PR #2020 完整报告

THUDM/slime

Accelerate raw HF save with node writers

合并时间：2026-06-04 23:36

原文链接：http://prhub.com.cn/THUDM/slime/pull/2020

执行摘要

- 一句话：多节点并行写入 HF 权重 shard 文件
- 推荐动作：本 PR 的设计模式值得精读 —— 通过分片和最终合并的思路实现分布式写入，可推广至其他需要并行写入文件的场景。作者在每个关键决策点（文件名唯一、state 合并排序）都做了防御性检查，代码质量较高。需要关注的是如何与外部引擎地址配置协同工作，可参考 #2016。

功能与动机

当前 save_hf_model_direct 使用单节点串行写入所有 shard，随着模型规模增长（千亿参数甚至更大）， checkpoint 保存时间成为训练瓶颈。通过分布式节点并行写入，可线性缩小保存时间（受限于节点间写入均衡和最终合并开销）。

实现拆解

1. 节点布局获取：在 save_hf_model_direct 开头调用新增的 _get_node_save_layout(args) 获取当前节点在写入配置中的信息，包括总写入节点数 num_save_nodes、当前节点序号 save_node_rank、是否为 writer 节点 is_writer_rank 等。
2. 并行写入循环：遍历 HF 权重 chunk 时，通过 chunk_idx % num_save_nodes == save_node_rank 判断当前 chunk 是否由本节点写入；若匹配则将 (chunk_idx, named_tensors) 缓存为 pending_write，否则释放 tensor。每累积 num_save_nodes 个 chunk 或遍历结束后，调用 _write_pending_chunk 将缓存的 chunk 写入磁盘。
3. ShardWriter 改造：_SafetensorShardWriter.write 新增 shard_idx 参数，使用 _next_filename(shard_idx) 生成唯一文件名，避免节点间文件名冲突。新增 state() 方法导出 total_size、weight_map、shard_files 用于后续合并。
4. 最终合并：调用 _finalize_distributed_shards（内部调用 _finalize_shard_files）收集所有节点 writer 的 state，按照 _shard_filename_sort_key 排序后重命名文件为 model-NNNNN-of-MMMMM.safetensors 格式，并写出 model.safetensors.index.json。
5. 测试配套：新增两个单元测试：test_finalize_shard_files_merges_node_writer_states 验证两个 writer state 合并后的完整流程；test_pending_chunk_write_flushes_incomplete_node_group 验证当 chunk 总数不是节点数的整数倍时，最后一个批次能正确 flush。

关键文件：

- slime/backends/megatron_utils/hf_checkpoint_saver.py（模块 模型保存；类别 source；类型 core-logic；符号 write, state, _next_filename, _write_pending_chunk）：核心实现文件，重写了保存逻辑以支持多节点并行写入，新增多个辅助函数。
- tests/utils/test_hf_checkpoint_saver.py（模块 单元测试；类别 test；类型 test-coverage；符号 test_finalize_shard_files_merges_node_writer_states, test_pending_chunk_write_flushes_incomplete_node_group）：补充两个关键单元测试，验证 node writer 状态的合并以及 incomplete chunk 的 flush 逻辑。

关键符号：_get_node_save_layout, save_hf_model_direct, _SafetensorShardWriter.write, _SafetensorShardWriter.state, _write_pending_chunk, _finalize_distributed_shards, _finalize_shard_files, _shard_filename_sort_key, _next_filename

