

PR #38503 完整报告

sgl-project/sglang

Fix mooncake scale joiner groups

合并时间: 2026-09-15 19:07

原文链接: <http://prhub.com.cn/sgl-project/sglang/pull/38503>

执行摘要

- 一句话: 修复 Mooncake 下弹性 EP 扩容 joiner 进程组建组阻塞
- 推荐动作: 建议精读, 重点读 `parallel_state.py` 中 `use_local_synchronization` 与 `max_world_size` 的语义拆分, 以及 `bootstrap.py` 里两个初始化调用参数的不对称——这是理解「WORLD 可扩张 vs 模型并行固定」契约的唯一入口。具体建议: (1) 按 #31804、#39332 的测试风格, 在 `test/registered/unit/` 下补一个覆盖 joiner 建组参数透传的单元测试, 至少断言 `rank_offset > 0` 时 `use_local_synchronization=True`、模型并行组不携带 `max_world_size`; (2) 澄清 AMD ROCm 任务失败是否为环境性 flaky, 避免把「只在 Mooncake + CUDA 验证」的改动当作全后端修复; (3) 若后续要新增进程组, 务必在 `initialize_model_parallel` 内同步传参, 避免重演 DCP / `self_pp` 的漏传问题。

功能与动机

PR body 明确目标是把 Mooncake 的进程组注册修正为「Enable Elastic EP scale joiners」所要求的形式: Keep WORLD groups expandable and append-aware, 同时 Create model-parallel groups as fixed, cohort-local groups with global rank offsets。背景问题是扩容时新加入的 cohort 里的 rank 是相对 `rank_offset` 的局部编号, 服务中的老 rank 不会同步执行同一段建组代码, 若仍走 torch 默认的全局 barrier 建组, joiner 侧会等不到老 rank 而阻塞; 反过来, 如果为了能建组而无差别地把所有组登记为可扩张, 模型并行组又会丢掉固定语义。作者给出的验证是: focused distributed unit tests passed、Mooncake 0.3.13 EP4→EP6 GPU passed、手工 GSM8K scale-up 通过且准确率 62.89%。

实现拆解

1. 入口层区分两类 joiner (`python/sglang/srt/distributed/bootstrap.py::_init_parallel_groups`): 函数先取出 `is_ep_joiner`、`is_ep_scale_joiner`, 并用 `rank_offset = get_parallel().ep_join_rank_offset if is_scale_joiner else 0` 计算全局 rank 偏移, `world_size/rank` 都按该偏移换算。随后对两个初始化调用使用不同的参数语义: `init_distributed_environment` 继续传 `recovered_rank=is_ep_joiner` 与 `max_world_size=get_parallel().max_ep_size` (WORLD 需要保持可扩张); `initialize_model_parallel` 改为 `recovered_rank=is_ep_joiner and not is_scale_joiner` 且 `max_world_size=None if is_scale_joiner else get_parallel().max_ep_size`, 即 scale joiner 的模型并行组不再是「恢复 / 可扩张」语义, 而是本 launch cohort 内的固定组。

