# PR #44774 完整报告

vllm-project/vllm

[KV Connector] Mooncake store: prefix-cache retention interval for sparse attention

合并时间：2026-06-11 12:36

原文链接：http://prhub.com.cn/vllm-project/vllm/pull/44774

## 执行摘要

- 一句话：Mooncake store 新增 retention interval 控制
- 推荐动作：本 PR 值得精读，特别是 coordinator.py 中 store_mask 的重构思路：复用引擎自身的 reachable_block_mask 而非独立维护一套模板逻辑，降低了维护成本并保证了行为一致。同时，通过参数化 retention_interval 实现了灵活的稀疏化策略。建议阅读者对比旧实现（可通过 git diff 查看），理解从模板到直接调用 manager 的演进。

## 功能与动机

Follow-up of #43447。原 store_mask 在每个 lcm 边界处为 SWA 群保留尾块，导致稀疏注意力模型（如 DSV4）写入过多无用块。本 PR 重用引擎自身的 reachable_block_mask，并支持 retention_interval 参数，使得 store 只写入未来本地 prefix cache 可能命中的块，显著减少写入量。

## 实现拆解

1. coordinator.py 重构 store_mask：废弃原先使用 _DUMMY_BLOCK_HASH 进行 find_longest_cache_hit 模板运算再平铺的方式；改为遍历每个 kv_cache_group，取出其对应的 KVCacheSpec，并通过 KVCacheSpecRegistry 获取对应的 manager_class，然后调用 manager_class.reachable_block_mask(block_size, retention_interval, ...) 计算掩码。对于 SWA 群，会根据 retention_interval 保留每段间隔的尾块，并额外保留由 num_prompt_tokens 计算出的 replay boundary 尾块。
2. data.py 扩展 ReqMeta：新增 num_prompt_tokens 字段，在 from_request_tracker 中从 tracker.prefill_end_tokens 赋值，后续在 worker 的 _handle_request 中传递给 store_mask。
3. worker.py 集成：在 MooncakeStoreWorker 初始化 coordinator 时传入 retention_interval=envs.VLLM_PREFIX_CACHE_RETENTION_INTERVAL；在 _handle_request 中调用 store_mask 时传入 req_meta.num_prompt_tokens。
4. 测试覆盖：新增 4 个测试用例覆盖 retention_interval 的默认行为（密集）、间隔稀疏化、为 0 时仅保留 replay boundary、间隔与 replay 共存等场景。

关键文件：

- vllm/distributed/kv_transfer/kv_connector/v1/mooncake/store/coordinator.py（模块 KV 连接器；类别 source；类型 core-logic；符号 store_mask）：核心重构：store_mask

方法完全重写，从 dummy-hash 模板机制改为直接使用 SingleTypeKVCacheManager.reachable_block_mask，新增 retention_interval 支持。

- tests/v1/kv_connector/unit/test_mooncake_store_coordinator.py（模块 单元测试；类别 test；类型 test-coverage；符号 _make_coord, _retention_groups, test_store_mask_dense_default_matches_every_lcm_boundary, test_store_mask_retention_interval_sparsifies_swa_tails）：新增 4 个测试覆盖 retention_interval 不同场景，验证行为正确性。
- vllm/distributed/kv_transfer/kv_connector/v1/mooncake/store/worker.py（模块 KV 连接器；类别 source；类型 core-logic）：传递 num_prompt_tokens 和 retention_interval 到 coordinator，是数据流动的关键环节。
- vllm/distributed/kv_transfer/kv_connector/v1/mooncake/store/data.py（模块 KV 连接器；类别 source；类型 core-logic）：ReqMeta 新增 num_prompt_tokens 字段，从 tracker.prefill_end_tokens 赋值，作为 replay boundary 的计算依据。

关键符号：store_mask, _verify_and_split_kv_cache_groups, MooncakeStoreCoordinator.init, ReqMeta.from_request_tracker, MooncakeStoreWorker._handle_request, MooncakeStoreWorker.init

## 关键源码片段

### vllm/distributed/kv_transfer/kv_connector/v1/mooncake/store/coordinator.py

核心重构：store_mask 方法完全重写，从 dummy-hash 模板机制改为直接使用 SingleTypeKVCacheManager.reachable_block_mask，新增 retention_interval 支持。

```
# coordinator.py: store_mask ( 重构后版本 ) defstore_mask(     self,     aligned_token_len: int,     num_prompt_tokens: int | None = None, ) -> tuple[list[bool], ...]: """ Per-group store masks: mask[g][i] is True if chunk i of group g should be written to the store so a future cache hit can consume it. Now reuses the engine's SingleTypeKVCacheManager.reachable_block_mask instead of computing a template via find_longest_cache_hit on dummy hashes. """     assert aligned_token_len % self.lcm_block_size == 0     masks: list[list[bool]] = []     for g_idx, g in enumerate(self.kv_cache_groups):         spec = _unwrap_spec(g.kv_cache_spec)         num_chunks = aligned_token_len // spec.block_size         # 获取对应 spec 的 manager 类         manager_cls = KVCacheSpecRegistry.get_manager_class(spec)         assert manager_cls is not None         # 调用 reachable_block_mask 计算掩码         masks.append(             list(manager_cls().reachable_block_mask(                 num_chunks,                 self.retention_interval,                 num_prompt_tokens=num_prompt_tokens,             ))         )     return tuple(masks)
```

