PR #26387 完整报告

sgl-project/sglang

Support KV events for UnifiedRadixCache

合并时间：2026-05-27 22:10

原文链接：http://prhub.com.cn/sgl-project/sglang/pull/26387

执行摘要

- 一句话：为 UnifiedRadixCache 添加 KV 事件支持，提升缓存命中率
- 推荐动作：该 PR 值得精读，特别是需要自定义缓存策略或集成第三方路由器的开发者。事件注入的模式（通过 Mixin、在关键操作点记录事件）可复用与其它缓存组件。同时展示了如何为复杂缓存结构编写高质量单元测试。

功能与动机

当 SGLANG_ENABLE_UNIFIED_RADIX_TREE=1 与 --kv-events-config 结合使用时，调度器初始化了 Dynamo KV event publisher，但 UnifiedRadixCache 以前继承 BasePrefixCache 的 take_events 空操作。因此外部 KV 感知路由器得不到统一 radix 树路径的放置事件，导致缓存利用率低下。

实现拆解

1. 类继承变更：在 python/sglang/srt/mem_cache/unified_radix_cache.py 中，将 UnifiedRadixCache 的基类从 BasePrefixCache 改为 KVCacheEventMixin, BasePrefixCache，使其具备事件队列和 take_events 接口。
2. 初始化扩展：在 __init__ 中新增 self.enable_kv_cache_events = params.enable_kv_cache_events 和 self.kv_event_queue = []，用于控制事件启用和暂存事件。
3. 事件发射点注入：在关键缓存操作路径中添加事件记录调用：
   - _add_new_node → _record_store_event(new_node)（GPU 存储事件）
   - _unevict_node_on_insert → _record_store_event(node, medium=StorageMedium.GPU)（重新插入被逐出节点的 GPU 存储事件）
   - _evict_device_leaf → _record_remove_event(node, medium=StorageMedium.GPU)（设备层逐出事件）
   - _evict_host_leaf → _record_remove_event(node, medium=StorageMedium.CPU)（主机层逐出事件）
   - _evict_to_host → _record_remove_event(node, medium=StorageMedium.GPU)（从设备到主机的迁移事件）
   - load_back → _record_store_event(node, medium=StorageMedium.GPU)（从主机加载回设备事件）

- _reset_full → _record_all_cleared_event()（完全重置时清空事件）

4. 测试覆盖：在 test/registered/unit/mem_cache/test_unified_radix_cache_unittest.py 中新增 TestUnifiedRadixCacheKVEvents 类，包含针对 store/remove 事件、split 父哈希继承、HiCache GPU/CPU 过渡等方面的聚焦单元测试。同时扩展 build_fixture 以接收 enable_kv_cache_events 参数，便于测试事件启用场景。

关键文件：

- python/sglang/srt/mem_cache/unified_radix_cache.py（模块 缓存层；类别 source；类型 core-logic；符号 UnifiedRadixCache.init, UnifiedRadixCache._add_new_node, UnifiedRadixCache._unevict_node_on_insert, UnifiedRadixCache._evict_device_leaf）：核心变更文件：修改类继承，添加事件队列，在多个关键路径插入事件记录调用。
- 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 build_fixture, TestUnifiedRadixCacheKVEvents, _insert, _stored_events）：新增测试类全面覆盖事件发射，包括 GPU/CPU store/remove、HiCache 过渡、split 父哈希继承。扩展 build_fixture 支持 KV 事件标志。

关键符号：UnifiedRadixCache.init, UnifiedRadixCache._add_new_node, UnifiedRadixCache._unevict_node_on_insert, UnifiedRadixCache._evict_device_leaf, UnifiedRadixCache._evict_host_leaf, UnifiedRadixCache._evict_to_host, UnifiedRadixCache.load_back, UnifiedRadixCache._reset_full

