

PR #2842 完整报告

radixark/miles

[multi-lora v2 3/7] Per-slot checkpoints with optimizer state

合并时间: 2026-09-16 16:25

原文链接: <http://prhub.com.cn/radixark/miles/pull/2842>

执行摘要

- 一句话: 新增多 LoRA 单槽位检查点, 含 Adam 状态与原子版本发布
 - 推荐动作: 值得精读, 但读法有侧重:
1. 优先看 `miles/backends/training_utils/checkpoint_io.py`: `publish_dir()` 里「`rename` 到 `_version_*` → 建相对符号链接 → `os.replace` 顶替公共路径」三步是整个栈崩溃安全性的基石, 也解释了为什么必须保留旧版本目录; 同时注意 `make_tmp_dir` 的三种拒绝路径 (`non-overwrite`、`legacy` 目录、崩溃残留)。
 2. 再看 `miles/backends/megatron_utils/lora/checkpoint.py`: `_canonicalize_slot_keys` 与 `load_optimizer` 两条分支 (`load_sharded_state` vs `reload_masters`) 是「槽位无关 + 槽位本地」两个设计意图的落点, 也是评审意见 1/2 的修复位置。
 3. 值得追踪的遗留项: IO 失败隔离 TODO、`_version_*` 回收策略、rank 局部失败的集体错误广播、docstring 与 PR body 关于拓扑约束的口径统一, 以及在 TP/PP/EP/MoE 布局下的恢复验证。
 4. 可借鉴的设计决策: 「先校验后赋值」的两段式加载、把跨槽不变量固化成 key 归一化、用负向对照 (master 舍入到 BF16 / 纯权重重启必须失败) 证明测试确实是在测正确的东西——这三点在同类持久化改动中都值得复用。

功能与动机

PR body 明确目标是「Save and restore one adapter slot independently」: 多租户 Tinker 场景下每个租户独占一个适配器槽位, 需要能单独落盘并在之后原样续训, 因此检查点不能只存权重, 还必须包含「adapter weight shards, per-rank Adam state and step counters, and FP32 master parameters」。作者同时点出关键约束——「loading optimizer state requires the same parallel topology」。该能力是后续章节的地基: 4/7 的适配器快照发布、5/7 的网关协议面与 6/7 的运行时都需要一个崩溃安全的检查点发布语义, 而旧实现 (`slots.py` 里的 `megatron_shard_name/adapter_shard_topology` 手工分片文件) 既不原子、也存在跨槽加载隐患, 因此本 PR 将其整体替换。

实现拆解

1. 新增检查点 IO 原语: 集体写入 + 原子发布 (`miles/backends/training_utils/checkpoint_io.py`, 新增 70 行) - `write_checkpoint_dir(path, write_shards, metadata=None, *, overwrite=True)` 统一编排 `make_tmp_dir()` → `_barrier()` → `write_shards(tmp_dir)` →

`_barrier()` → `publish_dir()` → `_barrier()`，要求所有 rank 参与。 - 发布阶段把临时目录改名为 `_version_<name>_<uuid>`，再创建一个同名相对符号链接并用 `os.replace` 顶替公共路径；因此公共路径任一瞬间都指向「旧的完整版本」或「新的完整版本」，旧版本目录被保留给已解析过它的读者。 - `overwrite=False` 时对已存在路径抛 `FileExistsError`；对「老式目录型」检查点（非符号链接）抛 `NotImplementedError`，提示换名字保存。文件头留了显式 TODO：IO 失败目前会直接终止 trainer cell。

