PR #16370 完整报告

sgl-project/sglang

【hicache】Optimize HiCache prefetch logic: adapt to remaining available memory when memory is insufficient

合并时间：2026-04-28 08:18

原文链接：http://prhub.com.cn/sgl-project/sglang/pull/16370

执行摘要

- 一句话：HiCache 预取自适应剩余可用内存优化
- 推荐动作：本 PR 是一次针对性的性能优化，逻辑清晰，值得阅读以理解 HiCache 预取流程和内存自适应策略。对于使用 HiCache 的部署场景，该改动能带来实际收益。建议在合并后运行相关测试（如 test_hicache_storage_mooncake_backend）验证无回归。

功能与动机

原始逻辑在分配失败后直接放弃预取，浪费了可能满足条件的可用内存。本 PR 通过在失败后获取可用大小并重新计算对齐长度，若大于阈值则继续分配，从而充分利用内存资源。

实现拆解

1. 在 prefetch_from_storage 方法中，当第二次调用 mem_pool_host.alloc 仍返回 None 时，不再直接释放节点并返回，而是通过 mem_pool_host.available_size() 获取当前可用内存大小。
2. 将可用大小按页面大小对齐，得到新的 prefetch_length。
3. 判断新 length 是否 >= prefetch_threshold：若满足，则裁剪 new_input_tokens 为前 prefetch_length 个 token，并重新调用 alloc；若不满足，则释放 host 节点并返回。
4. 后续的 prefetch 操作使用调整后的 host_indices 和 new_input_tokens，其余逻辑保持不变。

关键文件：

- python/sglang/srt/mem_cache/hiradix_cache.py（模块 缓存层；类别 source；类型 core-logic；符号 prefetch_from_storage）：核心修改文件，优化预取逻辑的内存自适应分配

关键符号：prefetch_from_storage

关键源码片段

python/sglang/srt/mem_cache/hiradix_cache.py

核心修改文件，优化预取逻辑的内存自适应分配

```
  def prefetch_from_storage(
```

```
    self,
    req_id: str,
    last_host_node: TreeNode,
    new_input_tokens: List[int],
    last_hash: Optional[str] = None,
    prefix_keys: Optional[List[str]] = None,
):
    # 如果是 eagle 模型，将 token 转换为 bigram key
    new_input_tokens = (
        convert_to_bigram_key(new_input_tokens)
        if self.is_eagle
        else new_input_tokens
    )
    # 对齐到页面大小
    prefetch_length = len(new_input_tokens) - (
        len(new_input_tokens) % self.page_size
    )
    new_input_tokens = new_input_tokens[:prefetch_length]
    # 检查前置条件：启用了存储、长度达到阈值、且速率限制未启用
    if (
        not self.enable_storage
        or prefetch_length < self.prefetch_threshold
        or self.cache_controller.prefetch_rate_limited()
    ):
        return

    last_host_node.protect_host()
    host_indices = self.cache_controller.mem_pool_host.alloc(prefetch_length)
    if host_indices is None:
        self.evict_host(prefetch_length)
        host_indices = self.cache_controller.mem_pool_host.alloc(prefetch_length)
    if host_indices is None:
        # 新增逻辑：获取可用内存大小并自适应调整预取长度
        avaliable_size = self.cache_controller.mem_pool_host.available_size()
        # 对齐到页面大小
        prefetch_length = avaliable_size - (avaliable_size % self.page_size)
        if prefetch_length >= self.prefetch_threshold:
            new_input_tokens = new_input_tokens[:prefetch_length]
            host_indices = self.cache_controller.mem_pool_host.alloc(
                prefetch_length
            )
        else:
            last_host_node.release_host()
            # 没有足够主机内存用于预取
            return
    # 执行预取操作
    operation = self.cache_controller.prefetch(
        req_id,
        host_indices,
```

```
            new_input_tokens,
            last_hash,
            prefix_keys,
            **self._get_extra_pools(),
        )
        self.ongoing_prefetch[req_id] = (
            last_host_node,
            new_input_tokens,
            host_indices,
            operation,
        )
        self.cache_controller.prefetch_tokens_occupied += len(new_input_tokens)
```

评论区精华

Review 中 hzh0425 提问：改变 prefetch_length 后 last_hash 和 prefix_keys 是否会不匹配？作者 CLFutureX 回应：last_hash 和 prefix_keys 依赖于已匹配的节点（req.fill_ids[:matched_len]），而 prefetch_length 作用于后续未匹配节点（req.fill_ids[matched_len:]），两者互不干扰。提问者接受解释并请求其他 reviewer，最终 xiezhq-hermann 批准。

- 改变 prefetch_length 后 last_hash 和 prefix_keys 是否匹配？ (correctness): 提问者接受解释并请求其他 reviewer，最终 reviewer 批准合并。

风险与影响

- 风险：
  1. 改变 prefetch_length 后，若与 last_hash/prefix_keys 的匹配关系假设有误，可能引起预取数据错误。但作者已从代码逻辑层面解释无冲突，风险较低。
  2. 未新增测试用例，回归风险依赖现有测试覆盖，由于 HiCache 测试较少，需关注。
  3. 引入了额外的 available_size 调用，可能带来微小性能开销，但通常可忽略。 - 影响：正面：提高主机内存利用率，减少因内存碎片导致的预取放弃，从而提升整体预取命中率和推理性能。负面影响：代码复杂度略有增加，但改动量小（+11/-3），易于理解。影响范围：仅 HiCache 预取路径，不影响其他模块。开发者需要了解新的自适应行为。 - 风险标记：缺少测试覆盖，核心路径变更

关联脉络

- PR #20460 [HiCache] Add synchronization for context parallelism: 同一模块（HiCache）的改进，涉及预取相关逻辑，可了解整体演进方向。