PR #34053 完整报告

sgl-project/sglang

[Fix] Account resident weight memory in KV sizing

合并时间：2026-08-27 17:43

原文链接：http://prhub.com.cn/sgl-project/sglang/pull/34053

执行摘要

- 一句话：修正权重缓存驻留内存导致 KV 过度分配、启动 OOM
- 推荐动作：值得精读。重点学习两个设计决策：其一，rank-local 校正必须先于分布式 MIN，因为不同 rank 的权重驻留量不同，MIN 之后再补偿会算错；其二，用“loader 字段默认 0 + no-op 校正”替代显式 daemon 模式判断，使普通启动零开销、未来任何预驻留权重的 loader 都能复用同一钩子。此外，review 中“去掉额外本地快照与第二次 MIN，直接加回已有基线”的演进展示了如何收敛过度防御的实现。若团队后续推广此机制，建议补充 daemon 集成测试与 KV sizing 单测。

功能与动机

PR body 明确指出：IPC weight-cache daemon 在引擎启动前就让权重常驻显存，ModelRunner 在权重已占用显存之后才采样 pre-load 空闲内存基线；不校正该基线时，KV sizing 只基于剩余空闲内存预留 headroom，会过度分配 KV cache。实测中两个客户端依次连接同一 daemon 时，无修复版本两次都在 decode CUDA graph 捕获阶段 CUDA OOM，而 daemon 仍存活。校正必须在分布式 MIN 之前按 rank 本地完成，因为各 rank 的权重驻留量不同，MIN 之后再补偿会算错；同时 target KV sizing 发生在 draft 加载之后，目标、多 runner 与未来 speculative draft 权重必须先聚合再参与 sizing。

实现拆解

实现拆解如下：

1. daemon 侧测量与协议返回：python/sglang/srt/weight_cache/daemon.py 的 DaemonState.load() 在加载权重前先 current_platform.empty_cache() 并记录 torch.cuda.memory_reserved(self.gpu_id)，加载完成并 synchronize() 后再 empty_cache()，以 max(0, 加载后 reserved - 加载前 reserved) 作为进程内 PyTorch allocator 的净增长，存入 DaemonState.preloaded_weights_bytes；_handle_connection 在响应中随 entries 一并返回该字段。这样客户端无需自行猜测权重占用。
2. loader 契约与客户端读取：python/sglang/srt/model_loader/loader.py 的 BaseModelLoader 新增默认字段 preloaded_weights_bytes = 0；python/sglang/srt/weight_cache/ipc_loader.py 的 IpcModelLoader.load_model() 每次开始时把该字段重置为 0，从 cache_data.get("preloaded_weights_bytes", 0) 读取（旧 daemon 缺失该字段时按 0 处理，保持兼容），并显式校验返回值必须是非 bool、非负 int

，非法时抛 RuntimeError 而不是静默接受脏元数据；成功映射权重后写入 self.preloaded_weights_bytes。

3. rank-local 校正：python/sglang/srt/model_executor/model_runner.py 新增 preloaded_weights_bytes property（从 loader 取值并校验非负 int，非法抛 ValueError）与 account_preloaded_weights() 方法（字节数换算成 GiB 后加回 pre_model_load_memory，0 字节时直接 no-op）。这保证校正发生在分布式 MIN 之前的本地 rank，且初始 sizing 与 capture 后 resizing 共用同一个校正后的基线。
4. 跨 worker 聚合：python/sglang/srt/managers/tp_worker.py 的 TpModelWorker 与 python/sglang/srt/speculative/base_spec_worker.py 分别新增 preloaded_weights_bytes property，汇总 model_runner_list 与 draft_runners 的字节数；python/sglang/srt/managers/scheduler.py 的 init_target_memory_pool() 在 KV 池分配前把 target 与 draft 的预驻留字节相加后调用 account_preloaded_weights()。代码注释明确承认"daemon + speculative draft"组合仍被显式拒绝（draft daemon 支持未实现），但核算路径已为其就绪。
5. 配套改动：python/sglang/srt/weight_cache/transport.py 把该字段接入可插拔 transport 协议（merge main 时适配了新的 transport 结构）；test/registered/unit/model_loader/test_weight_cache_protocol.py 补充了字段相关的协议断言。整体为纯增量改动（+72/-0），普通非 daemon 启动路径因 loader 上报 0 字节而完全不受影响。