2. 单槽位保存 / 装载核心逻辑 (`miles/backends/megatron_utils/lora/checkpoint.py`, 新增 61 行) - `_slot_weights_sharded_state_dict(model, slot)` 汇合各 chunk 的 `sharded_state_dict()`，按 `.adapters.<slot>` 过滤出本槽位张量，并用断言保证不会写出空检查点。 - `_canonicalize_slot_keys(tree, slot)` 把存储 key 中的 `.adapters.<slot>` 改写成槽位无关的 `.adapter.`，这是「slot 0 存、slot 1 载」能成立的前提。 - `save_slot` 把 `_WEIGHTS_KEY` (adapter 权重) 与 `_OPTIM_KEY` (`SlotOptimizer.sharded_state(..., is_loading=False)`) 合成一棵树后交给 `dist_checkpointing.save`。 - `load_slot(model, slot_optimizer, path, load_optimizer)` 先构造壳 (`is_loading=True`) 再 `dist_checkpointing.load`，成功后才 `load_sharded_state`；`load_optimizer=False` 走权重路径并调用 `reload_masters()`。
3. 优化器侧补槽位本地的 sharded 接口 (`miles/backends/megatron_utils/lora/optimizer.py`, +7) - 新增 `SlotOptimizer.sharded_state(model_sharded_state_dict, *, is_loading)` 与 `load_sharded_state(loaded)`，内部转发到该槽位自己的 chained optimizer (`self._inner`)。 - 这样装载只影响目标槽位，避免早期实现通过外层 LayerWise/Chained optimizer 重载、进而把所有活跃槽位的 FP32 master 从 BF16 权重重新舍入（对应评审的第 1 条意见）。
4. 控制面 API 打通：TrainerCell 与 Ray 训练组 (`miles/backends/megatron_utils/lora/actor.py` +14/-3; `miles/ray/train/group.py` +11/-4) - `MultiLoRATrainRayActor.load_slot` 增加 `ckpt_path` 与 `load_optimizer` 参数，装载成功后按需调用 `lora_checkpoint.load_slot`；新增 `save_slot(slot, path, metadata)`；`unload_slot` 返回值改为 `dict | None`（新增的槽位命令转发契约）。 - `TrainGroup` 的 `load_slot/save_slot/unload_slot` 统一经 `_execute_slots` 转发，并把返回类型从 `None` 收敛为 `list`，为 5/7 的协议层提供稳定签名。
5. 下线旧的文件式分片机制并补齐测试 - `miles/backends/megatron_utils/lora/slots.py` 删除 `megatron_shard_name` 与 `adapter_shard_topology` (-35)，`adapter_megatron_tp*_pp*[_ep*].pt` 手工分片命名与一次性 gloo all-gather 拓扑探测随 `dist_checkpointing` 方案整体退役。 - 新增 `e2e tests/e2e/lora/test_slot_checkpoint_resume.py` (+180)，通过 `register_cuda_ci(est_time=120, suite="stage-b-2-gpu-h200", labels=["lora"], hardware=["hopper"])` 注册为 2 卡 Hopper 用例，在真实 Megatron DDP + MultiLoRALinear + SlotOptimizer 上做 3 步预热后保存、跨槽恢复并比对权重 / FP32 master / Adam moment / step 计数，另含两个负向对照（master 舍入到 BF16、纯权重重启）。 - 新增 fast 单测 `tests/fast/backends/megatron_utils/test_slot_checkpoint_keys.py` (+13)，用 `SimpleNamespace(key=...)` 直接守护「保存槽位与装载槽位在 canonical key 上相遇」这一不变量。

关键文件：

- `miles/backends/megatron_utils/lora/checkpoint.py`（模块 适配器检查点；类别 source；类型 core-logic；符号 `save_slot`, `load_slot`, `_slot_weights_sharded_state_dict`,

_canonicalize_slot_keys)：本 PR 的核心逻辑：save_slot/load_slot 把 adapter 权重与槽位优化器状态合成一致检查点，_canonicalize_slot_keys 提供跨槽装载能力；评审的 1/2 两条正确性意见最终都落在这个文件的实现形态上（改用 dist_checkpointing 的「先校验后赋值」，并配合槽位本地 reload）。