2. 建组同步范围收敛 (`python/sglang/srt/distributed/parallel_state.py::GroupCoordinator.__init__`) : 新增 `use_local_synchronization = rank_offset > 0 and not recovered_rank` , 并把它透传给 mooncake、mooncake-cpu、gloo 以及默认后端的全部 `torch.distributed.new_group()` 调用。这样 joiner cohort 只在自身内部完成建组同步, 不再等待未参与建组的老 rank。普通部署 `rank_offset == 0`, 该开关恒为 False, 行为与改动前完全一致, 回归面被压到 joiner 路径内。
3. Mooncake active-rank 语义调整 (同一函数) : 把 `if not recovered_rank and max_world_size is not None` 简化为 `if max_world_size is not None`; 两处 (`pg_active_size` 断言与 `MooncakeBackendOptions` 构造) 同步修改。原因是 joiner 的 `recovered_rank=False` 之外, 更根本的判据应该是「这个组要不要预留扩容位」; 简化后 `max_world_size` 成为组是否 append-aware 的唯一开关。
4. WORLD 组增加扩容透传能力 (`parallel_state.py::init_world_group / init_distributed_environment`) : `init_world_group` 新增 `max_world_size: Optional[int] = None` 形参并转交给 `GroupCoordinator`, `init_distributed_environment` 改为关键字调用并把 `max_world_size` 一并传入。这是本 PR 的关键契约——WORLD 是全流程唯一被允许扩张的组, 其余组不带该参数即为固定组。
5. 补齐偏移与扩容参数在子组上的传播 (`parallel_state.py::initialize_model_parallel`) : `world_size` 的局部推导条件由 `if recovered_rank` 改为 `if recovered_rank or rank_offset > 0` (joiner 也要按全局 rank 空间推导自己的 TP/PP 布局); DCP 组 (`group_name="dcp"`) 补上 `rank_offset` 与 `max_world_size`; `self_pp` 组补上 `recovered_rank`、`rank_offset` 与 `max_world_size`。这些子组此前拿不到偏移, joiner 会按错误的 rank 列表建组, 属于同一 bug 的连带修复。
6. 测试与验证配套: 本 PR 未新增或修改任何测试文件, 验证完全依赖 PR body 描述的手工路径 (Mooncake 0.3.13 EP4→EP6、GSM8K 62.89%) 与既有 CI; PR body 中 AMD ROCm 10 任务状态为失败, 讨论中未给出结论。

关键文件:

- `python/sglang/srt/distributed/parallel_state.py` (模块 分布式通信; 类别 source; 类型 core-logic; 符号 `GroupCoordinator.init`, `init_world_group`, `init_distributed_environment`, `initialize_model_parallel`) : 本 PR 的核心实现文件: `GroupCoordinator.__init__` 新增 `use_local_synchronization` 并把 `max_world_size` 改为可扩张性的唯一开关; `init_world_group/init_distributed_environment` 透传 `max_world_size` 使 WORLD 组成为唯一 append-aware 的组; `initialize_model_parallel` 修正 `world_size` 判定并给 DCP、`self_pp` 子组补齐 `rank_offset/max_world_size`。
- `python/sglang/srt/distributed/bootstrap.py` (模块 启动引导; 类别 source; 类型 core-logic; 符号 `_init_parallel_groups`) : 改动入口: `_init_parallel_groups` 中区分 `is_ep_joiner` 与 `is_ep_scale_joiner`, 把 `initialize_model_parallel` 的 `recovered_rank` 改为 `is_ep_joiner and not is_ep_scale_joiner`、`max_world_size` 改为 `None if is_ep_scale_joiner else max_ep_size`, 从调用点确立了「只有 WORLD 扩张」的契约。

关键符号: `GroupCoordinator.init`, `init_world_group`, `init_distributed_environment`, `initialize_model_parallel`, `_init_parallel_groups`

