PR #24726 完整报告

sgl-project/sglang

env: add SGLANG_RADIX_FORCE_MISS to force radix prefix-cache miss

合并时间：2026-05-09 08:46

原文链接：http://prhub.com.cn/sgl-project/sglang/pull/24726

执行摘要

- 一句话：新增环境变量强制 radix 前缀缓存 miss
- 推荐动作：值得精读，尤其是其“在调度边界而非缓存内部 gating”的设计决策，以及通过 MatchResult._replace 和 [:0] 切片保留 tensor 元数据的技巧。该 PR 展示了一个轻量但考虑周全的特性实现，适合作为基准测试基础设施的参考范例。

功能与动机

现有 --disable-radix-cache 会切换到 ChunkCache 代码路径，无法衡量真实 radix 调度路径下不含缓存命中的性能。PR 需要一种手段使缓存代码路径完全运行但始终返回 miss，以便隔离缓存命中带来的吞吐 / 延迟混淆因素。

实现拆解

1. 环境变量声明（python/sglang/srt/environ.py）：新增 SGLANG_RADIX_FORCE_MISS 为 EnvBool，默认 False，与其他 radix 相关环境变量并列。
2. 零匹配辅助函数（python/sglang/srt/mem_cache/base_prefix_cache.py）：新增 zero_match_result(tree_cache, match_result)，通过 MatchResult._replace 将 device_indices 切片为 [:0]（保留 dtype/device 但不分配新内存），last_device_node/last_host_node 指向缓存的 root_node，host_hit_length 置 0。若缓存无 root_node 属性（如实验性 RadixCacheCpp），则抛出 RuntimeError 避免静默泄漏命中。
3. 调度策略 gate（python/sglang/srt/managers/schedule_policy.py）：在 match_prefix_for_req 函数中，调完 tree_cache.match_prefix 后立即判断标志并调用 zero_match_result 覆盖结果；同样在 _compute_prefix_matches 中的等待队列前缀树查询上应用同一 gate，防止调度器基于缓存命中做等待队列优先级调整。
4. 调度批处理 gate（python/sglang/srt/managers/schedule_batch.py）：在 Req.init_next_round_input 的 match_prefix 调用后插入相同的门控逻辑，覆盖第三个外部调用点。
5. 单元测试（test/registered/unit/mem_cache/test_radix_force_miss.py）：CPU 隔离测试，验证 zero_match_result 正确清零、无 root_node 时抛异常、以及 match_prefix_for_req 在标志开启时强制 miss，关闭时正常命中。测试已注册到 CPU CI 套件。

关键文件：

- python/sglang/srt/mem_cache/base_prefix_cache.py（模块 缓存层；类别 source；类型 core-logic；符号 zero_match_result）：新增 zero_match_result 核心零匹配函数，是整个特性的逻辑基石。
- python/sglang/srt/managers/schedule_policy.py（模块 调度器；类别 source；类型 dependency-wiring；符号 match_prefix_for_req）：在 match_prefix_for_req 和等待队列前缀匹配两个调用点插入 gate，覆盖调度器核心路径。
- test/registered/unit/mem_cache/test_radix_force_miss.py（模块 测试；类别 test；类型 test-coverage；符号 _StubReq, TestZeroMatchResult, TestMatchPrefixForReqForceMiss）：CPU 单元测试，验证零匹配函数的正确性、异常行为及与调度策略的集成，已注册到 CI。
- python/sglang/srt/managers/schedule_batch.py（模块 调度器；类别 source；类型 dependency-wiring）：第三个外部 match_prefix 调用点，确保 init_next_round_input 中也应用 gate。
- python/sglang/srt/environ.py（模块 配置；类别 source；类型 core-logic）：环境变量声明入口，新增 SGLANG_RADIX_FORCE_MISS 标志。

