PR #22894 完整报告

sgl-project/sglang

fix(hicache): emit KV events for L2 host cache insertions

合并时间：2026-04-21 10:07

原文链接：http://prhub.com.cn/sgl-project/sglang/pull/22894

执行摘要

- 一句话：修复 HiCache L2 主机缓存插入时缺失 KV 事件发射的问题。
- 推荐动作：建议技术管理者和工程师精读此 PR，关注设计决策如存储介质枚举化和事件发射时机的优化。这些模式展示了如何扩展事件系统以支持多级缓存，对于类似事件驱动的架构有借鉴价值。

功能与动机

根据 PR body，此变更是共享 KV 缓存路由（ai-dynamo/dynamo#7536）的先决条件。Dynamo 路由器依赖 KV 事件构建基数树进行设备级重叠评分。当 HiCache 从 L3（Mooncake）获取页面到 L2（主机）时，路由器需要看到这些事件来更新每个工作者的缓存视图。

实现拆解

1. 定义存储介质枚举：在 kv_events.py 中，将字符串常量重构为 StorageMedium 枚举，包含 GPU、CPU、DISK、EXTERNAL 层级，以标准化存储介质表示。
2. 扩展事件发射方法：在 radix_cache.py 中，修改 _record_store_event 和 _record_remove_event 方法，添加 medium 参数，默认使用 StorageMedium.GPU，允许调用者覆盖用于不同存储层级。
3. 在主机缓存操作中调用事件：在 hiradix_cache.py 中，多处修改如 _insert_helper_host、writing_check、_evict_backuped、evict_host 和 load_back，在缓存状态变化时发射带正确介质（如 StorageMedium.CPU 或 StorageMedium.GPU）的事件。
4. 延迟事件发射确保准确性：根据 review 反馈，将 store(CPU) 事件从 write_backup 移到 writing_check，在 DMA 传输确认完成后发射，避免在传输未完成时发出错误事件。
5. 配套调整：更新 .gitignore 忽略 artifacts 目录，无关紧要；无直接测试文件变更，但通过指定单元测试验证。

关键文件：

- python/sglang/srt/mem_cache/radix_cache.py（模块 基数树缓存；类别 source；类型 core-logic；符号 _record_store_event, _record_remove_event）：核心事件发射方法的修改，添加介质参数，影响所有缓存事件的生成。
- python/sglang/srt/disaggregation/kv_events.py（模块 缓存事件；类别 source；类型 data-contract；符号 StorageMedium）：定义 StorageMedium 枚举，标准化存储介质表

示，为事件提供类型安全。

- python/sglang/srt/mem_cache/hiradix_cache.py（模块 主机缓存；类别 source；类型 dependency-wiring）：在主机缓存操作中调用事件发射，指定正确介质，确保下游索引器能跟踪缓存状态。

关键符号：_record_store_event, _record_remove_event, write_backup, writing_check, _insert_helper_host, _evict_backuped, evict_host, load_back

关键源码片段

python/sglang/srt/mem_cache/radix_cache.py

核心事件发射方法的修改，添加介质参数，影响所有缓存事件的生成。

```
def _record_store_event(self, node: TreeNode, medium=None):
    # 每个 page_size 块发出一个 BlockStored 事件。
    # medium 默认为 StorageMedium.GPU，但调用者可覆盖用于低级插入（如主机 /L2 缓存）。
    if self.enable_kv_cache_events:
        if medium is None:
            medium = StorageMedium.GPU # 默认 GPU 介质
        # 如果未设置，延迟计算 hash_value
        if node.hash_value is None:
            node.hash_value = compute_node_hash_values(node, self.page_size)
        # 获取父节点最后一个哈希值用于第一页
        parent_block_hash = None
        if node.parent is not None and node.parent != self.root_node:
            if node.parent.hash_value is not None and len(node.parent.hash_value) > 0:
                parent_block_hash = hash_str_to_int64(node.parent.hash_value[-1])
        page_index = 0
        for start in range(0, len(node.key), self.page_size):
            page_tokens = node.key.token_ids[start : start + self.page_size]
            if not page_tokens:
                continue
            block_hash = hash_str_to_int64(node.hash_value[page_index])
            self.kv_event_queue.append(
                BlockStored(
                    block_hashes=[block_hash],
                    parent_block_hash=parent_block_hash,
                    token_ids=page_tokens,
                    block_size=len(page_tokens),
                    lora_id=None,
                    medium=medium, # 使用传入的存储介质
                )
            )
            parent_block_hash = block_hash
            page_index += 1
```