关键源码片段

python/sglang/srt/distributed/parallel_state.py

本 PR 的核心实现文件: `GroupCoordinator.__init__` 新增 `use_local_synchronization` 并把 `max_world_size` 改为可扩张性的唯一开关; `init_world_group/init_distributed_environment` 透传 `max_world_size` 使 WORLD 组成为唯一 append-aware 的组; `initialize_model_parallel` 修正 `world_size` 判定并给 DCP、self_pp 子组补齐 `rank_offset/max_world_size`。

```
# python/sglang/srt/distributed/parallel_state.py
# GroupCoordinator.__init__: 弹性 EP 扩容 (scale joiner) 下的建组适配
self.local_size = get_int_env_var("LOCAL_SIZE", 0)

# 扩容 cohort 内的 rank 是相对 rank_offset 的局部编号, 服务中的老 rank 不会
# 同步执行同一段建组代码; 建组同步必须限制在本 cohort 内, 否则 new_group 会
# 停在全局 barrier 上等老 rank 而永久阻塞。常规部署 rank_offset == 0,
# use_local_synchronization 恒为 False, 行为与改动前完全一致。
use_local_synchronization = rank_offset > 0 and not recovered_rank

for ranks in group_ranks:
    subgroup_timeout = _MODEL_PARALLEL_GROUP_TIMEOUT
    if "mooncake" in torch_distributed_backend:
        from mooncake.pg import MooncakeBackendOptions

        # max_world_size 表示该组最终会扩张到的规模: WORLD 组由 init_world_group
        # 带上它, 需要按目标规模预留 active ranks 掩码以支持 append; 模型并行组
        # 不带该参数, 保持固定规模。这里去掉了原先的 recovered_rank 门控,
        # 避免 joiner 的 WORLD 组被误当成固定组而丢掉 append 能力。
        pg_active_size = len(ranks)
        if max_world_size is not None:
            assert max_world_size >= len(ranks), (
                f"max_world_size ({max_world_size}) must be >= "
                f"group size ({len(ranks)})"
            )
            pg_active_size = max_world_size

        pg_active_ranks = torch.zeros(
            pg_active_size, dtype=torch.int32, device=self.device
        )
        pg_active_ranks[: len(ranks)] = 1
        pg_active_ranks_cpu = torch.zeros(pg_active_size, dtype=torch.int32)
        pg_active_ranks_cpu[: len(ranks)] = 1

        if max_world_size is not None:
            # 可扩张组: 显式把目标规模交给 Mooncake 后端, 登记为 append-aware
            dev_opts = MooncakeBackendOptions(
                pg_active_ranks, recovered_rank, max_world_size
            )
```

```

    cpu_opts = MooncakeBackendOptions(
        pg_active_ranks_cpu, recovered_rank, max_world_size
    )
else:
    # 固定组：按当前 cohort 规模一次建好，不预留扩容槽位
    dev_opts = MooncakeBackendOptions(pg_active_ranks, recovered_rank)
    cpu_opts = MooncakeBackendOptions(pg_active_ranks_cpu, recovered_rank)

active_ranks = pg_active_ranks[: len(ranks)]
active_ranks_cpu = pg_active_ranks_cpu[: len(ranks)]
device_group = torch.distributed.new_group(
    ranks,
    backend="mooncake",
    pg_options=dev_opts,
    timeout=subgroup_timeout,
    group_desc=f"{group_name}:device",
    use_local_synchronization=use_local_synchronization, # 只在本 cohort 内同步
)
cpu_group = torch.distributed.new_group(
    ranks,
    backend="mooncake-cpu",
    pg_options=cpu_opts,
    timeout=subgroup_timeout,
    group_desc=f"{group_name}:cpu",
    use_local_synchronization=use_local_synchronization,
)

```

python/sglang/srt/distributed/bootstrap.py

改动入口：_init_parallel_groups 中区分 is_ep_joiner 与 is_ep_scale_joiner，把 initialize_model_parallel 的 recovered_rank 改为 is_ep_joiner and not is_scale_joiner、max_world_size 改为 None if is_scale_joiner else max_ep_size，从调用点确立了「只有 WORLD 扩张」的契约。

```

# python/sglang/srt/distributed/bootstrap.py
# _init_parallel_groups: 从调用点区分「需要扩容的 WORLD 组」和「固定的模型并行组」
init_distributed_environment(
    backend=backend,
    world_size=world_size, # rank_offset + tp_size * pp_size (joiner 含偏移)
    rank=rank,
    local_rank=gpu_id,
    distributed_init_method=dist_init_method,
    timeout=get_parallel().dist_timeout,
    moe_a2a_backend=get_exec().moe.moe_a2a_backend,
    recovered_rank=is_ep_joiner,
    max_world_size=get_parallel().max_ep_size, # WORLD 始终按目标规模预留，保持可 append
)
initialize_model_parallel(
    tensor_model_parallel_size=tp_size,
    attention_data_parallel_size=attn_dp_size,