- miles/backends/training_utils/checkpoint_io.py（模块 检查点发布；类别 source；类型 core-logic；符号 write_checkpoint_dir, make_tmp_dir, publish_dir, _rank）：新增的集体写入 + 原子发布原语，是评审第 3 条意见（发布窗口崩溃导致公共路径消失）的直接产物，也是后续 4/7 快照发布复用的基础语义；文件头的 IO 失败 TODO 是本次改动留下的主要技术债。
- tests/e2e/lora/test_slot_checkpoint_resume.py（模块 端到端测试；类别 test；类型 test-coverage；符号 CheckpointModel, snapshot, assert_same_state, apply_step）：评审第 4 条意见要求补正向 round-trip 与跨槽污染断言，该文件是最终交付物：在真实 Megatron DDP + MultiLoRALinear + SlotOptimizer 上做 3 步预热后跨槽恢复，并含两个负向对照证明测试确实在测正确性。
- miles/backends/megatron_utils/lora/optimizer.py（模块 槽位优化器；类别 source；类型 core-logic；符号 sharded_state, load_sharded_state）：新增 sharded_state/load_sharded_state 两个槽位本地接口，是修复「装载一个槽位会重舍入所有活跃槽位 FP32 master」的关键落点，为 lora/checkpoint.py 提供优化器侧状态视图。
- miles/backends/megatron_utils/lora/actor.py（模块 适配器训练；类别 source；类型 core-logic；符号 load_slot, save_slot, unload_slot）：训练侧入口：load_slot 扩展 ckpt_path/load_optimizer，新增 save_slot，并把 unload_slot 返回值改为 dict | None；检查点执行失败在这里冒泡到 TrainerCell（对应 PR body 的失败语义）。
- miles/ray/train/group.py（模块 训练组接口；类别 source；类型 core-logic；符号 load_slot, save_slot, unload_slot）：控制面转发层：load_slot 新增 ckpt_path/load_optimizer，新增 save_slot，并把槽位命令返回类型统一为 list，是后续 Tinker 协议层调用检查点能力的契约面。
- miles/backends/megatron_utils/lora/slots.py（模块 槽位管理；类别 source；类型 refactor；符号 megatron_shard_name, adapter_shard_topology）：删除 megatron_shard_name 与 adapter_shard_topology（-35 行），标志旧的手工分片文件方案整体退役，由 torch_dist 格式取代，是理解本次方案切换的关键对照。
- tests/fast/backends/megatron_utils/test_slot_checkpoint_keys.py（模块 键名测试；类别 test；类型 test-coverage；符号 test_saved_and_loading_slots_meet_at_the_canonical_key）：以 13 行单测固化「保存槽位与装载槽位在 canonical key 上相遇」这一跨槽装载不变量，是 _canonicalize_slot_keys 唯一直接守护。

关键符号：write_checkpoint_dir, make_tmp_dir, publish_dir, save_slot, load_slot, _slot_weights_sharded_state_dict, _canonicalize_slot_keys, SlotOptimizer.sharded_state, SlotOptimizer.load_sharded_state, SlotOptimizer.reload_masters, MultiLoRATrainRayActor.save_slot, TrainGroup.save_slot, run_worker, assert_same_state

关键源码片段

miles/backends/megatron_utils/lora/checkpoint.py

本 PR 的核心逻辑：`save_slot/load_slot` 把 adapter 权重与槽位优化器状态合成一致检查点，`_canonicalize_slot_keys` 提供跨槽装载能力；评审的 1/2 两条正确性意见最终都落在这个文件的实现形态上（改用 `dist_checkpointing` 的「先校验后赋值」，并配合槽位本地 reload）。

```
"""Per-slot checkpoints in megatron's torch_dist format.
```

```
Every tensor carries its global shard coordinates and a slot-agnostic key,  
so a checkpoint reloads under any tp/pp/ep/world-size layout and into any  
free slot."""
```

```
from collections.abc import Sequence  
from pathlib import Path
```

```
from megatron.core import dist_checkpointing  
from megatron.core.dist_checkpointing.dict_utils import nested_values  
from megatron.core.distributed import DistributedDataParallel as DDP  
from megatron.core.utils import unwrap_model
```

```
from miles.backends.megatron_utils.lora.optimizer import SlotOptimizer  
from miles.backends.training_utils.checkpoint_io import write_checkpoint_dir
```

```
_WEIGHTS_KEY = "adapter_weights"  
_OPTIM_KEY = "adapter_optimizer"
```

```
def _slot_weights_sharded_state_dict(model: Sequence[DDP], slot: int) -> dict:  
    # 汇合每个 chunk 的 sharded state dict，再按 ".adapters.<slot>." 只挑出本槽位的 adapter  
    # 张量；  
    # 每个张量自带全局 shard 坐标，因此具备跨 tp/pp/ep 布局重建的能力  
    sharded: dict = {}  
    for chunk in unwrap_model(model):  
        sharded |= chunk.sharded_state_dict()  
    marker = f".adapters.{slot}."  
    slot_sharded = {key: value for key, value in sharded.items() if marker in key}  
    # 一个张量都没匹配到说明 slot 索引或模型装配有问题，立即失败而不是写出空检查点  
    assert slot_sharded, f"slot {slot} exposed no adapter tensors"  
    return slot_sharded
```

```
def _canonicalize_slot_keys(tree: dict, slot: int) -> dict:  
    """Rewrite storage keys so a checkpoint saved from slot i loads into slot j."""  
    # 把 ".adapters.<slot>." 统一改写成槽位无关的 ".adapter.",  
    # 于是保存端（slot i）与装载端（slot j）在同一个 canonical key 上相遇；  
    # 该不变量由 tests/fast/backends/megatron_utils/test_slot_checkpoint_keys.py 单独守护  
    marker, canonical = f".adapters.{slot}.", ".adapter."  
    for sharded in nested_values(tree):  
        if hasattr(sharded, "key"):
```

```

        sharded.key = sharded.key.replace(marker, canonical)
    return tree

