PR #2739 完整报告

radixark/miles

feat: dist_muon offloading in megatron

合并时间：2026-08-29 09:16

原文链接：http://prhub.com.cn/radixark/miles/pull/2739

执行摘要

- 一句话：为 Muon 优化器新增 NVMe 磁盘状态卸载，规避主机 OOM
- 推荐动作：值得精读。核心设计决策有三个：其一，不重写 DistOpt 分桶 / 写回逻辑，而是子类化 ChunkedOptimizerStateOffloader 只覆盖分配器，用最小改动复用 Megatron 既有的 chuncked restore、assert_master_weights_resident 和 prefetch 钩子；其二，利用文件页可回收性而不是简单扩内存，从内核语义上解决 OOM；其三，用 tensor 标记 + msync 与 Megatron checkpoint 协议对齐，避免全量重分配和不可归因的写回。这些模式对任何"把显存 / 内存压力转移到磁盘"的需求都有借鉴价值。

功能与动机

PR body 明确指出原有 --stream-optimizer-state-to-disk 只对 Adam 生效：该标志绑定 DistributedOptimizer，而 Muon 通过 LayerWiseDistributedOptimizer 构建，set_default_megatron_args 会为非 Adam 优化器清掉 use_distributed_optimizer，导致守护该标志的两个 assert 对 Muon 天然不可达。更深层的动机是文件页的内存语义：文件支撑的页落在 cgroup 可回收的 file 中而非 anon，内核能在压力下驱逐而不是 OOM 杀掉训练进程。实测 pinned CPU 路径在 919GB 限制下峰值 908GB 于 step 6 被 SIGTERM，磁盘后端则 rc=0 跑完，anon 稳定在 86GB 而 file 增长到 619GB。

实现拆解

1. 确定复用路径而非移植 DistOpt store：在 miles/backends/megatron_utils/model.py 的 setup_model_and_optimizer 中，当 _is_muon_optimizer(config.optimizer) 且开启 --stream-optimizer-state-to-disk 时，调用 setup_muon_state_on_disk(args)；同时把原本的 Adam store 接线条件改为 not _is_muon_optimizer(...)，避免两条路径同时挂载。设计上放弃把 DistOpt 的分桶规划、main 参数写回和 master 权重驻留协议移植到 Muon，而是复用 Float16OptimizerWithFloat16Params 已有的 _optimizer_state_offloader 槽位（ChunkedOptimizerStateOffloader 填充）。
2. 实现文件后端分配器：在 miles_plugins/optimizers/nvme_stream.py 中新增 setup_muon_state_on_disk，它子类化 ChunkedOptimizerStateOffloader，仅覆盖其 _new_cpu_buffer 分配器；新增 _disk_backed_like 用 tempfile.mkstemp 建文件、_reserve（posix_fallocate 预留物理块，失败回退 ftruncate）后通过 torch.UntypedStorage.from_file(path, shared=True) 映射并在映射后 os.unlink，返回带 _miles_disk_backed 标记的 tensor。copy_ 语义与 pinned 内存一致，而 is_pinned() 为

False 使继承代码走同步拷贝路径；另新增 _flush_mapping 通过 ctypes 调用 msync 把映射的脏页主动刷出，使 checkpoint 写入成本可归因。

3. 拆分参数栅栏与文案：在 miles/utils/arguments.py 的 miles_validate_args 中，按优化器家族拆分原有两个 assert：Muon 要求 --optimizer 带 dist_ 前缀（因为 Megatron 的 validate 尚未运行，不能依赖 use_layer_wise_distributed_optimizer）、--chunked-optimizer-state-offload 且 --optimizer-state-offload-fraction > 0；Adam 保留原要求。同时重写 --stream-optimizer-state-to-disk 的 help，分别描述 adam 与 dist_muon 两种语义，并在 --offload-train-disk-dir 帮助中补充 Muon 缓冲区 unlink 后 df 可见而 du 不可见的说明。
4. 修复 checkpoint 交互并配套测试：review 发现 adopt_cpu_optimizer_state 会重分配 optimizer.state 中所有非 pinned CPU tensor，而磁盘映射从不报告 pinned，导致每个 checkpoint 全量拷贝整个 offloaded 状态；修复为缓冲区带标记、分配器原样返回，并在 synchronize_for_checkpoint 时先 msync。测试配套包括 tests/fast/optimizers/test_nvme_stream.py（8 个用例，覆盖 shape/dtype/round-trip/unlink 后无残留 / 标记识别 /msync 重复廉价 /_reserve 尺寸 / 多 dtype，无需 GPU）以及 tests/e2e/megatron/test_qwen3_4B_muon_offload_disk.py（镜像 Adam 的 test_qwen3_4B_offload_disk_stream.py，用 --optimizer dist_muon，断言 4 个 rank 的 worker 日志都出现 Muon disk state step:，防止磁盘后端静默退回 pinned 内存）。