python/sglang/srt/disaggregation/kv_events.py

定义 StorageMedium 枚举，标准化存储介质表示，为事件提供类型安全。

```
import enum

class StorageMedium(str, enum.Enum):
    """KV 缓存事件的存储层级。"""
    GPU = "GPU" # L1: 设备 HBM
    CPU = "CPU_PINNED" # L2: 主机固定内存
    DISK = "DISK" # L3: SSD / NVMe
    EXTERNAL = "EXTERNAL" # L4: 共享 / 远程池（如 Mooncake）
```

python/sglang/srt/mem_cache/hiradix_cache.py

在主机缓存操作中调用事件发射，指定正确介质，确保下游索引器能跟踪缓存状态。

```
    def writing_check(self, write_back=False):
        if write_back:
            while len(self.ongoing_write_through) > 0:
                for _, finish_event, ack_list in self.cache_controller.ack_write_queue:
                    finish_event.synchronize() # 等待 DMA 传输完成
                    for ack_id in ack_list:
                        backuped_node = self.ongoing_write_through.pop(ack_id)
                        # DMA 确认完成 -- 块现在在主机上，发射 store(CPU) 事件
                        self._record_store_event(backuped_node, medium=StorageMedium.CPU)
                        if self.enable_storage:
                            self.write_backup_storage(backuped_node)
                self.cache_controller.ack_write_queue.clear()
                assert len(self.ongoing_write_through) == 0
            return
        # 其他代码省略 ...
```

评论区精华

review 中关键讨论点：

- 存储介质定义：ShangmingCai 建议使用枚举代替字符串常量以提高类型安全，ishandhanani 采纳并实现为 StorageMedium StrEnum，序列化保持与旧字符串相同，确保向后兼容。
- 事件发射时机：huangtingwei9988 指出应在 DMA 传输完成后发射事件以避免 write_backup 失败时的错误事件，ishandhanani 将 store(CPU) 事件延迟到 writing_check 中确认传输完成。
- 初始问题：gemini-code-assist[bot] 发现 _record_store_event 硬编码为 GPU 介质，导致主机缓存事件错误，已通过添加介质参数修复。所有讨论均已解决，PR 得到批准。
- 存储介质定义优化 (design): ishandhanani 实现为 StorageMedium StrEnum，保持向后兼容。
- 事件发射时机准确性 (correctness): ishandhanani 将事件发射移到 writing_check 中，确认传输完成。

风险与影响

- 风险：技术风险包括：
  - 回归风险：修改核心事件发射逻辑可能影响现有缓存管理，但通过运行指定单元测试（如 test_radix_cache_unit.py）覆盖，降低了风险。
  - 性能影响：新增枚举和参数传递开销微小，事件发射频率不变。
  - 兼容性：StorageMedium 枚举是 StrEnum，序列化为字符串值，与旧事件格式完全兼容，不会破坏下游消费者。
  - 事件准确性：延迟发射确保事件反映实际缓存状态，避免了因异步传输导致的错误。
- 影响：影响范围：
  - 用户：对最终用户透明，无直接功能变化，但提高了共享缓存路由的准确性。
  - 系统：增强缓存事件的可观测性，使下游路由（如 Dynamo）能基于更完整的缓存视图进行重叠评分，优化请求分配和资源利用率。
  - 团队：为共享缓存集成提供基础，促进跨团队协作，代码更易维护和扩展。
  - 风险标记：事件发射时机调整，枚举兼容性，核心路径变更

关联脉络

- 暂无明显关联 PR