

PR #49644 完整报告

vllm-project/vllm

[Feat][Core] Add disk offloading support to SimpleCPUOffloadConnector

合并时间: 2026-08-07 15:37

原文链接: <http://prhub.com.cn/vllm-project/vllm/pull/49644>

执行摘要

- 一句话: KV offload 新增磁盘后端, reload 提速超 10 倍
- 推荐动作: 值得精读, 尤其是 `disk_backend.py` 的双缓冲流水线与 `launch_copy` 接口抽象: 它演示了如何在不改动 worker 传输逻辑的前提下替换 offload 后端, 以及 `O_DIRECT` 对齐、`iovec` 预构建、`store/load` 队列分离等工程细节。设计决策上推荐关注两点: 一是调度层完全复用 `BlockPool/LRU` 使得磁盘槽位与 CPU 块同构, 大幅降低接入成本; 二是安全加固 (先 `unlink` 再 `O_EXCL`、`shutdown` 泄漏 `fd`) 体现了对多租户环境的考量。后续应补齐单测并明确 IO 线程异常恢复策略。

功能与动机

PR body 明确这是 RFC #19854 设想的可插拔 offload 后端中的磁盘扩展: "Add disk offloading support" 旨在覆盖 "host DRAM is limited but local NVMe capacity is abundant" 的场景, 并与 #45036 的外部 Mooncake Store 方案形成对照——"#45036 provides SSD offload via the external Mooncake Store connector, this PR adds a native vLLM disk offloading path with zero external dependencies"。其性能卖点是 GPU cache 被淘汰后的 Reload 阶段: 100GB 磁盘替换 8GB DRAM 后 TTFT P50 从 15.8s 降到 1.5s。

实现拆解

实现按照「配置入口 -> 调度容量 -> worker 后端选择 -> 磁盘 IO 引擎 -> 安全加固」五步展开:

1. 配置入口与参数校验 (`vllm/distributed/kv_transfer/kv_connector/v1/simple_cpu_offload_connector.py`): 新增 `VALID_KV_OFFLOAD_BACKENDS = ("cpu", "disk")` 与 `_DISK_ONLY_KEYS`, 解析 `kv_offload_backend`、`disk_path`、`disk_capacity_bytes`、`disk_buffer_slots`、`use_page_cache` 五个新键。磁盘模式要求 `disk_path` 非空, 非法后端值直接 `raise ValueError`; 非磁盘模式下对 `disk` 专属键给出忽略警告。选择 `kv_offload_backend` 而非 `offload_backend` 是为了避免与 `OffloadConfig.offload_backend` (模型权重 offload) 语义冲突。
2. 调度端槽位容量切换 (`vllm/v1/simple_kv_offload/manager.py`): `SimpleCPUOffloadScheduler.__init__` 新增 `disk_capacity_bytes` 参数, 磁盘模式下 `offload_capacity = disk_capacity_bytes`, 否则回退到 `cpu_capacity_bytes`; 随后沿用 `_derive_cpu_config` 派生 CPU KV 配置并复用 `BlockPool/LRU` 与前缀匹配逻辑, 磁盘槽位与 CPU 块在调度层完全同构, 无需改动元数据格式。

3. Worker 端后端选择 (vllm/v1/simple_kv_offload/worker.py) : register_kv_caches 提取 unique_gpu_caches 与 total_bytes_per_block 后按 disk_mode 分支: init_cpu_mode 保持原有 pinned 内存分配与 DmaCopyBackend; _init_disk_mode 不再分配 CPU KV 张量, 而是按 disk_capacity_bytes // total_bytes_per_block 计算槽数, 构造 per-rank 文件路径 f"{disk_path}.rank{device.index or 0}" 并初始化 DiskBackend。两种后端共用 launch_copy 接口, get_finished 等调用点无需改动。
4. 磁盘后端核心实现 (vllm/v1/simple_kv_offload/disk_backend.py, 新增约 336 行) : DiskBackend 用两个独立线程分别处理 store 与 load (load 是延迟关键路径, 不被 store 阻塞), 各自持有独立 pinned staging buffer 与预构建的 memoryview/iovec 列表; _do_store/_do_load 实现双缓冲流水线——一个 slot 做 GPU DMA 时另一个 slot 并行执行 pwritev/preadv, 并借助 CUDA 事件保证顺序。init 时先 unlink 再以 O_EXCLIO_NOFOLLOWIO_DIRECT 打开文件并 ftruncate 预分配磁盘空间。
5. 安全与生命周期加固: 文件以 0o600 权限创建, 避免继承旧文件权限; _writev_slot/_readv_slot 检查 pwritev/preadv 返回值, 短写 / 短读显式抛 OSError; shutdown 时先 unlink 文件 (防止用户 prompt 残留磁盘), 若 IO 线程在 join 超时后仍存活则故意泄漏 fd 并告警, 避免 fd 复用竞态写穿到无关文件。