```

```

def save_slot(model: Sequence[DDP], slot_optimizer: SlotOptimizer, path: str, metadata: dict |
None = None) -> None:
    weights = _slot_weights_sharded_state_dict(model, slot_optimizer.slot)
    # is_loading=False 让优化器给出「可写入」的 sharded 状态:
    # 内含 FP32 master、Adam 的 exp_avg/exp_avg_sq 以及 param_groups 中的 step 计数
    sharded = {_WEIGHTS_KEY: weights, _OPTIM_KEY: slot_optimizer.sharded_state(weights, is_
loading=False)}
    _canonicalize_slot_keys(sharded, slot_optimizer.slot)
    # 集体写入 + 原子发布; 任一 rank 的 IO 异常都会向上抛到 TrainerCell
    write_checkpoint_dir(path, lambda tmp_dir: dist_checkpointing.save(sharded, str(tmp_dir)),
metadata=metadata)

```

```

def load_slot(model: Sequence[DDP], slot_optimizer: SlotOptimizer, path: str, load_optimizer:
bool) -> None:
    checkpoint_dir = str(Path(path))
    weights = _slot_weights_sharded_state_dict(model, slot_optimizer.slot)
    shells = {_WEIGHTS_KEY: weights}
    if load_optimizer:
        # is_loading=True 只构造与本地布局对齐的「空壳」, 真实赋值发生在 load 内部
        shells[_OPTIM_KEY] = slot_optimizer.sharded_state(weights, is_loading=True)
        _canonicalize_slot_keys(shells, slot_optimizer.slot)

    # 先整体加载 (含 key/ 形状校验) 再赋值: 坏分片不会半途污染正在训练的槽位
    loaded = dist_checkpointing.load(shells, checkpoint_dir)
    if load_optimizer:
        slot_optimizer.load_sharded_state(loaded[_OPTIM_KEY])
    else:
        # 纯权重装载保留槽位初始化刚建好的 Adam 状态, 只把 FP32 master 按新权重重算
        slot_optimizer.reload_masters()

```

miles/backends/training_utils/checkpoint_io.py

新增的集体写入 + 原子发布原语, 是评审第 3 条意见 (发布窗口崩溃导致公共路径消失) 的直接产物, 也是后续 4/7 快照发布复用的基础语义; 文件头的 IO 失败 TODO 是本次改动留下的主要技术债。

```

"""Checkpoint directories: written collectively, complete at their final path."""

```

```

# TODO: 隔离 checkpoint IO 失败; 目前它们会直接终止 trainer cell。

```

```

import json
import os
import shutil
from collections.abc import Callable
from pathlib import Path