关键文件：

- miles_plugins/optimizers/nvme_stream.py（模块 优化器；类别 source；类型 dependency-wiring；符号 _allocate_file, _reserve, _disk_backed_like, _is_disk_backed）：核心实现文件：新增 setup_muon_state_on_disk、_disk_backed_like、_is_disk_backed、_flush_mapping、_reserve 等符号，把 Muon 的 chunked offloader 分配器换成文件 mmap，并处理 checkpoint 交互与 msync 写回。
- tests/e2e/megatron/test_qwen3_4B_muon_offload_disk.py（模块 E2E 测试；类别 test；类型 test-coverage；符号 prepare, _assert_disk_backed_steps, _assert_offloaded_to_disk, execute）：新增 e2e 测试，镜像 Adam 磁盘 offload 测试并改用 --optimizer dist_muon，核心价值是断言 4 个 rank 都真实记录磁盘 step，防止磁盘后端静默退回 pinned 内存。
- tests/fast/optimizers/test_nvme_stream.py（模块 单元测试；类别 test；类型 test-coverage；符号 test_disk_buffer_matches_shape_and_dtype_and_is_not_pinned, test_disk_buffer_round_trip_is_bit_exact, test_disk_buffer_leaves_no_file_behind, test_disk_buffer_is_recognized_as_already_managed）：快速单测（无需 GPU），专门守护静默失败路径：若分配器不再返回文件后备存储，测试能第一时间暴露。
- miles/utils/arguments.py（模块 参数解析；类别 source；类型 core-logic）：参数校验与帮助文案：--stream-optimizer-state-to-disk 的 gate 按优化器家族拆分（Adam 需分布式优化器，Muon 需 dist_ 前缀 + chunked offload 开关 + 非零 fraction）。
- miles/backends/megatron_utils/model.py（模块 后端接线；类别 source；类型 data-contract）：接线入口：Muon 分支在优化器构建前调用 setup_muon_state_on_disk，并让 Adam 的 setup_optimizer_state_streaming 跳过 Muon，避免两条 offload 路径冲

突。

- tests/fast/optimizers/__init__.py（模块 测试包；类别 test；类型 test-coverage）：为 tests/fast/optimizers 包提供初始化文件，使新的 fast 测试可被测试发现机制收集。

关键符号：setup_muon_state_on_disk, _disk_backed_like, _is_disk_backed, _flush_mapping, _reserve, _allocate_file, _assert_disk_backed_steps, _assert_offloaded_to_disk

关键源码片段

miles_plugins/optimizers/nvme_stream.py

核心实现文件：新增 setup_muon_state_on_disk、_disk_backed_like、_is_disk_backed、_flush_mapping、_reserve 等符号，把 Muon 的 chunked offloader 分配器换成文件 mmap，并处理 checkpoint 交互与 msync 写回。

```
# miles_plugins/optimizers/nvme_stream.py 中文件后端分配器的核心实现

def _reserve(fd: int, nbytes: int) -> None:
    """先预留物理块，让磁盘写满在分配期以 ENOSPC 暴露。

    如果只用 ftruncate 撑大文件，映射会成功但页是稀疏的，
    进程在首次触碰时才因 SIGBUS 死亡，且没有任何可指向的错误。
    """
    try:
        os.posix_fallocate(fd, 0, nbytes) # 真正占住物理块
    except OSError as e:
        # 不支持 fallocate 的文件系统退回 ftruncate，此时稀疏风险仍在
        if e.errno not in (errno.EOPNOTSUPP, errno.ENOTSUP, errno.EINVAL):
            raise
        os.ftruncate(fd, nbytes)


def _allocate_file(path: str, nbytes: int) -> int:
    # 打开文件后立刻预留物理块，供 Adam 的 read/write 路径使用
    fd = os.open(path, os.O_RDWR | os.O_CREAT | os.O_CLOEXEC, 0o600)
    _reserve(fd, nbytes)
    return fd


def _disk_backed_like(tensor: torch.Tensor, directory: str) -> torch.Tensor:
    # 按被替换 tensor 的字节数建文件，并映射为同 shape/dtype 的 CPU buffer
    nbytes = max(tensor.numel() * tensor.element_size(), 1)
    fd, path = tempfile.mkstemp(dir=directory, suffix=".bin")
    try:
        _reserve(fd, nbytes) # 先预分配，避免后续写映射时 SIGBUS
    finally:
        os.close(fd)
    storage = torch.UntypedStorage.from_file(path, shared=True, nbytes=nbytes)
```

```
    os.unlink(path) # 映射已建立，立刻删除目录项，进程退出不残留文件
    buffer = torch.empty(0, dtype=tensor.dtype).set_(storage, 0, tensor.shape)
    buffer._miles_disk_backed = True # 标记：checkpoint adoption 应原样保留而非重分配
    return buffer


def _is_disk_backed(tensor: torch.Tensor) -> bool:
    # 供同步与 checkpoint 逻辑识别磁盘缓冲区
    return getattr(tensor, "_miles_disk_backed", False)


# msync 常量与 libc 句柄：_MS_SYNC = 4 对应 msync 的 MS_SYNC 标志
_MS_SYNC = 4
_libc = ctypes.CDLL(None, use_errno=True)


def _flush_mapping(tensor: torch.Tensor) -> int:
    """msync 一个文件后备 buffer，返回它覆盖的字节数。

    checkpoint 会对自己的文件调用 os.fsync，而 os.fsync 等待内核 writeback
    队列——我们的映射每个 step 都在重写，到保存时队列里已经积压了数 GB
    脏页。这里主动 msync 把成本归因到本模块，且对已干净映射的重复调用
    会立即返回，所以重复冲刷很便宜。
    """
    storage = tensor.untyped_storage()
    nbytes = storage.nbytes()
    # msync 失败（如 EIO）时上抛 OSError，让保存路径显式失败而不是静默丢数据
    if _libc.msync(ctypes.c_void_p(storage.data_ptr()), ctypes.c_size_t(nbytes), _MS_SYNC) != 0:
        raise OSError(ctypes.get_errno(), "msync of optimizer state mapping failed")
    return nbytes
```

