PR #29265 完整报告

sgl-project/sglang

fix: batch BlockRemoved events per radix node

合并时间：2026-06-26 16:13

原文链接：http://prhub.com.cn/sgl-project/sglang/pull/29265

执行摘要

- 一句话：批量 radix 节点删除事件，降低网络开销
- 推荐动作：值得精读，尤其是事件设计的权衡：批量 vs 离散。对理解 SGLang KV 事件机制和 radix 缓存清除逻辑有帮助。核心变更简单但经过充分验证，可放心合并。

功能与动机

一个被删除的 radix 节点可能包含多个完整页面。每个页面单独发送 BlockRemoved 增加事件量和网络开销。将删除 hash 打包发送使事件流更紧凑，同时保持节点内顺序。这对于通过事件流跟踪缓存状态的远程 KV 事件订阅者（如 Dynamo 风格的 KV 感知路由）有益。

实现拆解

1. 修改 python/sglang/srt/mem_cache/events.py 中的 _record_remove_event 方法：移除循环中每页直接发送 BlockRemoved 的逻辑，改用列表收集所有 block_hash，循环后一次性发送 BlockRemoved(block_hashes=block_hashes, medium=medium)。
2. 更新注释：从 "One BlockRemoved per chunk" 改为 "One BlockRemoved per radix node"。
3. 调整 radix cache 单元测试 test_radix_cache_unit.py 中 test_kv_cache_events_with_eviction：使用多页面序列（page_size=2, seq 长度 4），验证 BlockRemoved 事件只有 1 个且包含两个 hash，与存储的 hash 匹配。
4. 在 Mamba、SWA、unified radix cache 测试中添加 _event_hashes 辅助函数，将原来取单个 hash 的断言改为 _event_hashes 收集所有 hash，适配新格式。
5. 手动 KV 事件测试 test_manual/test_kv_events.py 中更新 BlockRemoved 断言：从 len(event.block_hashes)==1 改为 len>0，并使用 removed_hashes.update 收集所有 hash。

关键文件：

- python/sglang/srt/mem_cache/events.py（模块 缓存事件；类别 source；类型 core-logic；符号 _record_remove_event）：核心变更文件，修改 _record_remove_event 方法，实现 BlockRemoved 事件批量发送。
- test/registered/unit/mem_cache/test_radix_cache_unit.py（模块 Radix 缓存；类别 test；类型 test-coverage）：调整 eviction 测试用例，验证批量删除事件的正确性。

- test/registered/unit/mem_cache/test_mamba_unittest.py（模块 Mamba 缓存；类别 test；类型 test-coverage；符号 _event_hashes）：添加 _event_hashes 辅助函数，适配多 hash 格式的断言。
- test/registered/unit/mem_cache/test_swa_unittest.py（模块 SWA 缓存；类别 test；类型 test-coverage；符号 _event_hashes）：添加 _event_hashes 辅助函数，适配多 hash 格式的断言。
- test/registered/unit/mem_cache/test_unified_radix_cache_unittest.py（模块 统一缓存；类别 test；类型 test-coverage；符号 _event_hashes）：添加 _event_hashes 并将断言从单 hash 改为多 hash 的兼容。
- test/manual/test_kv_events.py（模块 手动测试；类别 test；类型 test-coverage）：更新 BlockRemoved 断言，支持多 hash。

关键符号：_record_remove_event

关键源码片段

python/sglang/srt/mem_cache/events.py

核心变更文件，修改 _record_remove_event 方法，实现 BlockRemoved 事件批量发送。

```
def _record_remove_event(self, node: Any, medium=None):
    # One BlockRemoved per radix node.
    # ``medium`` 默认 StorageMedium.GPU，调用者可重写为更低层级（如 StorageMedium.
    CPU）。
    if self.enable_kv_cache_events:
        if medium is None:
            medium = StorageMedium.GPU

        # 若 hash_value 未计算（需与存储时一致），则惰性计算。
        if node.hash_value is None:
            node.hash_value = compute_node_hash_values(node, self.page_size)

        block_hashes = []
        logical_len = len(node.key)
        page_index = 0
        for start in range(0, logical_len, self.page_size):
            end = min(start + self.page_size, logical_len)
            if end <= start:
                continue
            # 将每个页面的 hash 收集到列表中，循环结束后一次性发送。
            block_hashes.append(hash_str_to_int64(node.hash_value[page_index]))
            page_index += 1

        if block_hashes:
            self.kv_event_queue.append(
                BlockRemoved(block_hashes=block_hashes, medium=medium)
            )
```

test/registered/unit/mem_cache/test_radix_cache_unit.py

调整 eviction 测试用例，验证批量删除事件的正确性。

```
def test_kv_cache_events_with_eviction(self):
    mock_allocator = unittest.mock.Mock()
    mock_allocator.device = torch.device("cpu")

    cache = RadixCache.create_simulated(
        mock_allocator=mock_allocator,
        page_size=2, # 固定 page_size 为 2，确保多页删除。
        enable_kv_cache_events=True,
    )

    seq = [1, 2, 3, 4] # 4 tokens → 2 pages，仅一个 radix 节点。
    cache.insert(
        InsertParams(
            key=RadixKey(array("q", seq)),
            value=torch.tensor([10, 20, 30, 40], dtype=torch.int64),
        )
    )
    result = cache.evict(EvictParams(num_tokens=len(seq)))
    ...
    events = cache.take_events()
    # 收集所有存储事件产生的 hash。
    stored_hashes = [
        event.block_hashes[0] for event in events if isinstance(event, BlockStored)
    ]
    self.assertEqual(len(stored_hashes), 2)

    remove_events = [e for e in events if isinstance(e, BlockRemoved)]
    # 验证仅产生 1 个 BlockRemoved 事件，且包含所有存储的 hash。
    self.assertEqual(len(remove_events), 1)
    self.assertEqual(remove_events[0].block_hashes, stored_hashes)
```

评论区精华

核心讨论围绕 PR 优先级和 CI 结果。ishandhanani 要求优先合并以进行基准测试，确认事件测试通过即可。经过 rerun unit/mem_cache 专用 CI，所有 16 个测试变绿。剩余失败 job（如 B200 NVFP4 GSM8K 精度阈值失败）与 PR 变更无关。PeaBrane 请求合并，Reviewer hzh0425 批准后合并。

- PR 优先级和 CI 验证 (other): CI 全绿，reviewer 批准，PR 被合并。

风险与影响

- 风险：风险较低。主要风险是下游消费者是否兼容多 hash 的 block_hashes。PR 已验证 experimental/sgl-router 已支持向量负载，且 SMG 不消费该事件。无其他内部消费者发现。但若有未发现的外部消费者以单 hash 假设处理，则需更新。建议在合并后观察兼容性反馈。
- 影响：对最终用户透明，不改变任何 API 或行为。内部事件流更紧凑，减少序列化 / 网络开销。对跟踪缓存状态的下游系统（如 Dynamo 路由）有积极影响。对维护者，需要知晓事

件负载可能从单 hash 变为多 hash。

- 风险标记：下游消费者兼容性已验证，事件量减少

关联脉络

- PR #29044 Fix KV event publisher bind conflict under PP: 也与 KV 事件发布相关，修改了 scheduler 和 kv_events_publisher.py，影响事件流可靠性。