关键符号：zero_match_result, match_prefix_for_req

关键源码片段

python/sglang/srt/mem_cache/base_prefix_cache.py

新增 zero_match_result 核心零匹配函数，是整个特性的逻辑基石。

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

def zero_match_result(tree_cache, match_result: "MatchResult") -> "MatchResult":
    # 安全获取 root_node，若不存在（如 RadixCacheCpp）则无法确定根节点来重置位置；
    # 必须主动报错，避免静默泄漏缓存命中。
    root = getattr(tree_cache, "root_node", None)
    if root is None:
        raise RuntimeError(
            f"SGLANG_RADIX_FORCE_MISS is not supported by {type(tree_cache).__name__} "
            "(no `root_node` attribute). Disable the flag or use a cache backend "
            "that exposes a tree root."
        )
    return match_result._replace(
        # [:0] 切片不分配新内存，但保留原始 tensor 的 dtype 和 device（例如 CUDA int64），
        # 避免创建空 tensor 时可能隐含的设备 / 类型不匹配。
        device_indices=match_result.device_indices[:0],
        last_device_node=root, # 将最后节点重置为根节点，表示无缓存命中
        last_host_node=root,
        host_hit_length=0, # 主机缓存未命中
    )
```

python/sglang/srt/managers/schedule_policy.py

在 match_prefix_for_req 和等待队列前缀匹配两个调用点插入 gate，覆盖调度器核心路径。

```
# python/sglang/srt/managers/schedule_policy.py (match_prefix_for_req 函数片段 )

def match_prefix_for_req(
    tree_cache: BasePrefixCache,
    req: Req,
    token_ids: Optional[List[int]] = None,
    *,
    cow_mamba: bool = False,
    include_req: bool = False,
):
    if token_ids is None:
        token_ids = req.origin_input_ids + req.output_ids

    match_result = tree_cache.match_prefix(
        MatchPrefixParams(
            key=RadixKey(token_ids=token_ids, extra_key=req.extra_key),
            cow_mamba=cow_mamba,
            req=req if include_req else None,
        )
    )
    # 【门控】仅在调度器侧清零，不影响 radix cache 内部插入后的重匹配合约
    if envs.SGLANG_RADIX_FORCE_MISS.get():
        match_result = zero_match_result(tree_cache, match_result)
    (
        req.prefix_indices,
        req.last_node,
        req.last_host_node,
        req.host_hit_length,
    ) = (
        match_result.device_indices,
        match_result.last_device_node,
        match_result.last_host_node,
        match_result.host_hit_length,
    )
    # ... 省略 mamba 和 cache_protected_len 处理
    return match_result
```

test/registered/unit/mem_cache/test_radix_force_miss.py

CPU 单元测试，验证零匹配函数的正确性、异常行为及与调度策略的集成，已注册到 CI。

```
# test/registered/unit/mem_cache/test_radix_force_miss.py

class TestZeroMatchResult(unittest.TestCase):
    def test_zero_replaces_indices_and_nodes(self):
        # 构建一个含数据的 RadixCache，插入 [1,2,3,4,5] 并用 [1,2,3,9] 查询，正常应命中前 3 个
        token
        tree = RadixCache.create_simulated()
        tree.insert(InsertParams(key=RadixKey(token_ids=[1, 2, 3, 4, 5])))
```

```
        match = tree.match_prefix(
            MatchPrefixParams(key=RadixKey(token_ids=[1, 2, 3, 9]))
        )
        self.assertGreater(len(match.device_indices), 0)
        # 调用 zero_match_result 后，所有命中信息被清零
        zeroed = zero_match_result(tree, match)
        self.assertEqual(int(zeroed.device_indices.numel()), 0)
        self.assertIs(zeroed.last_device_node, tree.root_node)
        self.assertIs(zeroed.last_host_node, tree.root_node)
        self.assertEqual(zeroed.host_hit_length, 0)
        # 验证 dtype 和 device 被保留（[:0] 切片方式）
        self.assertEqual(zeroed.device_indices.dtype, match.device_indices.dtype)
        self.assertEqual(zeroed.device_indices.device, match.device_indices.device)

    def test_no_root_node_raises(self):
        # 模拟无 root_node 的缓存对象，调用 zero_match_result 应抛 RuntimeError
        class _NoRoot:
            pass
        original = MatchResult(
            device_indices=torch.tensor([7, 8, 9], dtype=torch.int64),
            last_device_node="sentinel-device",
            last_host_node="sentinel-host",
            host_hit_length=4,
        )
        with self.assertRaisesRegex(RuntimeError, "SGLANG_RADIX_FORCE_MISS"):
            zero_match_result(_NoRoot(), original)