```

```

from uuid import uuid4

import torch.distributed as dist

from miles.utils.distributed_utils import get_gloo_group


def write_checkpoint_dir(
    path: str | Path,
    write_shards: Callable[[Path], None],
    metadata: dict | None = None,
    *,
    overwrite: bool = True,
) -> None:
    """Write collectively, then atomically point ``path`` at the completed version.

    All ranks must call. Readers may still hold an older version, so retain it.
    """
    final_dir = Path(path)
    # 临时目录名字固定（非随机），崩溃残留由 make_tmp_dir 负责清理
    tmp_dir = final_dir.parent / f"_tmp_{final_dir.name}"

    def make_tmp_dir():
        if _rank() == 0:
            if not overwrite and final_dir.exists():
                raise FileExistsError(f"checkpoint {final_dir} already exists")
            # 老式「目录型」检查点无法安全原地替换，只能换名字保存（只读兼容）
            if final_dir.exists() and not final_dir.is_symlink():
                raise NotImplementedError(
                    f"cannot overwrite a legacy checkpoint directory {final_dir}; save under a new name"
                )
            # a crashed attempt may leave shards or an unpublished version link
            # 上一次崩溃可能留下半成品分片或未发布的版本链接，先清干净再写
            if tmp_dir.is_symlink():
                tmp_dir.unlink()
            elif tmp_dir.exists():
                shutil.rmtree(tmp_dir)
            tmp_dir.mkdir(parents=True)

    def publish_dir():
        if _rank() != 0:
            return
        if metadata is not None:
            (tmp_dir / "META.json").write_text(json.dumps(metadata, indent=2))
        # 三步发布保证崩溃原子性：先把 tmp 改名为不可变版本目录，
        # 再用相对符号链接指向它并 os.replace 到公共路径。
        # 进程在任何时刻退出，公共路径要么指向旧的完整版本，要么指向新的完整版本；
        # 旧版本目录保留在磁盘上，供已经解析过它的读者继续使用。

```

```

version_dir = final_dir.parent / f"_version_{final_dir.name}_{uuid4().hex}"
os.replace(tmp_dir, version_dir)
tmp_dir.symlink_to(version_dir.name, target_is_directory=True)
os.replace(tmp_dir, final_dir)

make_tmp_dir()
_barrier()
write_shards(tmp_dir)
_barrier()
publish_dir()
_barrier()

def _rank() -> int:
    return dist.get_rank() if dist.is_initialized() else 0

def _barrier() -> None:
    if dist.is_initialized():
        # 三个 phase 前后各插一道 gloo barrier, 保证写入与发布在全 rank 视角下有序
        dist.barrier(group=get_gloo_group())

```

tests/e2e/lora/test_slot_checkpoint_resume.py

评审第 4 条意见要求补正向 round-trip 与跨槽污染断言，该文件是最终交付物：在真实 Megatron DDP + MultiLoRALinear + SlotOptimizer 上做 3 步预热后跨槽恢复，并含两个负向对照证明测试确实在测正确性。

```

register_cuda_ci(est_time=120, suite="stage-b-2-gpu-h200", labels=["lora"], hardware=["hopper"])

ADAM = dict(learning_rate=3e-4, beta1=0.9, beta2=0.95, eps=1e-8, weight_decay=0.01, grad_
clip_norm=1.0)

def apply_step(model, optimizer, step):
    # Identical post-all-reduce gradients isolate checkpoint/Adam correctness from forward
    kernels.
    # 直接写入「全 rank 相同」的梯度，把检查点 /Adam 的正确性与 forward kernel 的数值噪声隔离
    for index, param in enumerate(adapter_slot_parameters(model, optimizer.slot)):
        gradient = torch.arange(param.numel(), device=param.device, dtype=torch.float32).
        reshape(param.shape)
        param.main_grad.copy_(torch.sin(gradient + index + step) * 0.1)
    outcome = step_slot_optimizers({optimizer.slot: optimizer}, {optimizer.slot: ADAM})
    assert outcome[optimizer.slot]["grad_norm"] > 0

def assert_same_state(actual, expected):
    # rtol=0 / atol=0: 逐位严格比对，任何舍入或漏存都会被抓住
    actual_values, actual_spec = tree_flatten(actual)
    expected_values, expected_spec = tree_flatten(expected)

