关注两个 alloc 函数尾部循环的「分配即提交」记账模式，以及 _OWNER_SITES 用 mutation-count 强制审计写入点归属的测试设计——这是 sglang 保护核心记账不变量的一种轻量而有效的做法。对于想理解 owned-KV 解耦重构链的读者，可把它作为入口之一。

功能与动机

29429 提出「让 req.kv 的存在意味着 owned KV 资源已分配」，要求在真正分配 / 释放 KV 的位置分配、释放 req.kv，而不是在 req_pool_idx 处理处，并且从不构造零值占位 ReqKvInfo。此前 kv_committed_len 的写入点与真实分配点相隔几步：extend 侧在 prepare_for_extend 的请求循环里写 seq_len，decode 侧在 prepare_for_decode 里自增，而真正的 KV 分配发生在 alloc_for_extend / alloc_for_decode。PR body 直接点名该问题（TODO(th4)），本 PR 即把两端写入收拢进分配函数，消除记账与分配错位。

实现拆解

1. extend 侧写入点迁移：在 python/sglang/srt/managers/schedule_batch.py 的 ScheduleBatch.prepare_for_extend 中删除 TODO(th4) 注释与 req.kv.kv_committed_len = seq_len 赋值；在 python/sglang/srt/mem_cache/allocation.py 的 alloc_for_extend 尾部循环中，紧挨 kv_allocated_len = seq_len 同位新增 kv_committed_len = seq_len，让「已分配 / 已提交」两个水位在分配收尾时一次性落定。
2. decode 侧写入点迁移：prepare_for_decode 的请求循环中保留 req.decode_batch_idx += 1，删除 req.kv.kv_committed_len += 1；alloc_for_decode 尾部循环在 kv_allocated_len += token_per_req 后同位新增 kv_committed_len += token_per_req，保持 decode 每步「分配即提交」的记账顺序。

3. 测试配套：test/registered/unit/spec/test_decode_bookkeeping_ownership.py 的 _OWNER_SITES 将 kv_committed_len 的两条 owner 记录从 ScheduleBatch.prepare_for_extend / prepare_for_decode 迁移到 alloc_for_extend / alloc_for_decode；该表是 mutation-count 审计，任何写入点增删都会让测试失败，从而强制未来改动显式评审记账归属。
4. 不变部分：spec 路径在 prepare_for_decode 中提前走 spec_prepare_for_decode 返回，不经过 alloc_for_decode；spec-v2 的 kv_committed_len 仍由 SchedulerBatchResultProcessor._resolve_spec_v2_tokens 根据接受结果统一提交。每个 alloc 函数只有一个调用者，且旧写入点与新写入点之间无任何读取，故迁移顺序等价。

关键文件：

- python/sglang/srt/managers/schedule_batch.py（模块 调度批次；类别 source；类型 core-logic；符号 ScheduleBatch.prepare_for_extend, ScheduleBatch.prepare_for_decode）：非投机路径 KV 记账的旧写入点所在：删除 prepare_for_extend 与 prepare_for_decode 中的 kv_committed_len 写入及 TODO(th4) 注释，是本次迁移的源头侧。
- python/sglang/srt/mem_cache/allocation.py（模块 分配器；类别 source；类型 core-logic；符号 alloc_for_extend, alloc_for_decode）：真正的 KV 分配现场；在分配循环尾部与 kv_allocated_len 同位新增 kv_committed_len 写入，是本 PR 的核心落点。
- test/registered/unit/spec/test_decode_bookkeeping_ownership.py（模块 记账测试；类别 test；类型 test-coverage；符号 _OWNER_SITES）：用 mutation-count 审计所有 KV 记账写入点归属；本 PR 同步迁移 kv_committed_len 的 owner 记录，防止未来写入点漂移。

关键符号：alloc_for_extend, alloc_for_decode, ScheduleBatch.prepare_for_extend, ScheduleBatch.prepare_for_decode

