PR #30547 完整报告

sgl-project/sglang

[MLX] Honor --max-running-requests in the model runner stub

合并时间：2026-07-17 23:24

原文链接：http://prhub.com.cn/sgl-project/sglang/pull/30547

执行摘要

- 一句话：修复 MLX 后端忽略 max-running-requests 问题
- 推荐动作：建议精读。该 PR 不仅修复了一个长期存在的参数失效 bug，还通过 review 迭代完善了辅助状态池的边界处理与释放所有权，体现了良好的设计权衡。测试完备，代码质量高。

功能与动机

PR body 指出：--max-running-requests is silently ignored on the MLX backend，MLX stub 硬编码了并发上限，导致用户设置该参数无效果。在混合注意力模型上还可能导致 auxiliary 状态槽不足的运行时断言错误。

实现拆解

1. 添加解析方法：在 model_runner_stub.py 中新增 _resolve_max_running_requests()，它先按 DP 拆分 --max-running-requests（若未设置则使用默认 min(capacity, 4096)），再按 KV 池容量 max_total_num_tokens // 2 上限截断，同时针对混合注意力模型（hybrid/linear-attention）根据 auxiliary 状态池容量和每请求槽位比率（_aux_state_slots_per_request()）进一步约束，若结果 <= 0 则抛错。新增 _aux_state_slots_per_request() 根据 disable_radix_cache 返回 1 或 4（对应 radix 快照预留）。修改 initialize() 调用新方法替代硬编码。
2. 辅助状态池释放修正：在 auxiliary_state.py 中，MlxAuxiliaryStateReqToTokenPool 新增 owns_auxiliary_state_release 参数，当为 True 时（非 radix 缓存路径），free(req) 会先调用 free_mamba_cache(req) 释放 auxiliary 槽，避免泄漏。修改了池构造时的相关逻辑。
3. 单元测试覆盖：新增 test_max_running_requests.py，使用 mock 方式创建 stub 并调用 _resolve_max_running_requests，覆盖了参数未设置、参数在容量内、DP 拆分、被容量上限截断、混合模型被 auxiliary 池约束、以及初始化和请求分配端到端场景。测试也在 MLX CI 套件注册。

关键文件：

- python/sglang/srt/hardware_backend/mlx/model_runner_stub.py（模块 MLX 后端；类别 source；类型 core-logic；符号 _resolve_max_running_requests, _aux_state_slots_per_request）：核心变更文件，新增 max_running_requests 解析逻辑

和 auxiliary 池边界的处理，修改初始化流程。

- test/registered/unit/hardware_backend/mlx/test_max_running_requests.py（模块 测试；类别 test；类型 test-coverage；符号 TestMlxMaxRunningRequests, _resolve, _stub, _hybrid_stub_for_initialize）：新增单元测试，全面覆盖解析方法的各种场景，验证回归修复和辅助状态边界。
- python/sglang/srt/hardware_backend/mlx/kv_cache/auxiliary_state.py（模块 缓存层；类别 source；类型 core-logic；符号 MlxAuxiliaryStateReqToTokenPool.init, MlxAuxiliaryStateReqToTokenPool.free）：修改辅助状态池的释放逻辑，新增 owns_auxiliary_state_release 参数以防止槽位泄漏。

关键符号：_resolve_max_running_requests, _aux_state_slots_per_request, MlxAuxiliaryStateReqToTokenPool.free, MlxAuxiliaryStateReqToTokenPool.init

关键源码片段

python/sglang/srt/hardware_backend/mlx/model_runner_stub.py

核心变更文件，新增 max_running_requests 解析逻辑和 auxiliary 池边界的处理，修改初始化流程。

```
# MLX_AUX_STATE_SIZE_MAX_RUNNING_REQUESTS_RATIO = 4
# _aux_state_slots_per_request 根据 disable_radix_cache 返回 1 或 4

def _aux_state_slots_per_request(self) -> int:
    """返回每个运行请求占用的 auxiliary 槽位数。

    当 radix cache 禁用时，无快照预留，每个请求正好占 1 个槽；
    启用 radix cache 时预留 4 个槽给快照和 chunk track 缓冲区。
    """
    if self.server_args.disable_radix_cache:
        return 1
    return MLX_AUX_STATE_SIZE_MAX_RUNNING_REQUESTS_RATIO

def _resolve_max_running_requests(self) -> int:
    """基于参数和容量计算最终并发上限。

    考虑以下约束（按顺序应用）：
    1. 若 flag 未设置，则使用默认 min(capacity, 4096)；
    2. 若 flag 设置，按 DP worker 数拆分；
    3. 被 KV 池容量（max_total_num_tokens // 2）截断；
    4. 对混合注意力模型，根据 auxiliary 池容量和每请求槽位比率进一步约束；
    5. 若结果 <= 0 则抛出 RuntimeError。
    """
    capacity_cap = self.max_total_num_tokens // 2
    requested = self.server_args.max_running_requests
    if requested is None:
        requested_per_worker = None
        resolved = min(capacity_cap, 4096)
    else:
```

```
            requested_per_worker = requested // self.dp_size
            resolved = min(requested_per_worker, capacity_cap)

        # 混合模型约束
        aux_state_size = self.server_args.max_mamba_cache_size
        if mambaish_config(self.model_config) is not None and aux_state_size is not None:
            ratio = self._aux_state_slots_per_request()
            resolved = min(resolved, aux_state_size // ratio)
            if resolved <= 0:
                raise RuntimeError(
                    f"MLX auxiliary state cache too small: max_mamba_cache_size={aux_state_size}, "
                    f"slots_per_request={ratio}, cannot serve any request")
        return resolved
```