关键源码片段

python/sglang/srt/mem_cache/unified_radix_cache.py

核心变更文件：修改类继承，添加事件队列，在多个关键路径插入事件记录调用。

```
from sglang.srt.mem_cache.events import KVCacheEventMixin
from sglang.srt.disaggregation.kv_events import StorageMedium

class UnifiedRadixCache(KVCacheEventMixin, BasePrefixCache):
    def __init__(self, params):
        # ... 其他初始化 ...
        self.enable_kv_cache_events = params.enable_kv_cache_events # 是否启用 KV 事件
        self.kv_event_queue = [] # 事件暂存队列

    def _add_new_node(self, ...):
        # ... 分配节点并插入树 ...
        self._record_store_event(new_node) # 发射 BlockStored(GPU) 事件

    def _unevict_node_on_insert(self, node, fresh_value):
        # ... 恢复被逐出的节点 ...
        self._record_store_event(node, medium=StorageMedium.GPU) # 恢复后记录存储事件

    def _evict_device_leaf(self, node, ...):
        # ... 逐出设备层叶子 ...
        self._record_remove_event(node, medium=StorageMedium.GPU) # 发射
        BlockRemoved(GPU)
```

```
def _evict_host_leaf(self, node, ...):
    # ... 逐出主机层叶子 ...
    self._record_remove_event(node, medium=StorageMedium.CPU) # 发射 BlockRemoved(CPU)

def load_back(self, ...):
    # ... 从主机加载回设备 ...
    for node in kv_xfer.nodes_to_load or ():
        self._record_store_event(node, medium=StorageMedium.GPU) # 发射加载事件

def _reset_full(self):
    # ... 完全重置 ...
    self._record_all_cleared_event() # 清空所有事件
```

test/registered/unit/mem_cache/test_unified_radix_cache_unittest.py

新增测试类全面覆盖事件发射，包括 GPU/CPU store/remove、HiCache 过渡、split 父哈希继承。扩展 build_fixture 支持 KV 事件标志。

```
from sglang.srt.disaggregation.kv_events import BlockRemoved, BlockStored, StorageMedium

class TestUnifiedRadixCacheKVEvents(CustomTestCase):
    cfg = CacheConfig(page_size=2, kv_size=64, max_context_len=64)

    def _stored_events(self, tree, medium=None):
        """从树中获取BlockStored事件，可按存储介质过滤"""
        events = [e for e in tree.take_events() if isinstance(e, BlockStored)]
        if medium is not None:
            events = [e for e in events if e.medium == medium]
        return events

    def _removed_events(self, tree, medium=None):
        """从树中获取BlockRemoved事件，可按存储介质过滤"""
        events = [e for e in tree.take_events() if isinstance(e, BlockRemoved)]
        if medium is not None:
            events = [e for e in events if e.medium == medium]
        return events

    # 测试方法（示例）：
    def test_simple_insert_emits_block_stored(self):
        tree, alloc, _ = build_fixture(self.cfg, enable_kv_cache_events=True)
        self._insert(tree, alloc, [1, 2]) # 插入会触发 _add_new_node → _record_store_event
        stored = self._stored_events(tree)
        self.assertGreater(len(stored), 0) # 确认发射了 BlockStored 事件
```

评论区精华

1. 机器人审查发现的遗漏事件：gemini-code-assist[bot] 指出 _unevict_node_on_insert 恢复逐出节点时未记录 store 事件，可能导致外部路由器状态不同步。作者后在提交中补充了 BlockStored(GPU) 事件并添加对应测试。

2. 避免重复实现：hzh0425 提醒 split_node_hash_value 已在主分支中存在，作者合并后移除重复。

- _unevict_node_on_insert 缺少 store 事件 (correctness): 作者在后续提交中补充了 self._record_store_event(node, medium=StorageMedium.GPU) 并添加了对应测试。
- split_node_hash_value 去重 (design): 作者合并主分支并移除了重复代码。

风险与影响

- 风险：
  1. 事件队列清空时机：事件队列 kv_event_queue 可能随 take_events 被消费，但若外部消费者未及时拉取，队列可能累积导致内存增长（当前队列为列表，无大小限制）。风险低，因为每次 take_events 会清空队列。
  2. 性能开销：记录事件本身为轻量操作，但若并发高且事件量大，可能引入微小延迟。现有基准测试显示吞吐提升，说明开销可忽略。
  3. 配置依赖：事件仅当 params.enable_kv_cache_events=True 时启用，默认关闭，不影响现有行为的兼容性。
- 影响：
  - 用户影响：使用动态 KV 感知路由器的用户在启用统一 radix 树后，缓存命中率从 50% 提升至 95%，TTFT 降低约 8 倍，吞吐提升约 30%。
  - 系统影响：UnifiedRadixCache 现在与 RadixCache 并行支持 KV 事件，外部路由器能更智能地放置缓存 block，减少 GPU 间冗余传输。
  - 团队影响：维护者需确保新事件发射点覆盖所有路径（如未来添加新的逐出 / 加载逻辑时同步添加事件）。测试套件为回归提供保障。
  - 风险标记：核心路径变更，事件同步风险，配置依赖

关联脉络

- 暂无明显关联 PR