tests/e2e/megatron/test_qwen3_4B_muon_offload_disk.py

新增 e2e 测试，镜像 Adam 磁盘 offload 测试并改用 --optimizer dist_muon，核心价值是断言 4 个 rank 都真实记录磁盘 step，防止磁盘后端静默退回 pinned 内存。

```
# tests/e2e/megatron/test_qwen3_4B_muon_offload_disk.py 的关键断言


def _assert_disk_backed_steps():
    """每个 rank 都必须真正用文件后端执行过优化器 step。

    完成训练本身证明不了什么：如果磁盘后端静默失效，训练会照样
    在 pinned 主机内存上进行并收敛，所以必须核对日志证据。
    """
    logs = glob.glob("/tmp/ray/session_latest/logs/worker-*")
    assert logs, "no Ray worker logs to check for the disk-backed state path"

    backed = set()
    for path in logs:
        with open(path, errors="ignore") as f:
```

```
            # 每 step 一行的日志是磁盘后端生效的证据
            if any("Muon disk state step:" in line for line in f):
                backed.add(path)

    assert (
        len(backed) == NUM_GPUS
    ), f"expected {NUM_GPUS} ranks to log disk-backed optimizer steps, saw {len(backed)}:
    {sorted(backed)}"
    print(f"Muon optimizer state was file-backed on {len(backed)} ranks")


def _assert_offloaded_to_disk():
    """每个 rank 都必须在自己的目录下 arm 了暂停 actor 的磁盘 offload。"""
    logs = glob.glob("/tmp/ray/session_latest/logs/worker-*")
    assert logs, "no Ray worker logs to check for the disk-offload path"

    armed = set()
    for path in logs:
        with open(path, errors="ignore") as f:
            for line in f:
                if "Train disk-offload reclaim armed" in line:
                    armed.add(line.split("reclaim armed for ")[1].split()[0])

    # 期望每个 rank 都在自己的目录下 arm，目录名形如 cell0_rank<N>
    expected = {os.path.join(OFFLOAD_DIR, f"cell0_rank{rank}") for rank in range(NUM_GPUS)}
    assert armed == expected, f"expected disk offload armed for {sorted(expected)}, saw
    {sorted(armed)}"
```

评论区精华

核心讨论集中在三处，均由 yueming-yuan 提出并最终 approve：

- --optimizer-state-offload-chunk-size-mb 默认值为 0，在 Megatron 中 0 表示非 streaming，峰值内存无法节省，用户可能无感知；作者回应“Megatron has this warning already”，维持默认值，结论为不修改。
- checkpoint 路径的 adopt_cpu_optimizer_state 会把所有非 pinned CPU tensor 整体重分配，导致每次保存全量拷贝 offloaded 状态，_disk_bytes 永久膨胀；修复方式是给磁盘缓冲区加 _miles_disk_backed 标记，让分配器原样返回。
- UntypedStorage.from_file(shared=True) 只做 ftruncate 不预留物理块，磁盘容量不足时进程会以 SIGBUS 死亡而非得到可操作的 ENOSPC；修复为在 mmap 前用 posix_fallocate 预留（即 _reserve）。此外 claude[bot] 还指出 os.unlink 仅在 from_file 成功后执行，ENOSPC/ 权限错误会泄漏 mkstemp 临时文件，属于窄错误路径问题，未阻塞合并。
- --optimizer-state-offload-chunk-size-mb 默认 0 可能导致用户误以为 streaming 已启用 (question): 作者回复“Megatron has this warning already”，维持默认值不变，靠 Megatron 既有警告兜底。

