PR #46412 完整报告

vllm-project/vllm

[Mooncake] Only check and store new KV cache range

合并时间：2026-06-24 18:10

原文链接：http://prhub.com.cn/vllm-project/vllm/pull/46412

执行摘要

- 一句话：仅增量检查并存储 Mooncake KV 缓存
- 推荐动作：建议阅读，尤其增量存储思路和 process_tokens 重构，展示了如何通过记录 offset 优化重复路径。对于 Mooncake Connector 后续开发有参考价值。

功能与动机

减少存储路径中的 CPU 开销，特别是 batch_is_exist 查找。现有代码每次存储都从头检查 KV 缓存范围（PR body），该 PR 通过增量方式只处理新生成的 KV 缓存，从而节省 CPU 开销。benchmark 显示 DeepSeek v4 在 8×B300 上有显著性能提升。

实现拆解

1. worker.py：增量存储核心，新增 _saved_offset 字典记录每个请求已保存的 token 数，_record_saved 方法更新该标记；_handle_request 先从 _saved_offset 读取 save_start，传递给 store_mask 和 process_tokens，只处理新产生的 KV 缓存。
2. data.py：process_tokens 重构，重写 process_tokens 方法，接受 chunk_mask、put_step、put_step_rank 参数，支持按 chunk_id 跨 TP 秩分发以避免重复存储；新增 key_for 方法（替代旧 _make_key_by_hash）直接生成池键字符串，减少对象创建开销。
3. coordinator.py：store_mask 支持偏移，store_mask 方法新增 start_token 参数，传递至 _reachable_masks，使得可以从指定位置开始计算 mask，配合增量存储。
4. 测试配套，新增 test_mooncake_store_worker.py 大量测试验证 process_tokens 新参数组合（mask、stride）和增量存储正确性；新增 test_mooncake_store_coordinator.py 测试 store_mask 前缀稳定性；调整 test_mooncake_store_hma_e2e.py 适应新的 BlockHash 返回类型。

关键文件：

- vllm/distributed/kv_transfer/kv_connector/v1/mooncake/store/worker.py（模块 存储线程；类别 source；类型 core-logic；符号 _record_saved）：核心变更：添加 _saved_offset 字典和 _record_saved 方法，修改 _handle_request 以使用 save_start 优化增量存储。
- vllm/distributed/kv_transfer/kv_connector/v1/mooncake/store/data.py（模块 缓存数据库；类别 source；类型 core-logic；符号 _make_key_by_hash, key_for）：重构 process_tokens 方法，支持 chunk_mask、put_step 等参数，新增 key_for 方法替代旧

_make_key_by_hash，优化键构建性能。

- vllm/distributed/kv_transfer/kv_connector/v1/mooncake/store/coordinator.py（模块 存储协调器；类别 source；类型 dependency-wiring）：store_mask 方法新增 start_token 参数，传递至 _reachable_masks 以支持从中间位置开始计算 mask，配合增量存储。
- tests/v1/kv_connector/unit/test_mooncake_store_worker.py（模块 存储线程测试；类别 test；类型 test-coverage；符号 _RecordingBlockHashes, init, len, getitem）：新增大量测试覆盖 process_tokens 新参数（mask、stride）和增量存储正确性，引入 _RecordingBlockHashes 辅助类追踪 hash 访问。
- tests/v1/kv_connector/unit/test_mooncake_store_coordinator.py（模块 协调器测试；类别 test；类型 test-coverage；符号 test_store_mask_swa_prefix_stable_as_aligned_length_grows, test_store_mask_suffix_matches_full_mask_tail, test_store_mask_retention_prefix_stable_as_aligned_length_grows）：新增测试验证 store_mask 前缀在 token 长度增长时的稳定性，以及 suffix mask 与 full mask 尾部的匹配性。
- tests/v1/kv_connector/unit/test_mooncake_store_hma_e2e.py（模块 端到端测试；类别 test；类型 test-coverage）：微调测试断言以适配 process_tokens 返回类型从 PoolKey 变为 BlockHash。

关键符号：_record_saved, key_for, process_tokens, store_mask, _reachable_masks, _RecordingBlockHashes