关键文件：

- python/sglang/srt/model_executor/model_runner.py（模块 内存核算；类别 source；类型 data-contract；符号 preloaded_weights_bytes, account_preloaded_weights）：核心核算入口：新增 preloaded_weights_bytes property（校验 loader 上报值）与 account_preloaded_weights 方法（把驻留字节加回 pre_model_load_memory），确保校正发生在分布式 MIN 之前的本地 rank，且初始 sizing 与 resizing 共用同一基线。
- python/sglang/srt/managers/scheduler.py（模块 调度器；类别 source；类型 core-logic；符号 init_target_memory_pool）：聚合校正的调用点：init_target_memory_pool 在 KV 池分配前把 target 与 draft worker 的预驻留字节相加并调用 account_preloaded_weights，是修复生效的关键编排。
- python/sglang/srt/weight_cache/daemon.py（模块 权重缓存；类别 source；类型 core-logic；符号 DaemonState.load, DaemonState._handle_connection）：数据来源：DaemonState 测量进程内 PyTorch allocator 在权重加载前后的净增长，并通过 IPC 响应的 preloaded_weights_bytes 字段返回给客户端。
- python/sglang/srt/weight_cache/ipc_loader.py（模块 权重缓存；类别 source；类型 core-logic；符号 IpcModelLoader.load_model）：客户端读取与校验：load_model 从 daemon 响应读取 preloaded_weights_bytes，向后兼容旧 daemon（缺失按 0），并校验类型合法性后写入 loader 字段供 ModelRunner 使用。
- python/sglang/srt/speculative/base_spec_worker.py（模块 投机解码；类别 source；类型 core-logic；符号 preloaded_weights_bytes）：draft 侧聚合：为未来 speculative draft daemon 支持预留核算路径，将 draft_runners 的预驻留字节汇总；当前 daemon + draft 组合仍被显式拒绝。

- python/sglang/srt/managers/tp_worker.py（模块 TP 执行；类别 source；类型 core-logic；符号 preloaded_weights_bytes）：target 侧聚合：TpModelWorker 将 model_runner_list（支持 multi-runner）的预驻留字节汇总，供 scheduler 统一调用。
- python/sglang/srt/model_loader/loader.py（模块 模型加载；类别 source；类型 data-contract）：契约定义：BaseModelLoader 新增默认字段 preloaded_weights_bytes = 0，使 ModelRunner 无需 getattr 即可直接访问，并让所有 loader 默认具备该核算钩子。
- python/sglang/srt/weight_cache/transport.py（模块 权重缓存；类别 source；类型 core-logic）：协议适配：merge main 时把 preloaded_weights_bytes 从原始 dict 接入新的可插拔 weight-cache transport，保证字段在协议层正确传递。
- test/registered/unit/model_loader/test_weight_cache_protocol.py（模块 协议测试；类别 test；类型 test-coverage）：唯一测试配套，补充协议字段断言；覆盖仍较薄，未包含端到端 daemon 集成测试。

关键符号：ModelRunner.preloaded_weights_bytes, ModelRunner.account_preloaded_weights, Scheduler.init_target_memory_pool, TpModelWorker.preloaded_weights_bytes, BaseSpecWorker.preloaded_weights_bytes, IpcModelLoader.load_model, DaemonState.load