- checkpoint adoption 会把 mmap 状态整体重分配，导致每次保存全量拷贝 (correctness): 缓冲区携带 _miles_disk_backed 标记，分配器对已管理 buffer 原样返回；实测 4x H200 三次保存下 offloaded 总量 3.38 -> 6.77 GB 后保持稳定，证明不再重复拷贝。
- UntypedStorage.from_file 未预留物理块，容量不足会 SIGBUS 而非 ENOSPC (correctness): 新增 _reserve 在 mmap 前调用 posix_fallocate 预留物理块（不支持时回退 ftruncate），磁盘耗尽在分配期以 ENOSPC 暴露。
- mkstemp 错误路径文件泄漏与 Muon 分支日志借用 Adam 文案 (other): 该错误路径泄漏被判定为窄风险、非正常路径正确性问题，未阻塞合并；日志文案问题已作为新 inline 评论提出，最终未在本 PR 中看到修复。

风险与影响

- 风险：
  1. Megatron 内部接口依赖：整个方案挂在 Float16OptimizerWithFloat16Params._optimizer_state_offloader 槽位和 ChunkedOptimizerStateOffloader 的 _new_cpu_buffer 分配器上，属于 Megatron 内部契约；后续升级 Megatron（如最近 flash-linear-attention、sglang 的连带升级）若调整 offloader 构造或分配器签名，nvme_stream.py 会静默失效或直接报错。
  2. checkpoint 性能回退：实测磁盘后端每次保存 381s，对比 pinned 内存的 229s 约慢 66%，原因是状态需经 mmap 读回；且 msync 只是把写回成本归因，I/O 总量未减少。
  3. CI 覆盖缺口：三保存的 e2e 需 19 分钟，超过 run_suite.py 1800s 单文件限制，因此 e2e 测试不带 checkpoint 保存，checkpoint 回归只能靠人工验证。
  4. 静默降级风险：若分配器覆盖失效，offloader 会退回 pinned 主机内存照常训练，只有日志与测试断言能暴露；fast 测试专门守护这一路径。
  5. 磁盘语义副作用：文件 unlink 后仍在写，占用在 df 可见而 du 不可见，运维排查磁盘占用时容易困惑；posix_fallocate 把 ENOSPC 提前到分配期，但底层文件系统若不支持会回退 ftruncate，稀疏映射的 SIGBUS 风险在该文件系统上依然存在。 - 影响：对用户：Muon 训练大规模模型（如 Qwen3.5-35B-A3B 级别）时可把优化器状态放在节点本地 NVMe，避免主机内存 OOM/SIGTERM，这是此前 pinned CPU 路径无法覆盖的场景。对系统：主机 anon 内存占用显著下降（实测 908GB 峰值降至 86GB 稳定），内存压力转由内核可回收的 file 页承担；代价是每个 step 的磁盘流量和 checkpoint 保存耗时增加约 66%。对团队：新增一种 e2e 测试模式——不只断言训练完成，还通过 worker 日志断言功能真实生效（防止静默降级），并对指标 gate 失效场景（rollout 全截断导致 gate 恒为零）做了明确说明，可复用到其他 offload 类特性。 - 风险标记：依赖 Megatron 内部 offloader 槽位，checkpoint 保存耗时 +66%，e2e 测试无 checkpoint 覆盖，磁盘容量不足时存在 SIGBUS 回退风险，错误路径存在临时文件泄漏

关联脉络

- PR #2790 Bump flash-linear-attention to 0.5.2: 同为 Megatron 训练栈的崩溃修复与镜像配套，与本 PR 共享 Megatron 优化器 / 训练链路，Megatron 依赖升级可能影响本 PR 所依赖的 offloader 内部接口。

- PR #2743 Skip non-numeric rewards in the episode average: 同为训练主路径上的可靠性修复（避免 OPD 训练崩溃），本 PR 解决的是另一个崩溃来源（主机 OOM/SIGTERM），两者共同保障大规模 PPO 训练的稳定性。
- PR #2714 Bump sglang to v0.5.18: rollout 后端 sglang 升级，本 PR 的 e2e 测试依赖 sglang + megatron 的完整运行链路，镜像了同系列 offload 测试的运行环境。