class TestMatchPrefixForReqForceMiss(unittest.TestCase):
    def test_force_miss_zeros_req_prefix(self):
        tree = RadixCache.create_simulated()
        tree.insert(
            InsertParams(key=RadixKey(token_ids=[10, 11, 12, 13, 14, 15, 16, 17]))
        )
        # 基准：未开启标志时，match_prefix_for_req 正常命中
        baseline_req = _StubReq([10, 11, 12, 13, 99, 100])
        with envs.SGLANG_RADIX_FORCE_MISS.override(False):
            match_prefix_for_req(tree, baseline_req)
        self.assertGreater(int(baseline_req.prefix_indices.numel()), 0)
        self.assertIsNot(baseline_req.last_node, tree.root_node)
        # 测试：开启标志后，相同请求的命中被强制清零
        forced_req = _StubReq([10, 11, 12, 13, 99, 100])
        with envs.SGLANG_RADIX_FORCE_MISS.override(True):
            match_prefix_for_req(tree, forced_req)
        self.assertEqual(int(forced_req.prefix_indices.numel()), 0)
        self.assertIs(forced_req.last_node, tree.root_node)
        self.assertIs(forced_req.last_host_node, tree.root_node)
        self.assertEqual(forced_req.host_hit_length, 0)
```

评论区精华

本次 PR 无公开 review 讨论。从提交历史可见的关键设计取舍包括：

- 首次提交将 gate 放在 RadixCache.match_prefix 内部，但随后发现这会破坏 cache_unfinished_req 的插入后重匹配断言，因此将 gate 移到调度器边界的三个外部调用点。
- 早期版本在 schedule_policy.py 中嵌入辅助函数，后根据反馈移到 base_prefix_cache.py 紧邻 MatchResult。
- 对无 root_node 的缓存从静默软降级改为主动抛 RuntimeError，确保不静默泄漏缓存命中。
- 暂无高价值评论线程

风险与影响

- 风险：
  - 回归风险：gate 只新增几行条件判断和函数调用，且只影响标志开启时的路径；默认关闭时仅增加一次布尔环境变量读取，风险极低。但若未来新增 match_prefix 调用点而忘记添加 gate，会导致标志失效，因此需要编码规范提醒。
  - 异常处理：对无 root_node 的缓存抛出 RuntimeError 可能中断服务，但实验性功能用户可通过关闭标志规避。
  - 性能影响：默认关闭时无额外开销；开启时每个调度周期多一次 zero_match_result 调用和切片操作，可忽略。
  - 测试覆盖：已覆盖核心函数和两个调度调用点，但未覆盖 _compute_prefix_matches 中的等待队列前缀树 gate（尽管代码结构一致）。
- 影响：
  - 用户 / 开发者：主要面向性能分析人员，提供标准化基准测试手段。通过 SGLANG_RADIX_FORCE_MISS=1 环境变量即可启用，无需修改代码。
  - 系统：运行时无持久影响，标志仅影响当前进程。
  - 团队：该工具可统一团队内 radix 缓存性能分析的方法论，避免因禁用缓存而测到不同代码路径。
  - 影响范围：低，属于调试辅助功能，默认关闭。
  - 风险标记：若新增 match_prefix 调用点遗漏 gate 则标志失效，无 root_node 的缓存后端会直接崩溃，等待队列前缀树 gate 未被测试直接覆盖

关联脉络

- PR #23189 feat(scheduler): add adaptive queue-based prefill delayer trigger: 同一调度器模块，涉及 schedule_policy.py 和 scheduler.py 的前缀匹配逻辑，与本 PR 共同构成 radix 缓存调度路径的调优基础设施。
- PR #24632 fix(fa3): skip scheduler_metadata precompute under DP attention: 同样修改了调度批处理相关逻辑，与本 PR 有潜在冲突区域（schedule_batch.py）。