PR #41735 完整报告

vllm-project/vllm

File system secondary tier implemented in python

合并时间：2026-05-25 02:14

原文链接：http://prhub.com.cn/vllm-project/vllm/pull/41735

执行摘要

为 KV 卸载框架新增纯 Python 文件系统二级存储层，支持异步 I/O、原子写入和双优先级线程池。这是 SecondaryTierManager 接口的第一个非示例实现，为多级卸载场景提供了磁盘扩展选项。

功能与动机

PR 目标明确："Added file system secondary tier for multi-tier offload"。在 vLLM 的卸载架构中，此前只有示例级别的二级存储实现，本 PR 提供一个生产可用的纯 Python 文件系统后端，允许 KV 缓存块通过文件系统持久化到磁盘，从而在 GPU 内存不足时借助外部存储扩展容量。

实现拆解

变更涉及 vllm/v1/kv_offload 下的新增文件，按依赖层次分 5 步：

1. FileMapper（文件映射器）：路径 file_mapper.py。负责将卸载键转换为文件路径。通过 SHA-256 哈希前缀散列到三级子目录，避免单目录文件爆炸。支持 parallel_agnostic 模式，使不同并行配置可共享相同存储布局。
2. I/O 原语：路径 tiering/fs/io.py。store_block 先写临时文件再原子 rename，保证写入完整性；load_block 使用 os.readv 直接读取到 memoryview。两者均尝试使用 O_DIRECT 绕过页缓存，并在 macOS 自动降级。
3. DualQueueThreadPool：路径 tiering/fs/thread_pool.py。双队列（load/store）双优先线程组：load 优先线程先处理 load 队列，空闲时窃取 store 任务；store 优先线程反之。JobState 追踪每个 job 的子任务完成状态，完成后将 (job_id, success) 推入完成队列。
4. FileSystemTierManager：路径 tiering/fs/manager.py。继承 SecondaryTierManager，组合 FileMapper 和线程池。submit_store/submit_load 将每个键的 I/O 操作封装为 callable 批量入队；get_finished 从线程池拉取完成结果并转换为 JobResult。
5. 注册与测试：工厂类 tiering/factory.py 加入 FileSystemTierManager，配置 tiering/spec.py 自动推导 gpu_blocks_per_file。新增两个测试文件覆盖单元功能，并扩展现有连接器集成测试。

以下为 I/O 层的核心实现，展示原子写入和读取逻辑：

```
import os
import random
import threading
```

```
O_DIRECT = getattr(os, 'O_DIRECT', 0)
_thread_local = threading.local()

def _get_tmp_suffix() -> str:
    try:
        return _thread_local.tmp_suffix
    except AttributeError:
        _thread_local.tmp_suffix = f'_{random.randint(0, 2**63 - 1)}.tmp'
        return _thread_local.tmp_suffix

def _ensure_dirs(path: str) -> None:
    os.makedirs(os.path.dirname(path), exist_ok=True)

def store_block(dest_path: str, buffer: memoryview, offset: int, block_size: int) -> None:
    if os.path.exists(dest_path):
        return
    tmp_path = dest_path + _get_tmp_suffix()
    _ensure_dirs(dest_path)
    view_slice = buffer.cast('B')[offset : offset + block_size]
    try:
        fd = os.open(tmp_path, os.O_CREAT | os.O_EXCL | os.O_WRONLY | os.O_TRUNC | O_
        DIRECT, 0o644)
        try:
            written = os.write(fd, view_slice)
            if written < len(view_slice):
                raise OSError(f'Short write: expected {len(view_slice)} bytes, wrote {written}')
        finally:
            os.close(fd)
        os.replace(tmp_path, dest_path)
    except Exception:
        try:
            os.remove(tmp_path)
        except OSError as cleanup_exc:
            logger.warning('Failed to remove temp file %s: %s', tmp_path, cleanup_exc)
        raise

def load_block(source_path: str, view: memoryview, offset: int, block_size: int) -> None:
    fd = None
    view_slice = view.cast('B')[offset : offset + block_size]
    try:
        fd = os.open(source_path, os.O_RDONLY | O_DIRECT)
        bytes_read = os.readv(fd, [view_slice])
        if bytes_read < block_size:
            raise OSError(f'Short read: expected {block_size} bytes, read {bytes_read}')
    except Exception:
        try:
            os.remove(source_path)
        except OSError as cleanup_exc:
            logger.warning('Failed to remove unreadable file %s: %s', source_path, cleanup_exc)
```

```
            raise
        finally:
            if fd is not None:
                os.close(fd)
```

评论区精华

- 架构拆分：orozery 要求将大文件拆分为 manager、thread_pool 和 io，并重命名去下划线，保障可维护性。作者全部采纳。
- 完成追踪设计：orozery 指出 manager 轮询活跃字典效率低，建议线程池内建完成队列。最终通过 DualQueueThreadPool.get_finished 直接返回完成项，使 manager 代码简化。
- O_DIRECT 兼容：orozery 提醒 os.O_DIRECT 在 macOS 不可用，作者使用 getattr 优雅降级。
- shutdown 行为：orozery 认为停止时应丢弃所有未完成任务（best effort），作者实现清空队列。
- 磁盘空间管理：NUABO 提问 eviction 策略，orozery 回复依赖外部清理，本实现假设磁盘有足够空间。

风险与影响

风险	描述	影响
性能	文件系统 I/O 可能在高吞吐下成为瓶颈，O_DIRECT 在 macOS 不可用	中等
兼容	O_DIRECT 仅 Linux，macOS 自动降级	低
磁盘空间	无内置 eviction，写满会报错	需外部清理
临时文件	异常 crash 可能残留 .tmp 文件	低
测试覆盖	单元测试覆盖核心路径，集成测试通过	中

关联脉络

本 PR 与 KV 卸载模块的演进紧密相关：#44454 重构了 KV 缓存配置，为本 PR 的 FileMapper 提供稳定的配置接口；#44854 移除了不再使用的连接器，显示卸载层在精简和增强并进。后续可在此基础上添加对象存储等二级 tier 实现。