测试配套: 本 PR 未新增任何测试文件, ivanium 批准时明确要求后续补充 disk offload backend 的测试用例。

关键文件:

- vllm/v1/simple_kv_offload/disk_backend.py (模块 磁盘后端; 类别 source; 类型 core-logic; 符号 _alloc_aligned, DiskBackend, init, launch_copy) : 本 PR 的核心: 新增 DiskBackend 类, 实现 GPU->pinned 双缓冲 ->O_DIRECT 文件的全异步磁盘传输, store/load 独立线程、双缓冲流水线、对齐分配与安全加固均在此文件。
- vllm/v1/simple_kv_offload/worker.py (模块 KV 卸载; 类别 source; 类型 core-logic; 符号 SimpleCPUOffloadWorker.init, register_kv_caches, _init_disk_mode, _init_cpu_mode) : 在 register_kv_caches 中按 disk_mode 分支, 抽出 _init_disk_mode/_init_cpu_mode, 是后端切换的枢纽: 磁盘模式不分配 CPU KV 张量, 改为初始化 DiskBackend 并构造 per-rank 文件路径。
- vllm/distributed/kv_transfer/kv_connector/v1/simple_cpu_offload_connector.py (模块 KV 卸载; 类别 source; 类型 configuration; 符号 VALID_KV_OFFLOAD_BACKENDS, _DISK_ONLY_KEYS, SimpleCPUOffloadConnector.init) : 配置入口: 解析 kv_offload_backend 等新键, 做合法性校验、disk_path 必填检查与 cpu 模式下 disk 键忽略警告, 并把参数传给 scheduler 与 worker。
- vllm/v1/simple_kv_offload/manager.py (模块 KV 卸载; 类别 source; 类型 core-logic; 符号 SimpleCPUOffloadScheduler.init) : 调度器容量切换: disk_capacity_bytes > 0 时 offload 池大小取磁盘容量, 其余逻辑 (BlockPool/LRU、前缀匹配、元数据) 完全复用。

关键符号: DiskBackend.init, DiskBackend.launch_copy, DiskBackend._do_store, DiskBackend._do_load, DiskBackend.shutdown, SimpleCPUOffloadWorker._init_disk_mode, SimpleCPUOffloadWorker._init_cpu_mode, SimpleCPUOffloadScheduler.init