```

```

assert actual_spec == expected_spec
for actual_value, expected_value in zip(actual_values, expected_values, strict=True):
    if isinstance(expected_value, torch.Tensor):
        torch.testing.assert_close(actual_value, expected_value, rtol=0, atol=0)
    else:
        assert actual_value == expected_value

# --- run_worker 中的关键断言 (DP=2, 真实 DDP + MultiLoRA + SlotOptimizer) ---
save_slot(model, source, str(checkpoint_dir))
resumed = SlotOptimizer(args, model, slot=1)
load_slot(model, resumed, str(checkpoint_dir), load_optimizer=True)
# 跨槽恢复必须逐位一致: 权重 + FP32 master + Adam moment + step 计数
assert_same_state(snapshot(model, resumed), saved)
# 关键回归: 装载 slot 1 不得改动仍在训练的 slot 0 的 FP32 master (评审意见 1)
assert_same_state(snapshot(model, source), saved)

apply_step(model, source, 4)
uninterrupted = snapshot(model, source)
apply_step(model, resumed, 4)
# 恢复后的下一次 Adam 更新必须与「从未中断」的轨迹完全重合
assert_same_state(snapshot(model, resumed), uninterrupted)
assert all(group["step"] == 4 for child in resumed._inner.chained_optimizers for group in
child.param_groups)

# 负向对照 1: 把 FP32 master 舍入回 BF16 必须被检测到
load_slot(model, resumed, str(checkpoint_dir), load_optimizer=True)
resumed.reload_masters()
with pytest.raises(AssertionError):
    assert_same_state(snapshot(model, resumed), saved)

# 负向对照 2: 只恢复权重、不恢复优化器状态, 下一步更新必须发散
resumed = SlotOptimizer(args, model, slot=1)
load_slot(model, resumed, str(checkpoint_dir), load_optimizer=False)
apply_step(model, resumed, 4)
with pytest.raises(AssertionError):
    assert_same_state(snapshot(model, resumed), uninterrupted)

```

评论区精华

评审由 [nanjiangwill](#) 主导, 共 5 条意见, 全部打在 早期版本的 [miles/backends/megatron_utils/lora/checkpoint.py](#) (当时是 +193 行的自定义分片实现, 含 [run_checkpoint_phase](#)、[megatron_shard_name](#)) 与 [checkpoint_io.py](#) 上; 最终树改用 [dist_checkpointing](#) + 版本目录符号链接, 可视为对这批意见的整体回应。

1. 跨槽 FP32 master 污染 (正确性, 最重要)

"[optimizer](#) is the outer LayerWise/Chained optimizer, so this reload walks every slot's child and copies BF16 model weights into FP32 masters. Loading slot S

therefore rounds the masters of every live slot; only S is restored from `masters` below."

建议「Add a slot-local reload helper and use it in both paths, with a two-slot test that keeps slot B's FP32 master different from its BF16 model value」。最终由 `SlotOptimizer.load_sharded_state / reload_masters` (槽位本地) 解决, e2e 里也用 `assert_same_state(snapshot(model, source), saved)` 明确断言装载 slot 1 后 slot 0 状态未变。

1. 读阶段校验不足, 坏分片会半途污染活跃槽位

"It does not validate master/moment tensor shapes or adapter weight keys and shapes. `load_adapter()` copies matching tensors first and reports missing or unused keys afterward, so a bad shard can mutate part of the live slot before raising."

最终改为「先 `dist_checkpointing.load` 整体校验、再 `load_sharded_state` 赋值」的两段式流程。

1. 发布窗口不原子 + 清理失败误报

"the published path is still absent between `final_dir -> old_dir` and `tmp_dir -> final_dir`. A process exit in that window leaves the last valid checkpoint only under `_old_<name>`, so readers lose the public path... Use immutable version directories plus an atomically replaced pointer/exchange for crash atomicity, and make old-version cleanup best-effort."

最终实现正是这条建议的形状: `_version_<name>_<uuid>` 不可变目录 + 相对符号链接 `os.replace`, 且删除动作只剩 `make_tmp_dir` 里对 `_tmp_*` 的清理。

1. 测试只覆盖读阶段失败, 缺正向 round-trip

"These tests cover read-phase failures only. They never apply a valid checkpoint, so they do not verify FP32 masters, Adam moments and step counters, weights-only loading, or equivalence of the next Adam update."

