PR #26549 完整报告

sgl-project/sglang

[UnifiedTree]: Support eviction priority

合并时间：2026-05-30 15:19

原文链接：http://prhub.com.cn/sgl-project/sglang/pull/26549

执行摘要

- 一句话：为 UnifiedRadixCache 和 RadixCache 引入 eviction priority 支持，节点可分配优先级影响驱逐顺序。
- 推荐动作：建议阅读此 PR，尤其关注 get_eviction_strategy 工厂函数和驱逐排序抽离的设计，对缓存策略扩展有参考价值。如需实现 QoS 分级，可直接使用 priority 策略。

功能与动机

为了使缓存驱逐策略能够区分不同请求的重要性，支持优先级控制，避免关键请求的缓存被低优先级请求过早驱逐。通过引入 priority 字段和 Factory 设计模式，统一并简化了策略初始化，便于后续扩展更多驱逐策略。

实现拆解

1. 引入 eviction strategy 工厂函数：在 utils.py 中新增 get_eviction_strategy 函数，定义 _EVICTION_POLICY_FACTORIES 字典映射策略名称到策略类，函数返回对应实例。取代了 radix_cache.py 和 unified_radix_cache.py 中原有的 if-elif 链。
2. 为节点添加优先级属性：修改 UnifiedTreeNode.__init__，新增 priority 和 creation_time 参数，初始化节点优先级。在 reset 时设置根节点优先级为 -sys.maxsize，确保根节点不会被驱逐。
3. 重构 RadixCache 策略初始化：移除 radix_cache.py 中的导入和条件分支，改为一行 self.eviction_strategy = get_eviction_strategy(self.eviction_policy)，降低重复。
4. 在插入路径传递优先级：在 UnifiedRadixCache.cache_finished_req 和 cache_unfinished_req 中，构造 InsertParams 时增加 priority 字段，从请求对象提取优先级（默认 0）。
5. 修改驱逐堆排序：在 full_component.py 的 drive_eviction 和 drive_host_eviction 中，将堆的比较键从 node.last_access_time 改为 self.cache.eviction_strategy.get_priority(node)，使驱逐顺序由策略决定。
6. 新增单元测试：在 test_unified_radix_cache_unittest.py 中添加 test_evict_respects_priority_policy，构造 eviction_policy='priority' 的缓存，插入高低优先级序列，验证低优先级节点先被驱逐。

关键文件：

- python/sglang/srt/mem_cache/unified_radix_cache.py（模块 统一缓存；类别 source；类型 core-logic；符号 init, reset, cache_finished_req, cache_unfinished_req）：核心变更文件，添加 priority 字段、creation_time、修改插入和重置逻辑，导入 get_eviction_strategy
- python/sglang/srt/mem_cache/utils.py（模块 工具函数；类别 source；类型 dependency-wiring；符号 get_eviction_strategy）：新增 get_eviction_strategy 工厂函数，集中管理策略实例化，为后续扩展提供统一入口
- python/sglang/srt/mem_cache/radix_cache.py（模块 前缀缓存；类别 source；类型 dependency-wiring）：重构初始化，使用 get_eviction_strategy 替代 if-elif 链，移除重复导入
- test/registered/unit/mem_cache/test_unified_radix_cache_unittest.py（模块 单元测试；类别 test；类型 test-coverage；符号 _insert, test_evict_respects_priority_policy）：新增 test_evict_respects_priority_policy 测试用例，验证 priority 策略驱逐顺序
- python/sglang/srt/mem_cache/unified_cache_components/full_component.py（模块 组件驱逐；类别 source；类型 core-logic）：修改驱逐堆排序，使用 eviction_strategy.get_priority 替代 last_access_time

关键符号：get_eviction_strategy, UnifiedTreeNode.init, RadixCache.init, UnifiedRadixCache.cache_finished_req, UnifiedRadixCache.cache_unfinished_req, FullComponent.drive_eviction, FullComponent.drive_host_eviction, test_evict_respects_priority_policy

关键源码片段

python/sglang/srt/mem_cache/unified_radix_cache.py

核心变更文件，添加 priority 字段、creation_time、修改插入和重置逻辑，导入 get_eviction_strategy

```
# python/sglang/srt/mem_cache/unified_radix_cache.py
# UnifiedTreeNode 的 __init__ 方法（修改后）
class UnifiedTreeNode:
    counter = 0
    def __init__(self, tree_components: tuple[ComponentType, ...], priority: int = 0):
        self.children = defaultdict(partial(UnifiedTreeNode, tree_components))
        self.parent: UnifiedTreeNode | None = None
        self.key: Optional[RadixKey] = None
        self.tree_components = tree_components
        # list indexed by ComponentType (int enum 0..N-1)
        self.component_data: list[ComponentData] = [
            ComponentData() for _ in range(_NUM_COMPONENT_TYPES)
        ]
        self.last_access_time = get_and_increase_time_counter()
        self.creation_time = get_and_increase_time_counter() # 新增：记录创建时间，供策略使用
        self.hash_value = None
        self.hit_count = 0
        self.priority = priority # 新增：节点优先级，越高越不易被驱逐
```

```
        self.lru_prev: list[UnifiedTreeNode | None] = [None] * (
            _NUM_COMPONENT_TYPES * 2
        )
        self.lru_next: list[UnifiedTreeNode | None] = [None] * (
            _NUM_COMPONENT_TYPES * 2
        )
        self.id = UnifiedTreeNode.counter
        UnifiedTreeNode.counter += 1
```