test/registered/unit/hardware_backend/mlx/test_max_running_requests.py

新增单元测试，全面覆盖解析方法的各种场景，验证回归修复和辅助状态边界。

```
# 辅助函数：创建 mock Stub 并调用解析方法

def _resolve(stub, hybrid=False):
    """在 mock 架构下运行解析器"""
    with _arch(hybrid):
        return stub._resolve_max_running_requests()

class TestMlxMaxRunningRequests(CustomTestCase):
    def test_flag_unset_uses_capacity_default(self):
        # 未设置 flag 时使用默认值 min(capacity, 4096)
        self.assertEqual(_resolve(_stub(None, 1000)), 500)
        self.assertEqual(_resolve(_stub(None, 100_000)), 4096)

    def test_flag_honored_within_capacity(self):
        # 核心回归场景：显式设置 flag 必须被尊重
        self.assertEqual(_resolve(_stub(1, 100_000)), 1)
        self.assertEqual(_resolve(_stub(64, 100_000)), 64)

    def test_flag_split_per_dp_worker(self):
        # DP 拆分：flag 值除以 worker 数
        self.assertEqual(_resolve(_stub(8, 100_000, dp_size=4)), 2)

    def test_flag_capped_by_pool_capacity(self):
        # 容量上限：flag 过大时被 pool 容量截断
        self.assertEqual(_resolve(_stub(1000, 100)), 50)

    def test_hybrid_bounded_by_aux_pool(self):
        # 混合模型：被 auxiliary 池限制
        self.assertEqual(_resolve(_stub(8, 1000, max_mamba_cache_size=4), hybrid=True), 1)
```

python/sglang/srt/hardware_backend/mlx/kv_cache/auxiliary_state.py

修改辅助状态池的释放逻辑，新增 owns_auxiliary_state_release 参数以防止槽位泄漏。

```
class MlxAuxiliaryStateReqToTokenPool(ReqToTokenPool):
    def __init__(self, *, ..., owns_auxiliary_state_release: bool = False):
        super().__init__(...)
        self._owns_auxiliary_state_release = owns_auxiliary_state_release
        ...

    def free(self, req):
        # 当池拥有释放所有权时（非 radix 缓存），先释放 auxiliary 槽
        if self._owns_auxiliary_state_release:
            # 调用 free_mamba_cache 释放槽位，注意防重入
            self.free_mamba_cache(req)
        super().free(req)
```

评论区精华

Review 中主要讨论了三点：

1. 混合模型 auxiliary 状态池容量约束：reviewer @yeahdongcn 指出初始实现未考虑 auxiliary 状态池容量，导致 max_running_requests=4 但 max_mamba_cache_size=2 时第三请求会因槽位不足而崩溃。作者随后添加了 auxiliary 池约束。
2. disable_radix_cache 时的槽位比率：reviewer 指出固定 4x 比率在禁用 radix 缓存时过于保守，因为无 radix 快照预留。作者添加了 _aux_state_slots_per_request() 根据 disable_radix_cache 返回 1 或 4。
3. 辅助状态释放所有权：reviewer 发现了在非 radix 缓存路径下，请求释放时 auxiliary 槽未被回收的缺陷。作者通过引入 owns_auxiliary_state_release 参数使池在释放时负责释放槽位。

- Hybrid 模型 auxiliary 状态池容量约束 (correctness)：已修复：添加了对 auxiliary 池容量的检查，并在容量不足时抛错。
- 禁用 Radix Cache 时 auxiliary 槽位比率 (design)：已修复：通过 _aux_state_slots_per_request 根据 disable_radix_cache 返回 1 或 4。
- Auxiliary 状态释放所有权 (correctness)：已修复：通过 owns_auxiliary_state_release 参数使池在释放时负责释放 auxiliary 槽。

风险与影响

- 风险：变更局限于 MLX 后端，不影响其他后端。主要风险是新增的辅助状态释放逻辑可能引入双重释放或仍存在泄漏，但已通过测试覆盖。_resolve_max_running_requests 中的容量约束和 DP 拆分逻辑与 CUDA 后端同步，若未来 CUDA 端逻辑更新而 MLX 未同步，可能产生偏差。当前代码复杂度增加（约 90 行源码 + 316 行测试），但逻辑可测试。
- 影响：用户影响：MLX 后端用户设置的 --max-running-requests 参数从被忽略变为正确生效；混合模型用户的 auxiliary 状态池瓶颈将在初始化时得到检查而非在运行时崩溃。系统影响：无性能影响（仅初始化时一次整数计算）。团队影响：增加了 MLX 后端的健壮性，减少线上配置误判。
- 风险标记：MLX 后端独占，辅助状态池所有权，测试覆盖完整

关联脉络

- 暂无明显关联 PR