对应交付物是 `tests/e2e/lora/test_slot_checkpoint_resume.py` (含跨槽比对与两个负向对照)。注意该意见指向的 `tests/fast/backends/megatron_utils/test_lora_slot_checkpoint_load.py` 并未出现在最终改动文件列表里, 读阶段失败用例随旧实现一起退场。

1. `yushengsu-thu` 给出交叉建议 ("Suggestion: <https://github.com/radixark/miles/pull/3143>") 并最终 APPROVED; 另有一条意见只是说明「This code moved; the current comment is here」的行号迁移提示。claude[bot] 的评论是仓库手工评审开关的模板说明, 无技术信息量。

- 装载检查点会重舍入所有活跃槽位的 FP32 master (correctness): 最终由 `SlotOptimizer.load_sharded_state` (槽位本地赋值) 与权重路径的 `reload_masters()` 解决, e2e 用 `assert_same_state(snapshot(model, source), saved)` 断言跨槽装载后源槽位状态不变。
- 读阶段校验不足, 坏分片会先污染活跃槽位再报错 (correctness): 改用 Megatron `dist_checkpointing.load` 先整体校验并返回张量、成功后才调用 `load_sharded_state` 赋值

；但原意见所指向的读阶段失败用例文件未出现在最终改动列表中，失败路径的快速单测覆盖实际被削弱。

- 发布窗口崩溃会让公共检查点路径消失 (design): 最终实现完整采纳该建议: 改为 `_version_<name>_<uuid>` 不可变目录 + 相对符号链接 `os.replace`, 并把 `shutil.rmtree` 从发布阶段移到 `make_tmp_dir` 的残留清理, 发布阶段不再做失败清理。
- 测试只覆盖读阶段失败, 缺正向 round-trip (testing): 交付 `tests/e2e/lora/test_slot_checkpoint_resume.py`: 2 卡 H200 上做跨槽恢复比对与「下一步 Adam 等价」断言, 并用 master 舍入、纯权重重启两个负向对照自证; 代价是资源需求从 CPU 单测升到 2 卡 GPU e2e。
- 评论位置随重写迁移 (question): 无技术争议, 仅反映该 PR 在评审期间经历了较大篇幅的实现重写。
- yushengsu-thu 指向 PR #3143 的交叉建议 (other): 已批准合入; 建议内容是否落地无法从现有材料判断, 需要回看 #3143 上下文确认。

风险与影响

- 风险:
 1. IO 失败即终止服务, 无隔离 / 重试 (`checkpoint_io.py` 头部 TODO; PR body 明说 "execution failures, including filesystem errors, propagate to TrainerCell and terminate the current single-cell Tinker service")。一次 GPFS 抖动会打挂整个 cell, 多租户场景下波及同一 cell 上的所有租户, 这是被显式记录但仍未偿还的技术债。
 2. rank 局部失败可能先形成 barrier 阻塞。`make_tmp_dir()` 中的 `FileExistsError` (`overwrite=False`) 与 `NotImplementedError` (legacy 目录) 只在 `_rank() == 0` 分支抛出, 其余 rank 会继续推进到 `_barrier()` 等待; 评审 `diff_hunk` 里早期版本的 `run_checkpoint_phase` 全局 all-gather 错误广播在最终实现中已不存在, 因此非 0 rank 需要依赖 gloo barrier 超时才能感知失败。
 3. 版本目录无回收策略。`_version_<name>_<uuid>` 只增不删 (清理逻辑仅针对 `_tmp_*`), 高频 SAVE 场景下磁盘单调增长, 需配合外部 GC 或后续补保留策略。
 4. 契约口径不一致。`lora/checkpoint.py` 的模块 docstring 声称检查点 "reloads under any tp/pp/ep/world-size layout", 而 PR body 明确 optimizer state 恢复 "requires the same parallel topology"。两者对「权重可重分片」与「优化器状态受拓扑约束」的边界没有写在一起, 容易被误读为检查点可自由跨拓扑迁移。
 5. key 归一化依赖字符串替换。`_canonicalize_slot_keys` 用 `sharded.key.replace(".adapters.{slot}.", ".adapter.")`, 对 key 命名约定有强假设; 若未来出现其他含该子串的 key 或层名, 存在误替换风险 (当前仅有 `tests/fast/.../test_slot_checkpoint_keys.py` 13 行的单点守护)。
 6. 拓扑与失败路径覆盖不足。PR body 自述验证范围为 `2xH200`、`DP=2`、`TP/PP/EP=1`, 未覆盖 MoE/TP/PP 布局、拓扑变化、server 重启、失败恢复与 forward/backward 数值; `tests/e2e/lora/test_slot_checkpoint_resume.py` 直接依赖 `optimizer._inner`、`chained_optimizers`、`fp32_from_float16_groups` 等 Megatron 内部结构, 上游重构易致测试失效。