关键源码片段

python/sglang/srt/model_executor/model_runner.py

核心核算入口：新增 preloaded_weights_bytes property（校验 loader 上报值）与 account_preloaded_weights 方法（把驻留字节加回 pre_model_load_memory），确保校正发生在分布式 MIN 之前的本地 rank，且初始 sizing 与 resizing 共用同一基线。

```
class ModelRunner:
    # ...（类头与初始化省略）

    @property
    def preloaded_weights_bytes(self) -> int:
        # 统一从 loader 读取预驻留权重字节数并校验合法性：
        # 拒绝 bool、非 int、负数，避免脏元数据污染 KV 容量计算
        value = self.loader.preloaded_weights_bytes
        if isinstance(value, bool) or not isinstance(value, int) or value < 0:
            raise ValueError(
                "ModelLoader.preloaded_weights_bytes must be a non-negative int, "
                f"got {value!r}"
            )
        return value

    def account_preloaded_weights(self, preloaded_weights_bytes: int) -> None:
        # dist-init 采样的 pre_model_load_memory 发生在 daemon 已驻留权重之后，
        # 导致 KV slack（B * (1 - mem_fraction_static)）偏小、KV 分配偏大；
        # 把驻留字节数加回已经过分布式 MIN 的基线上即可恢复正确 slack。
        # loader 上报 0 时直接跳过，普通启动不会多付出一次规约开销。
        if preloaded_weights_bytes == 0:
            return
```

```
        self.pre_model_load_memory += preloaded_weights_bytes / (1 << 30)

    def alloc_memory_pool(self, memory_pool_config: Optional[MemoryPoolConfig] = None):
        """只分配 KV cache 内存池（不初始化后端与 CUDA graph）。"""
        if memory_pool_config is not None:
            self.memory_pool_config = memory_pool_config

        self.init_kv_cache_configurator()
        # 初始 sizing 与 graph capture 后的 resizing 均使用校正后的基线
        result = self.kv_cache_configurator.configure(
            pre_model_load_memory=self.pre_model_load_memory
        )
        self.max_total_num_tokens = result.max_total_num_tokens
        self.max_running_requests = result.max_running_requests
        self.req_to_token_pool = result.req_to_token_pool
        self.token_to_kv_pool = result.token_to_kv_pool
        self.token_to_kv_pool_allocator = result.token_to_kv_pool_allocator
        self.memory_pool_config = result.memory_pool_config
        if self.is_hybrid_swa:
            self.full_max_total_num_tokens = result.full_max_total_num_tokens
            self.swa_max_total_num_tokens = result.swa_max_total_num_tokens
        # 保留引用，防止共享 byte buffer 被 GC 回收
        self._unified_memory_pool = result.unified_memory_pool

        self._init_post_memory_pool_components()
```

python/sglang/srt/managers/scheduler.py

聚合校正的调用点：init_target_memory_pool 在 KV 池分配前把 target 与 draft worker 的预驻留字节相加并调用 account_preloaded_weights，是修复生效的关键编排。

```
def init_target_memory_pool(self):
    # 已分配过 KV 池则直接返回，避免重复校正导致基线被加倍
    if (
        self.tp_worker.model_runner.token_to_kv_pool is not None
        and self.tp_worker.model_runner.token_to_kv_pool_allocator is not None
    ):
        return

    # 聚合 target（含 multi-runner）与 speculative draft worker 的预驻留权重字节数；
    # 必须在分布式 MIN 之前完成 rank-local 校正：
    # 各 rank 驻留量不同，MIN 之后再补偿会算错
    preloaded_weights_bytes = self.tp_worker.preloaded_weights_bytes
    if self.draft_worker is not None:
        preloaded_weights_bytes += self.draft_worker.preloaded_weights_bytes

    # 立即在 KV 池分配前把字节加回 pre_model_load_memory，
    # 使初始 sizing 与 capture 后 resizing 使用同一校正值
    self.tp_worker.model_runner.account_preloaded_weights(preloaded_weights_bytes)
    self.tp_worker.alloc_memory_pool()
```

python/sglang/srt/weight_cache/daemon.py

数据来源：DaemonState 测量进程内 PyTorch allocator 在权重加载前后的净增长，并通过 IPC 响应的 preloaded_weights_bytes 字段返回给客户端。

```
def load(self):
    # … 前置配置与 fingerprint 计算省略 …

    # 先清空缓存再采样，保证 memory_reserved 增量只反映权重加载本身，
    # 避免 caching allocator 的残留分配污染测量
    current_platform.empty_cache()
    memory_before_load = torch.cuda.memory_reserved(self.gpu_id)

    # … 实际加载权重并做 quant 预处理 …

    # 同步后再清一次缓存，随后计算进程内 allocator 的净增长；
    # max(0, …) 防止异常路径下出现负值
    current_platform.synchronize()
    current_platform.empty_cache()
    self.preloaded_weights_bytes = max(
        0, torch.cuda.memory_reserved(self.gpu_id) - memory_before_load
    )

    # 把全部权重导出为 IPC handle 供客户端零拷贝映射
    self._export_state()
```