关键源码片段

vllm/v1/simple_kv_offload/worker.py

在 `register_kv_caches` 中按 `disk_mode` 分支，抽出 `_init_disk_mode/_init_cpu_mode`，是后端切换的枢纽：磁盘模式不分配 CPU KV 张量，改为初始化 `DiskBackend` 并构造 per-rank 文件路径。

```
# vllm/v1/simple_kv_offload/worker.py (关键节选)
```

```
class SimpleCPUOffloadWorker:
```

```
    def register_kv_caches(self, kv_caches: dict[str, torch.Tensor]) -> None:
```

```
        # 前面逻辑从 kv_caches 推断 unique_gpu_caches 与 total_bytes_per_block
```

```
        self.num_cpu_blocks = max(1, self.cpu_capacity_bytes // total_bytes_per_block)
```

```
        # 统一使用低优先级流，让 KV I/O 让位于计算流
```

```
        low_pri, _ = torch.cuda.Stream.priority_range()
```

```
        self.load_stream = torch.cuda.Stream(priority=low_pri)
```

```
        self.store_stream = torch.cuda.Stream(priority=low_pri)
```

```
        self.gpu_kv_caches = unique_gpu_caches
```

```
        if self.disk_mode:
```

```
            self._init_disk_mode(unique_gpu_caches, total_bytes_per_block, self.device)
```

```
        else:
```

```
            self._init_cpu_mode(unique_gpu_caches, total_bytes_per_block, self.device)
```

```
    def _init_disk_mode(
```

```
        self,
```

```
        unique_gpu_caches: dict[str, torch.Tensor],
```

```
        total_bytes_per_block: int,
```

```
        device: torch.device,
```

```
) -> None:
```

```
    # 磁盘模式不再分配常驻 CPU KV 张量，只保留少量 pinned 暂存 buffer
```

```
    num_disk_slots = max(1, self.disk_capacity_bytes // total_bytes_per_block)
```

```
    self.num_cpu_blocks = num_disk_slots
```

```
    assert self.disk_path is not None
```

```
    # 每个 rank 用独立文件，防止 TP 并发写同一路径导致数据错乱
```

```
    rank_path = f"{self.disk_path}.rank_{device.index or 0}"
```

```
    self._backend = DiskBackend()
```

```
    self._backend.init(
```

```
        unique_gpu_caches,
```

```
        device,
```

```
        self.load_stream,
```

```
        self.store_stream,
```

```
        rank_path,
```

```
        num_disk_slots,
```

```
        total_bytes_per_block,
```

```
        self.disk_buffer_slots,
```

```

        self.use_page_cache,
    )

def _init_cpu_mode(
    self,
    unique_gpu_caches: dict[str, torch.Tensor],
    total_bytes_per_block: int,
    device: torch.device,
) -> None:
    # 原有 CPU offload 路径: 分配 pinned CPU 张量 + DmaCopyBackend
    self.cpu_kv_caches = {}
    for name, gpu_tensor in unique_gpu_caches.items():
        cpu_shape = (self.num_cpu_blocks,) + gpu_tensor.shape[1:]
        # 先普通分配再 pin, 绕过 CUDACachingHostAllocator 的 2 的幂次取整
        tensor = torch.zeros(cpu_shape, dtype=gpu_tensor.dtype, device="cpu")
        if PIN_MEMORY:
            pin_tensor(tensor)
        self.cpu_kv_caches[name] = tensor

    self._backend = DmaCopyBackend()
    self._backend.init(
        unique_gpu_caches,
        self.cpu_kv_caches,
        device,
        self.load_stream,
        self.store_stream,
    )

```

评论区精华

核心讨论围绕性能归因、配置语义与安全加固展开：

- aoshen02 质疑 10.4x 提速是否主要来自容量差异 (8GB DRAM vs 100GB disk) ；作者补充了 32GB 对 32GB 的同容量对照：O_DIRECT 磁盘 reload P50 为 11.4s，优于无 offload 的 15.6s 但低于 DRAM 的 6.2s，且两者最终都受限于 PCIe 带宽，可通过多 NVMe 聚合收窄差距。
- ivanium 与 aoshen02 都好奇 page cache 模式 (3.4s) 为何比 DRAM offload (6.2s) 更快；作者解释共享 12K system prefix 时 preadv 可直接由内核缓存服务，路径更短，但 page cache 会占用目标场景本已稀缺的 DRAM，因此 use_page_cache 默认关闭。
- aoshen02 建议配置键改为 offload_backend；作者回应 kv_offload_backend 是为了避免与 OffloadConfig.offload_backend (模型权重 offload) 冲突，最终采用显式键并 fail-fast。
- aoshen02 提出 CPU + disk 双副本 (Mooncake 实践) ；作者认可这是正确 end state，但超出本 PR 范围，ivanium 建议以 RFC 跟踪。
- depthfirst-app[bot] 的安全审查覆盖 0o644 权限、O_EXCL/O_NOFOLLOW、短写检查、shutdown fd 竞态与文件残留，作者逐项修复并在提交 0230d02/4be4349 中落实。