python/sglang/srt/mem_cache/utils.py

新增 get_eviction_strategy 工厂函数，集中管理策略实例化，为后续扩展提供统一入口

```
# python/sglang/srt/mem_cache/utils.py
# 新增：策略工厂函数
from sglang.srt.mem_cache.evict_policy import (
    EvictionStrategy, FIFOStrategy, FILOStrategy, LFUStrategy,
    LRUStrategy, MRUStrategy, PriorityStrategy, SLRUStrategy,
)

_EVICTION_POLICY_FACTORIES: dict[str, Callable[[], EvictionStrategy]] = {
    "lru": LRUStrategy,
    "lfu": LFUStrategy,
    "fifo": FIFOStrategy,
    "mru": MRUStrategy,
    "filo": FILOStrategy,
    "priority": PriorityStrategy,
    "slru": SLRUStrategy,
}

def get_eviction_strategy(eviction_policy: str) -> EvictionStrategy:
    """根据策略名称返回对应的 EvictionStrategy 实例。"""
    policy = eviction_policy.lower()
    try:
        return _EVICTION_POLICY_FACTORIES[policy]()
    except KeyError:
        supported = "', '".join(_EVICTION_POLICY_FACTORIES)
        raise ValueError(
            f"Unknown eviction policy: {policy}. Supported policies: '{supported}'."
        ) from None
```

test/registered/unit/mem_cache/test_unified_radix_cache_unittest.py

新增 test_evict_respects_priority_policy 测试用例，验证 priority 策略驱逐顺序

```
# test/registered/unit/mem_cache/test_unified_radix_cache_unittest.py
# 新增测试：验证 priority 策略下高优先级节点保留
def test_evict_respects_priority_policy(self):
    if self.cfg.components != (ComponentType.FULL,):
        self.skipTest("priority policy ordering is covered on Full-only configs")
    priority_cfg = replace(self.cfg, eviction_policy="priority")
```

```
    tree, allocator, req_to_token_pool = build_fixture(priority_cfg)
    seq_high = self._make_seq(1, 2) # 低 token ID
    seq_low = self._make_seq(500, 2) # 高 token ID
    # 插入高优先级（10）和低优先级（0）
    self._insert(tree, allocator, req_to_token_pool, seq_high, priority=10)
    self._insert(tree, allocator, req_to_token_pool, seq_low, priority=0)
    # 驱逐低优先级序列长度
    tree.evict(EvictParams(num_tokens=len(seq_low)))
    # 验证高优先级节点仍在，低优先级节点消失
    m_high = tree.match_prefix(
        MatchPrefixParams(key=RadixKey(array("q", seq_high)))
    )
    m_low = tree.match_prefix(
        MatchPrefixParams(key=RadixKey(array("q", seq_low)))
    )
    self.assertEqual(len(m_high.device_indices), len(seq_high))
    self.assertEqual(len(m_low.device_indices), 0)
    tree.sanity_check()
```

评论区精华

无实质性讨论。机器人自动代码审查总结了改动，无进一步问题。评审者 ispobock 直接批准。

- Eviction priority 设计 (design): 采用工厂函数和策略接口，设计清晰，无改动要求。

风险与影响

- 风险：风险较低。主要风险包括：1）新字段 priority 在请求间传递可能丢失，但当前仅在内存节点持有，未序列化；2）驱逐排序从 LRU 改为策略对象方法，现有 LRU 策略使用 get_priority 返回 last_access_time，行为保持不变；3）重构 radix_cache.py 策略初始化，影响所有使用 RadixCache 的地方，需确保 eviction_policy 参数正确；4）默认 priority 为 0，与未设置优先级请求兼容。
- 影响：影响面仅限于缓存模块。使用默认 LRU 策略的用户无感知。开启 priority 策略后，高优先级请求的缓存更持久，提升服务质量。对团队而言，此 PR 为后续扩展 FIFO、LFU 等策略提供了统一工厂入口，降低维护成本。
- 风险标记：新字段 priority 未序列化，eviction 堆比较键变更，radix_cache 初始化重构影响范围

关联脉络

- 暂无明显关联 PR