评论区精华

review 核心讨论如下：

- alexnails（COMMENTED）提出两点：一是该校正只在 weight daemon 启用时需要；二是 reduce_min_gpu_memory 及其结果 local_prep_model_load 或许应放进 global_ctx 或并入 available_gpu_memory。作者 galletas1712 回应：account_preloaded_weights 在 loader 上报 0 字节时是 no-op，普通启动不会多做一次规约；保留为 loader 字段默认 0 而非显式 --weight-cache-mode 判断，是为了让任何“映射已驻留权重”的 loader 都能复用同一钩子。同时作者承认原实现过度防御——daemon 只破坏 pre_model_load_memory 的首次测量，因此删掉了额外的本地快照和第二次 MIN，改为直接把字节加回已有 MIN 后的基线上（对应 commit 86b751c）。
- liusy58（APPROVED）在 code review 中建议避免 getattr 访问 loader 字段，作者确认修复，最终提交 e7ac84d 改为 BaseModelLoader 直接声明字段、ModelRunner 直接访问。
- 其余审核状态：liusy58 最终 APPROVED，PR 由 ch-wan 合并。
- 校正范围与实现位置：仅 daemon 需要？local_prep_model_load 归属？ (design): 采用“loader 字段默认 0 + no-op 校正”的通用契约；去掉额外快照与第二次 MIN，收敛为对现有基线的增量补偿。
- 避免 getattr 访问 loader 字段 (style): 改为在 BaseModelLoader 上显式声明 preloaded_weights_bytes 字段，ModelRunner 直接访问，消除了 getattr 兜底。

风险与影响

- 风险：技术风险如下：
- 测量依赖 allocator 行为：daemon.py 用 torch.cuda.memory_reserved 增量估算权重驻留量，虽在测量前后各做一次 empty_cache() 降低缓存分配噪声，但显存碎片化可能让增量略大于真实权重；若 daemon 与客户端 PyTorch 版本或 allocator 行为差异较大，测量值可能偏离实际驻留量。
- 新增 loader 数据契约：ModelRunner.preloaded_weights_bytes 对 loader 上报值做硬校验，任何第三方自定义 loader 若未实现该字段或返回非法值，会在启动时抛 ValueError——这是有意的契约强制，但属于新增的失败路径，需确保仓库内所有 loader 已实现默认 0 字段。
- 旧 daemon 兼容：ipc_loader.py 对缺失字段按 0 处理，行为降级为“不校正”，不会崩溃；但旧 daemon 下缺陷仍存在，属预期渐进式兼容。
- 双计数风险：scheduler.init_target_memory_pool() 有已分配判断避免重复校正；若未来放开 draft daemon 组合而实现不完整，聚合路径可能产生错误基线，当前通过显式拒绝规避。
- 测试覆盖偏薄：仅协议测试 +2 行断言，缺少 daemon 集成测试或 KV sizing 单测来锁定校正逻辑，回归主要依赖作者的人工 A/B 验证。
- 影响：对用户：使用 weight-cache daemon 且模型权重大于一半显存的场景（如 80 GB 卡跑 Qwen2.5-32B BF16）从确定性启动 OOM 变为可正常启动；修复后 KV token 数从 63,434 降为 38,430，说明 KV 容量核算更保守且正确。对系统：启动路径新增两次进程内 allocator 读取和一次已有 world-group 规约，稳态推理路径完全不变；普通（非 daemon）启动因 loader 上报 0 字节而零额外开销。对团队：引入了通用的 preloaded_weights_bytes loader 契约，为未来任何“预驻留权重”的加载器（含 speculative draft daemon）提供了统一的核算入口，但也要求后续 loader 实现遵守该契约。
- 风险标记：权重缓存场景专用修复，依赖 allocator 内存测量，新增 loader 数据契约，测试覆盖较薄

关联脉络

- PR #36343 [AMD] Fall back to CPU tensor for decode retraction on ROCm: 同为 KV cache 内存管理缺陷修复（kv_cache_builder 的显存核算与后备路径），与本次 KV sizing 校正同属显存核算类问题，但机制不同（disaggregation 场景）。