注：实际代码中 manager_cls().reachable_block_mask 可能会通过工厂方法实例化，此处为示意。新实现直接复用本地 prefix cache 的决定逻辑，保证了 store 写入块与本地缓存可达块完全一致。

### tests/v1/kv_connector/unit/test_mooncake_store_coordinator.py

新增 4 个测试覆盖 retention_interval 不同场景，验证行为正确性。

```
# test_mooncake_store_coordinator.py: 新增测试组 # ----- store_mask with retention_interval (DSV4 sparse SWA checkpointing) ----- def _retention_groups(): """Hybrid full-attn(block=32) + SWA(block=8, sw=8); lcm=32."""  full = _full(32)  swa = _swa(block_size=8, sliding_window=8)  return [KVCacheGroupSpec(["L0"], full), KVCacheGroupSpec(["L1"], swa)] def test_store_mask_dense_default_matches_every_lcm_boundary(): """retention_interval=None 时，SWA 群在每个 lcm 边界 (32/64/96/128) 保留尾块."""  coord = _make_coord(_retention_groups(), hash_block_size=8)  masks = coord.store_mask(128)  # full attention group: 全部 True  assert masks[0] == [True, True, True, True]  # SWA group: 16 个 chunk，每 4 个 chunk 的最后一个为 True (128/8=16, lcm=32 => chunks 3,7,11,15)  assert masks[1] == [i % 4 == 3 for i in range(16)] def test_store_mask_retention_interval_sparsifies_swa_tails(): """retention_interval=64 时，SWA 群每 64 token 段保留一个尾块 (chunks 7,15)."""  coord = _make_coord(_retention_groups(), hash_block_size=8, retention_interval=64)  masks = coord.store_mask(128)  assert masks[0] == [True, True, True, True]  assert masks[1] == [i in (7, 15) for i in range(16)] def test_store_mask_retention_interval_zero_keeps_only_replay_boundary(): """retention_interval=0 时，仅保留 replay boundary (num_prompt=100 => chunk 11)."""  coord = _make_coord(_retention_groups(), hash_block_size=8, retention_interval=0)  # 不传入 num_prompt_tokens 时无 replay 信息 -> 全部 False  assert coord.store_mask(128)[1] == [False] * 16  # 传入 num_prompt=100 -> latest hit boundary = (100-1)//32*32 = 96 -> chunk 11  masks = coord.store_mask(128, num_prompt_tokens=100)  assert masks[1] == [i == 11 for i in range(16)] def test_store_mask_retention_interval_keeps_segment_and_replay_tails(): """稀疏段尾块 (interval=64 => 7,15) 与 replay boundary (chunk 11) 共存."""  coord = _make_coord(_retention_groups(), hash_block_size=8, retention_interval=64)  masks = coord.store_mask(128, num_prompt_tokens=100)  assert masks[1] == [i in (7, 11, 15) for i in range(16)]
```

注：测试通过构建固定配置的 coordinator 和 store_mask 调用，验证了 retention_interval 不同参数下的 mask 精确性。

## 评论区精华

团队核心成员 wzhao18 确认该功能非常需要，并建议尽快合并。Dao007forever 审阅后给出 LGTM 评价。CI 曾因 pre-commit 失败和合并冲突被 mergify 提醒，开发者随后 rebase 解决。

- 功能需求确认 (question)：无异议，建议尽快合并。
- 代码审查通过 (other)：两位 reviewer 均 approve。
- CI 问题 (other)：开发者已通过 rebase 解决冲突和 CI 问题。

## 风险与影响

- 风险：
  1. 行为兼容性：retention_interval 默认 None 时保留原行为（在每个 lcm 边界保留 SWA 尾块），但重构后使用 reachable_block_mask 替代模板方式，理论上与原有行为一致，

但需确认 reachable_block_mask 的语义是否完全匹配。测试已覆盖默认行为。

2. 数据依赖：num_prompt_tokens 必须正确传入，否则 replay boundary 计算错误可能导致 cache miss 或写入多余块。该值来自 tracker.prefill_end_tokens，需要确保始终有效。
3. 模块耦合：coordinator 现在依赖于 engine 内部的 SingleTypeKVCacheManager.reachable_block_mask，如果该接口后续变化需要同步更新。
4. 性能影响：原本的快速路径（当所有 attention group 都是 FullAttentionSpec 或 block_size 等于 lcm_block_size 时）在新实现中被移除，现在总是遍历每个 group 并调用 reachable_block_mask。对于只有 full attention 的简单场景，可能引入额外的计算开销（需要检查 manager 的 reachable_block_mask 实现是否简单）。测试中 fast_path 相关用例仍然通过（新实现下 full attention group 的 mask 为全 True），但性能是否退化需验证。 - 影响：对用户：可通过设置环境变量 VLLM_PREFIX_CACHE_RETENTION_INTERVAL 控制 Mooncake store 的写入密度，对使用稀疏注意力模型（如 DSV4）的用户可大幅减少存储开销。对系统：统一了 store 端 mask 逻辑与本地 prefix cache 的一致性，减少冗余写入。对团队：需关注 reachable_block_mask 的接口稳定性及跨组件的同步维护。 - 风险标记：核心路径变更，依赖引擎内部接口，默认行为保留需验证，性能退化风险

# 关联脉络

- PR #43447 之前的 Mooncake store 相关 PR: 本 PR 是 #43447 的跟进，继承了之前的 store_mask 逻辑并进行了增强。