7. 向后兼容限制。legacy 目录型检查点可读不可覆盖 (NotImplementedError)，共享检查点根目录时历史命名与新逻辑冲突会直接报错，需要运维侧明确「换名字保存」的迁移动作。

• 影响：

- 对用户 / 租户：Tinker 多租户用户可以按槽位保存与恢复适配器及其完整 Adam 状态，续训保真度从「仅权重」提升到「权重 + FP32 master + moment + step 计数」，跨槽恢复 (slot 0 存、slot 1 载) 被 e2e 验证为逐位一致；代价是保存 / 装载失败会导致当前服务终止，用户需要预期这一行为。
- 对系统：新增了可复用的检查点发布原语 `write_checkpoint_dir`，「不可变版本目录 + 相对符号链接」语义会被 4/7 的 snapshot publisher 与 6/7 的网关运行时复用；`miles/ray/train/group.py` 的槽位命令 API 面扩大并收敛返回类型，是后续协议层的契约基础。
- 对性能与存储：走 Megatron `torch_dist` 写路径并插入三处 barrier，SAVE 期间所有 rank 同步等待；每个版本目录保留完整分片副本，存储放大与写延迟都需要在共享文件系统 (验证用 GPFS) 上评估。
- 对团队 / CI：新增 2 卡 Hopper e2e (stage-b-2-gpu-h200) 会占用 GPU CI 资源；检查点契约 (拓扑约束、legacy 兼容、无 GC、IO 失败语义) 需要在文档中补齐，否则后续章节容易踩同一批坑。
- 风险标记：检查点 IO 失败即终止服务，版本目录无回收策略，rank 局部失败可能阻塞 barrier，优化器状态拓扑约束未验证，key 归一化依赖字符串替换，legacy 目录检查点不可覆盖

关联脉络

- PR #2841 multi-lora v2 前置章节 (本 PR 的 stack base)：PR body 明确 "Stacked on #2841"，本 PR 的槽位机制、SlotOptimizer 与槽位命令 API 都建立在该前置改动之上。
- PR #2843 [multi-lora v2 4/7] Publish immutable adapter snapshots to disk: 同一功能线的下一环：快照发布复用「不可变版本目录 + 原子顶替」的语义，本 PR 的 `write_checkpoint_dir` 是其直接基础。
- PR #2844 [multi-lora v2 5/7] Tinker gateway protocol face: 同系列 5/7，协议层与每模型调度器需要调用本 PR 暴露的 `save_slot/load_slot(..., ckpt_path, load_optimizer)` 契约。
- PR #2845 [multi-lora v2 6/7] Tinker gateway runtime and entry point: 同系列 6/7，运行时把检查点保存 / 装载纳入网关命令流，依赖本 PR 的失败传播语义 (执行失败终止 trainer cell)。
- PR #2849 [multi-lora v2 7/7] Tinker gateway example and tests: PR body 提到检查点相关回归「included in the existing #2849 checkpoint test file」，且该 PR 验证了保存 / 装载在共享检查点根下的多租户行为。
- PR #3198 multi-lora v2 汇总树 (本系列最终验证对象)：PR body 多次引用 #3198 的最终树 SHA 与测试结论，本 PR 的实现被视为该汇总树的一部分。
- PR #3143 评审建议指向的 PR: yushengsu-thu 在评审中留下 "Suggestion: <https://github.com/radixark/miles/pull/3143>"，属于讨论中被提及的关联改动。

- PR #3245 fix(megatron): respect --no-load-optim when resuming a LoRA adapter: 同一「LoRA 检查点与优化器状态」主题线的历史修复（同族还有 #3083），说明优化器状态恢复的正确性在该仓库是反复出现的问题域，可与本 PR 的 load_optimizer 语义对照阅读。