关键源码片段

vllm/distributed/kv_transfer/kv_connector/v1/mooncake/store/worker.py

核心变更：添加 _saved_offset 字典和 _record_saved 方法，修改 _handle_request 以使用 save_start 优化增量存储。

```
# vllm/distributed/kv_transfer/kv_connector/v1/mooncake/store/worker.py

class KVCacheStoreSendingThread(threading.Thread):
    def __init__(self, /* ... */):
        # ... 原有初始化 ...
        self._store_pressure_active = False
        self._skip_store_requests: set[str] = set()
        # 新增：记录每个请求已保存的 token 高水位标记，下次恢复从此开始
        self._saved_offset: dict[str, int] = {}

    def _record_saved(self, req_id: str, token_len: int) -> None:
        """记录本次已成功保存的 token 位置，用于后续增量存储.

        使用 done_task_lock 保护，防止并发 finish / preempt 时重新创建.
        """
        with self.done_task_lock:
            if req_id in self.stored_requests:
                self._saved_offset[req_id] = token_len

    def _handle_request(self, req_meta: ReqMeta):
```

```
        lcm_block_size = self.coord.lcm_block_size
        token_len = req_meta.token_len_chunk // lcm_block_size * lcm_block_size
        req_id = req_meta.req_id
        # ... 检查 stored_requests ...
        try:
            if token_len == 0:
                return
            # 从高水位标记开始，只处理新生成的 KV 缓存
            save_start = self._saved_offset.get(req_id, 0)
            store_masks = self.coord.store_mask(
                token_len, save_start,
                num_prompt_tokens=req_meta.num_prompt_tokens
            )
            for g_idx, db in enumerate(self.token_databases):
                put_step_rank = (self.tp_rank + g_idx) % self.put_step
                for start, end, block_hash in db.process_tokens(
                    token_len,
                    req_meta.block_hashes,
                    mask_num=save_start,
                    chunk_mask=store_masks[g_idx],
                    put_step=self.put_step,
                    put_step_rank=put_step_rank,
                ):
                    # 组装 keys、地址列表等
                    ...
            # 记录本次已保存位置，供下次增量使用
            self._record_saved(req_id, token_len)
        except Exception:
            ...
        finally:
            ...
```

vllm/distributed/kv_transfer/kv_connector/v1/mooncake/store/data.py

重构 process_tokens 方法，支持 chunk_mask、put_step 等参数，新增 key_for 方法替代旧 _make_key_by_hash，优化键构建性能。

```
# vllm/distributed/kv_transfer/kv_connector/v1/mooncake/store/data.py

class ChunkedTokenDatabase:
    def __init__(self, metadata, block_size, hash_block_size=None):
        # ...
        self._key_prefix = PoolKey.build_prefix(metadata)

    def key_for(self, chunk_hash: BlockHash) -> str:
        """依据 BlockHash 直接生成池键字符串，避免创建 PoolKey 对象."""
        return PoolKey.build_key_string(self._key_prefix, chunk_hash.hex())

    def process_tokens(
        self,
```

```
        token_len: int,
        block_hashes: list[BlockHash],
        mask_num: int = 0,
        *,
        chunk_mask: list[bool] | None = None,
        put_step: int = 1,
        put_step_rank: int = 0,
    ) -> Iterable[tuple[int, int, BlockHash]]:
        """处理令牌并产生 (起始位置, 结束位置, 块哈希) 三元组.

        当 KV 头数少于 TP 秩数时，按 chunk_id 绝对位置跨 TP 秩分发，
        保证同一 chunk 始终落到同一秩，避免重复存储或加载.
        """
        assert put_step > 0
        if not block_hashes:
            return
        chunk_hashes = chunk_hashes_for_block_size(
            block_hashes, self.hash_block_size, self.block_size
        )
        start_chunk = max(0, cdiv(mask_num, self.block_size))
        max_chunks = min(len(chunk_hashes), cdiv(token_len, self.block_size))
        if chunk_mask is not None:
            max_chunks = min(max_chunks, start_chunk + len(chunk_mask))
        for chunk_id in range(start_chunk, max_chunks):
            if chunk_mask is not None and not chunk_mask[chunk_id - start_chunk]:
                continue
            if chunk_id % put_step != put_step_rank:
                continue
            h = chunk_hashes[chunk_id]
            start_idx = chunk_id * self.block_size
            end_idx = min(start_idx + self.block_size, token_len)
            yield start_idx, end_idx, h
```

评论区精华

reviewer ivanium 指出 key_for 方法中使用 PoolKey(self.metadata, chunk_hash.hex()).to_string()可能较昂贵，建议在另一个优化PR（#46188）合并后再跟进。ivanium 也提供了一个简化 commit（已合并）。最终无未解决问题。

- 新 key_for 方法性能考量 (performance): 经讨论，当前实现已满足性能需求，后续在 #46188 合并后进一步优化。

风险与影响

- 风险：核心存储路径变更，可能引入回归。但已通过大量新增单元测试覆盖主要场景（包括 process_tokens 参数组合和 store_mask 稳定性）。使用 done_task_lock 保护 _saved_offset 并发写入，避免竞态。需注意 _saved_offset 在请求完成时正确清理（ delete_finished_stored_request 中 pop）。对未启用 Mooncake Connector 的系统无影

响。

- 影响：对使用 Mooncake KV Connector 的用户透明，存储路径 CPU 开销降低，高并发长序列场景下吞吐提升。benchmark 显示 DeepSeek v4 在 8×B300 上有明显改善。无 API 或配置变更，团队后续需关注 key_for 的进一步优化。
- 风险标记：核心路径变更，并发控制依赖，新增数据结构清理风险

关联脉络

- PR #46188 [Mooncake] Optimize lookup pool key string construction: 同一作者对 Mooncake 查找键构建的优化，review 讨论中提及可在此 PR 基础上进一步优化 key_for 方法。