- 性能提升来自容量还是磁盘本身？ (performance): 同容量下磁盘 reload P50 为 11.4s, 优于无 offload 的 15.6s 但低于 DRAM 的 6.2s; 两者最终受制于 PCIe 带宽, 可通过多 NVMe 聚合收窄差距。
- 为什么 page cache 比 DRAM offload 更快？ (performance): 作者解释 preadv 可直接由内核 page cache 服务, 共享 12K system prefix 时路径更短; 但 page cache 会占用目标场景稀缺的 DRAM, use_page_cache 默认关闭。
- 配置键命名: offload_backend vs kv_offload_backend (design): 采用 kv_offload_backend, 非法值 fail-fast, cpu 模式下忽略 disk 专属键并告警。
- CPU + disk 双副本架构 (Mooncake 实践) (design): 作者认可这是正确的 end state, 但超出本 PR 范围, 留待后续 RFC/issue 跟踪。
- KV 缓存文件权限与 symlink 防护 (security): 作者改为 0o600, 并采用先 unlink 后 O_EXCLIO_NOFOLLOW 打开, 保证 mode 由自己控制。
- 短写 / 短读与 IO 异常处理 (correctness): 短写 / 短读已加显式 OSError 检查; 但 IO 线程内异常没有显式捕获 (代码中未见 try/except), 线程抛错会退出并丢失事件, 有待后续加固。
- shutdown 时的 fd 生命周期与敏感文件清理 (security): shutdown 先 unlink 文件 (只删目录项, 匿名 inode 由内核在最后 fd 关闭时释放); 若线程仍存活则泄漏 fd 并告警。
- 缺少磁盘后端测试 (testing): 已合入, 测试留作 follow-up。

风险与影响

- 风险: 风险集中在以下方面:
- 缺少测试覆盖: 4 个变更文件均为源码, 无对应单测; DiskBackend 的 IO 线程、双缓冲流水线、O_DIRECT 对齐与 shutdown 竞态均未纳入 CI, 回归只能靠人工 benchmark 暴露。
- IO 线程异常未处理: _store_loop/_load_loop 内没有 try/except 包裹, pwritev/preadv 抛 OSError (如磁盘故障、短写) 会直接终止线程, 事件不会 append 到 events_list, worker 的事件高水位可能永久等待; depthfirst bot 提过类似问题, 但最终代码未显式捕获。
- O_DIRECT 依赖文件系统支持: O_DIRECT 在部分文件系统 (如 tmpfs、部分网络 FS) 不可用或行为不同, getattr(os, "O_DIRECT", 0) 在非 Linux 平台回落为 buffered IO, 行为差异未测试; total_block_bytes % 4096 的断言失败会直接崩溃。
- 数据明文落盘: KV 块可能编码用户 prompt, 虽已用 0o600 权限和 shutdown unlink 缓解, 但进程崩溃或 kill -9 时文件仍可能残留。
- 容量语义变化: 磁盘模式下 cpu_capacity_bytes 仍从 cpu_bytes_to_use 读取并参与日志, 但实际 offload capacity 由 disk_capacity_bytes 决定, 配置混乱时日志可能误导排障。
- 影响: 用户侧: 新增 kv_offload_backend="disk" 配置项, 默认 cpu 行为完全不变, 向后兼容; 启用后可在 DRAM 受限节点上以 NVMe 换显存, Reload 场景 TTFT 从 15.8s 降至 1.5s。

系统侧: 每个 rank 生成一个 .rank_N 文件 (默认容量 100GB), 调度器复用

BlockPool/LRU 与元数据格式, 改动面限定在 simple_kv_offload 子系统和 connector 配置解析; 新增两个常驻 IO 线程与 pinned 双缓冲, 线程生命周期与 shutdown 逻辑需要额外关注。

团队侧：为多级 offload (CPU + disk 双层、Mooncake 式副本) 打下基础，ivanium 明确希望以此为第一步并跟进 RFC；该 PR 也为后续磁盘后端测试、多 NVMe 带宽聚合预留了方向。

- 风险标记：缺少测试覆盖，IO 线程异常未处理，依赖文件系统 O_DIRECT 支持，KV 明文落盘，核心传输路径变更

关联脉络

- PR #45036 外部 Mooncake Store SSD offload 方案（标题未在材料中提供）：PR body 中明确对比的另一条 SSD offload 路径：依赖外部 Mooncake Store，本 PR 是其零依赖原生替代。
- PR #51100 [Bugfix] Fix Mamba all-mode CPU offload boundary alignment: 同属 kv-connector/v1/simple_kv_offload 子系统，修改 scheduler 与 offloading 连接器的对齐逻辑，与本 PR 的磁盘槽位容量管理在同一代码区域。
- PR #51046 [CI] Exclude KV-connector subtree from broad source dependencies: 调整 KV-connector 子树的 CI 依赖范围，说明该子系统最近在独立演进，本 PR 的 disk 后端也应纳入后续 CI 测试范围。