PR #25959 完整报告

sgl-project/sglang

Ensure multi-node MM embedding cache consistency in insert_batch

合并时间：2026-05-29 13:59

原文链接：http://prhub.com.cn/sgl-project/sglang/pull/25959

执行摘要

- 一句话：确保多节点 MM 嵌入缓存一致性
- 推荐动作：该 PR 改动较小但重要，值得合入。建议后续考虑增加测试覆盖多节点场景下的缓存一致性。

功能与动机

在多节点部署场景中，一个节点本地缓存的嵌入数据对其他节点不可见，导致其他节点需要重复计算或无法获取。通过始终将嵌入数据推送到 Mooncake 全局缓存，确保所有节点都能共享缓存数据，提升系统整体效率。

实现拆解

1. 修改 insert_batch 方法：移除对本地缓存命中（h in self.hash_to_metadata）时的 continue 逻辑，改为即使命中也将数据指针和大小加入待推送列表。
2. 数据完整性处理：针对 review 中提出的风险，作者解释 Mooncake 的 batch_put 底层调用了 batch_is_exist 检查，仅当键不存在时才实际传输数据，因此现有本地偏移量对应的数据一定是已完成的，不存在未初始化风险。
3. 添加统计变量：引入 local_hit_count、new_count 和 skipped_count 三个计数器，分别统计本地命中、新插入和因分配失败跳过的数量。
4. 增强日志：修改日志信息，输出更详细的统计数据，便于调试和监控。
5. 更新文档字符串：为 insert_batch 方法添加文档注释，说明即使本地缓存命中也会推送以保证多节点一致性。

关键文件：

- python/sglang/srt/mem_cache/storage/mooncake_store/embedding_cache_controller.py（模块 缓存层；类别 source；类型 entrypoint；符号 insert_batch）：核心变更文件，修改了 insert_batch 方法，确保本地缓存命中时也推送数据到 Mooncake 集群，并增加详细日志。

关键符号：insert_batch

关键源码片段

python/sglang/srt/mem_cache/storage/mooncake_store/embedding_cache_controller.py

核心变更文件，修改了 insert_batch 方法，确保本地缓存命中时也推送数据到 Mooncake 集群，并增加详细日志。

```
def insert_batch(
    self, image_hashes: List[str], embedding_tensors: List[torch.Tensor]
):
    """Issues ONE batch PUT for all embeddings computed by this request.

    Note: Even if the embedding exists locally, we still push to Mooncake
    to ensure multi-node cache consistency. Mooncake's batch_put has
    built-in deduplication to avoid redundant transfers.
    """
    keys, ptrs, sizes = [], [], []
    local_hit_count = 0 # 本地缓存命中计数
    new_count = 0 # 新插入计数
    skipped_count = 0 # 因分配跳过计数

    with self.lock:
        for h, tensor in zip(image_hashes, embedding_tensors):
            if h in self.hash_to_metadata:
                # 本地缓存命中：仍然推送到 Mooncake 保证多节点一致性
                offset, num_tokens, dim, size_bytes = self.hash_to_metadata[h][:4]
                keys.append(h)
                ptrs.append(self.cpu_pool.data_ptr() + offset)
                sizes.append(size_bytes)
                local_hit_count += 1
                continue

            # 本地未命中：分配空间并复制数据
            num_tokens, dim = tensor.shape[0], tensor.shape[1]
            size_bytes = num_tokens * dim * self.element_size
            offset = self.allocator.allocate(size_bytes)
            if offset is None:
                skipped_count += 1
                continue

            # 复制到 pinned pool 用于 RDMA
            target_view = (
                self.cpu_pool[offset : offset + size_bytes]
                .view(torch.float32)
                .view(num_tokens, dim)
            )
            target_view.copy_(tensor.cpu())
            self.hash_to_metadata[h] = (offset, num_tokens, dim, size_bytes)

            keys.append(h)
```

```
            ptrs.append(self.cpu_pool.data_ptr() + offset)
            sizes.append(size_bytes)
            new_count += 1

        if keys:
            logger.info(
                f"Global Cache: Inserting {len(keys)} embeddings into Mooncake cluster "
                f"({new_count} new, {local_hit_count} existing for replication, "
                f"{skipped_count} skipped due to allocation failure)"
            )
            self.insert_queue.put(EmbeddingInsertOperation(keys, ptrs, sizes))
```

评论区精华

gemini-code-assist[bot] 提出高优先级问题：本地缓存命中时，如果之前 prefetch 操作尚未完成或失败，offset 处的数据可能是未初始化的，直接推送可能损坏全局缓存。建议即使本地命中也要复制 tensor 到 cpu_pool。QiuMike 回复解释了 Mooncake 的 batch_put 内部调用了 batch_is_exist，仅当键不存在时才发送数据，因此现有数据一定是可靠的。最终 liusy58 批准了 PR，未采纳修改建议。

- 本地缓存命中时数据完整性风险 (correctness): QiuMike 解释 Mooncake 的 batch_put 内部调用 batch_is_exist，仅当键不存在时才传输数据，因此偏移量对应数据一定是已完成写入的，不存在风险。最终未修改代码。

风险与影响

- 风险：低风险。变更集中在单个方法内，逻辑清晰。Mooncake 的 batch_put 去重机制保证了即使推送已存在键也不会造成重复传输。需要关注的是如果 hash_to_metadata 中的偏移量指向的数据被错误释放或覆盖，可能推送无效数据，但现有代码中没有此类操作。
- 影响：影响范围有限，仅影响多节点 MM 嵌入缓存一致性场景。对于单节点部署无影响（因为本地缓存已经一致）。变更增加了日志输出，便于运维监控。
- 风险标记：依赖内部去重机制，日志增加但没有监控

关联脉络

- PR #25083 fix(mooncake): honour MOONCAKE_PROTOCOL so EFA hardware can select efa transport: 同为 Mooncake 传输引擎相关的 bugfix，修改了相同模块的配置和初始化逻辑。
- PR #22587 [EPD] Optimize the Mooncake backend: 优化 Mooncake 后端，涉及视觉嵌入的 RDMA 零拷贝传输，与本 PR 共享 Mooncake 存储后端和嵌入缓存上下文。