关键源码片段

slime/backends/megatron_utils/hf_checkpoint_saver.py

核心实现文件，重写了保存逻辑以支持多节点并行写入，新增多个辅助函数。

```
# save_hf_model_direct 核心并行写入片段
num_save_nodes, save_node_rank, is_writer_rank, writer_ranks = _get_node_save_layout(args)
if is_save_rank:
    logger.info(
        "Raw HuggingFace save will write shards from %d node writer rank(s): %s",
        num_save_nodes,
        writer_ranks,
    )

writer = _SafetensorShardWriter(path, enabled=is_writer_rank)
pending_write = None

# 遍历所有 HF 权重块，分配给对应的节点 writer
for chunk_idx, hf_named_tensors in enumerate(
    hf_weight_iterator.get_hf_weight_chunks(megatron_local_weights, progress_desc="Save HF
    checkpoint")
):
    # 如果当前节点 writer 分配到该 chunk，则缓存 pending_write
    if is_writer_rank and chunk_idx % num_save_nodes == save_node_rank:
        pending_write = (chunk_idx, hf_named_tensors)
        hf_named_tensors = None
    else:
        del hf_named_tensors

    # 每累积 num_save_nodes 个 chunk，就刷新写入
    if (chunk_idx + 1) % num_save_nodes == 0:
        pending_write = _write_pending_chunk(writer, pending_write)

# 写入最后一批未完成的 chunk
pending_write = _write_pending_chunk(writer, pending_write)
# 合并所有节点 writer 的 state 并生成最终 index 文件
```

```
    _finalize_distributed_shards(path, writer.state())

    if is_save_rank:
        logger.info("Successfully saved HuggingFace model to %s", path)
```

tests/utils/test_hf_checkpoint_saver.py

补充两个关键单元测试，验证 node writer 状态的合并以及 incomplete chunk 的 flush 逻辑。

```
# 测试多个 writer state 合并为正确 index
def test_finalize_shard_files_merges_node_writer_states(tmp_path: Path):
    writer0 = _SafetensorShardWriter(tmp_path, enabled=True)
    writer1 = _SafetensorShardWriter(tmp_path, enabled=True)

    # 模拟两个节点分别写入 shard 0 和 shard 1
    writer0.write([("layers.0.weight", torch.ones(2, 2))], shard_idx=0)
    writer1.write([("layers.1.weight", torch.zeros(2, 2))], shard_idx=1)

    # 合并两个 writer 的 state
    _finalize_shard_files(tmp_path, [writer0.state(), writer1.state()])

    # 验证 index 正确且临时文件被重命名
    index = json.loads((tmp_path / "model.safetensors.index.json").read_text(encoding="utf-8"))
    assert index["metadata"]["total_size"] == 32
    assert index["weight_map"] == {
        "layers.0.weight": "model-00001-of-00002.safetensors",
        "layers.1.weight": "model-00002-of-00002.safetensors",
    }
    assert not (tmp_path / "model-00001.safetensors").exists()
    assert not (tmp_path / "model-00002.safetensors").exists()
```

评论区精华

无 review 讨论。

- 暂无高价值评论线程

风险与影响

- 风险：
  - 文件冲突风险：引入并行的节点写者后，不同节点可能写入同名文件。代码通过 _next_filename(shard_idx) 确保文件名严格基于 shard_idx，且 write 中检查文件已存在则抛出 ValueError，避免覆盖。
  - State 合并正确性：_finalize_shard_files 依赖所有 writer state 的 shard_files 列表按 chunck 顺序排序，若节点间 state 顺序错乱可能导致 index 错误。实现中使用 _shard_filename_sort_key 依 shard_idx 排序，且 writer 内部 shard_files 追加顺序与 write 调用顺序一致，风险较低。
  - 性能风险：若写入节点数大于 chunk 总数，部分节点将无 chunk 可写，但仍参与最终 barrier 同步，可能导致不必要等待。当前无保护，但实际 chunk 数通常远大于节点数（

每个 chunk 为一个 transformer layer 的权重），影响可忽略。

- 缺少集成测试：当前只有单元测试，没有端到端多节点并行写入的 CI 测试，可能遗漏分布式环境下的竞态或通信问题。
- 影响：对用户而言，大模型训练中的 checkpoint 保存速度将显著提升（实测预期可达节点数倍数加速）。对系统无外部接口或 API 变更，仅内部实现优化。对团队而言，需了解新的 _get_node_save_layout 配置（当前尚需结合 --rollout-external-engine-addrs 等参数，PR #2016 已提供基础）以确保多节点环境正确启动。测试覆盖主要路径，但建议补充集成测试。
- 风险标记：缺少集成测试，节点数超过 chunk 数可能导致闲置

关联脉络

- PR #2016 Fully support --rollout-external-engine-addrs: 该 PR 引入了 _get_node_save_layout 所需的外部引擎地址配置，两 PR 一同构成了多节点保存的完整基础设施。