关键源码片段

python/sglang/srt/mem_cache/allocation.py

真正的 KV 分配现场；在分配循环尾部与 kv_allocated_len 同位新增 kv_committed_len 写入，是本 PR 的核心落点。

```
# ------------------------------------------------------------------------
# alloc_for_extend 尾部：KV 页与 aux 长度分配完成后，
# 在同一循环中将"已分配"与"已提交"两个水位一次性落定。
# 此前 extend 侧的 kv_committed_len 写在 prepare_for_extend 中，
# 与真实分配现场相隔若干步骤；迁移后记账与分配同位，TODO(th4) 消除。
# ------------------------------------------------------------------------
for req, seq_len in zip(batch.reqs, batch.seq_lens_cpu.tolist()):
    req.kv.kv_allocated_len = seq_len
    req.kv.kv_committed_len = seq_len

# alloc_for_decode 尾部：每轮 decode 分配 1 个 token 的 KV 空间，
# 非投机路径下"分配即提交"，因此 kv_committed_len 与
# kv_allocated_len 同步推进，保持与原来 prepare_for_decode 中的
# 写入顺序等价。
```

```
for req in batch.reqs:
    req.kv.kv_allocated_len += token_per_req
    req.kv.kv_committed_len += token_per_req
```

评论区精华

本 PR 没有 review 评论，唯一的互动是作者触发的 /tag-and-rerun-ci（补跑 CI），无技术讨论。核心论证来自 PR body，值得记录：

> Each alloc function has exactly one caller and nothing reads kv_committed_len between the old and new write points, so both moves are order-equivalent; the spec-v2 commit stays in _resolve_spec_v2_tokens where acceptance is decided.

该论证是本次重构安全性的主要依据：等价性不依赖运行期断言，而是依赖「两个写点之间无读取 + 单调用者」的结构性事实。

- 暂无高价值评论线程

风险与影响

- 风险：
  - 顺序等价性依赖隐含前提：等价性建立于「prepare 函数与 alloc 函数之间无人读取 kv_committed_len」这一现状。未来若有人在 prepare_for_extend 与 alloc_for_extend 之间、或 prepare_for_decode 与 alloc_for_decode 之间插入对该字段的读取，将读到上一轮的旧值；_OWNER_SITES 审计只约束写入点，不约束读取点。
  - alloc 内部异常路径：alloc_aux_to_lengths 失败时会先释放 out_cache_loc 再 raise，此时 kv_committed_len 不会落定；由于旧写入点也在分配之后，行为与迁移前一致，无回归。
  - 测试覆盖方式：test_decode_bookkeeping_ownership.py 是静态写入点计数清单，并不直接断言运行期数值或行为等价，因此本 PR 的「无行为变化」主要靠代码审查与顺序论证保证，而非动态验证。
  - 影响范围：仅涉及非投机（non-spec）路径；spec、disaggregation decode prealloc、beam 等路径的记账 owner 均未变化。
  - 影响：对用户无感知，系统行为不变，属于纯内部重构。对团队而言，这是 #29429 所代表的「req.kv 生命周期与真实 KV 分配位置对齐」重构链的具体落地：从此 [protected, committed, allocated] 这条水位阶梯的 committed 与 allocated 两端在分配现场同步维护，降低后续重构中记账与分配不一致的风险，也为 alloc_for_spec_decode、disaggregation decode prealloc 等已有同款模式提供了统一范例。影响程度低，但代码整洁度与可维护性有提升。
  - 风险标记：核心路径变更，顺序等价性假设，仅静态审计覆盖

关联脉络

- PR #37078 [mem_cache] Move kv_committed_len into ReqKvInfo: 本 PR 直接 stacked 在其上：先由 #37078 将 kv_committed_len 字段收进 ReqKvInfo，本 PR 再将其写入点 settle 进 alloc 函数，解决其留下的 TODO(th4)。

- PR #37094 [mem_cache] Move req_pool_idx into ReqKvInfo: 同一条 req.kv 生命周期解耦重构链中的相邻步骤，均为 ReqKvInfo 记账字段与真实分配位置对齐。
- PR #37108 [mem_cache] Share one ReqKvInfo between a streaming session slot and its request: 同一重构方向：精简 ReqKvInfo 与请求 / 会话槽之间的所有权关系，与本 PR 的记账落定互为上下文。
- PR #37164 [mem_cache] Move mamba state and retraction_backup into ReqKvInfo: 继续把更多每请求 KV 相关状态收进 ReqKvInfo，属于同一系列收拢动作。
- PR #35245 refactor(unified-memory): translate the KV write location once, at ForwardBatch construction: 同一 unified-memory 重构脉络，强调 KV 相关翻译 / 记账只在单一位置执行，与本 PR 的记账落定思路一致。