```

```

pipeline_model_parallel_size=pp_size,
expert_model_parallel_size=moe_ep_size,
attention_context_model_parallel_size=attn_cp_size,
moe_data_model_parallel_size=moe_dp_size,
decode_context_parallel_size=dcg_size,
duplicate_tp_group=get_disagg().enable_pdmux,
enable_symm_mem=get_exec().comm.enable_symm_mem,
# 只有 WORLD 会在 scale-up 中扩张；joiner 的模型并行组是它自己那个
# launch cohort 内部的固定组，不参与扩容，也不应按 max_ep_size 预留槽位。
recovered_rank=is_ep_joiner and not is_scale_joiner,
rank_offset=rank_offset,
max_world_size=None if is_scale_joiner else get_parallel().max_ep_size,
)

# python/sglang/srt/distributed/parallel_state.py
# 与之配套：init_world_group / init_distributed_environment 新增并透传 max_world_size
def init_world_group(
    ranks: List[int],
    local_rank: int,
    backend: str,
    recovered_rank: bool = False,
    max_world_size: Optional[int] = None, # 新增：WORLD 组的扩容目标规模
) -> GroupCoordinator:
    return GroupCoordinator(
        group_ranks=[ranks],
        local_rank=local_rank,
        torch_distributed_backend=backend,
        ...,
        group_name="world",
        recovered_rank=recovered_rank,
        max_world_size=max_world_size, # 透传给 MooncakeBackendOptions，使 WORLD 可追加
    )

def init_distributed_environment(..., max_world_size=None):
    if _WORLD is None:
        ranks = list(range(torch.distributed.get_world_size()))
        _WORLD = init_world_group(
            ranks=ranks,
            local_rank=local_rank,
            backend=backend,
            recovered_rank=recovered_rank,
            max_world_size=max_world_size, # WORLD 是唯一被允许扩张的组
        )

```

评论区精华

本 PR 的 [review_comments](#) 为空，实质性讨论集中在 review 结论和 PR 时间线上的 CI 反复 rerun:

- UNIDY2002 给出明确判断：「The fix is correct and should be safe. Let's wait for the CI to go green.」——即技术上认可，但把最终放行条件绑在 CI 绿灯上；nvpohanh 与 ch-wan 随后直接 APPROVED，未留下文字意见。
- CI 侧，UNIDY2002 与 nvpohanh 合计发起 8 次 /rerun-failed-ci 与 /tag-and-rerun-ci，说明改动所在测试面本身不稳定；UNIDY2002 指出「The same registered/hicache/test_hicache_storage.py also fails in other PRs.」，把该失败归因为跨 PR flaky 而非本改动引入。
- 真正值得记录的讨论缺口是「扩容语义契约」没有落到代码评审问答里：作者在 `initialize_model_parallel` 处加了注释“Only WORLD is extended during scale-up. The joiner's model-parallel groups are fixed groups local to its launch cohort.”，但评审中没有人追问该契约如何被自动化测试保护，最终也没有测试补充。
- 修复正确性与放行条件 (correctness): 三方批准，未提出具体技术异议；实质技术判断只有一句「correct and should be safe」，评审深度有限，结论主要依赖作者提供的手工验证。
- CI 反复 rerun 与 flaky 归因 (testing): hicache storage 测试被判定为与本改动无关的 flaky；但 PR body 中 AMD ROCm 10 (Run #34810986397) 仍为失败状态，讨论中未见原因澄清或修复结论。
- 扩容语义契约：WORLD 可扩张 vs 模型并行组固定 (design): 契约被写进代码注释并落成参数约定，但没有任何自动化测试守护该契约，后续新增子组容易漏传 `rank_offset/max_world_size` (DCP、self_pp 本次正是被连带修复的漏传案例)。

风险与影响

- 风险：
 1. 缺少测试覆盖（主要风险）：`rank_offset`、`use_local_synchronization`、`max_world_size` 三者共同决定建组行为，却没有任何测试文件新增。对比同仓库 #31804（补 `test/registered/unit/eplb/test_waterfill_eplb.py` 夹具）、#39332（补 `test/registered/unit/disaggregation/...`）的做法，本 PR 的 joiner 建组路径只有手工验证记录，后续任何新增子组（如再加一个 `group_name`）都可能漏传 `rank_offset/max_world_size` 而静默退化。
 2. 同步语义依赖 `rank_offset` 的正确性：`use_local_synchronization = rank_offset > 0` and `not recovered_rank` 是纯布尔判断，一旦非 joiner 路径被误赋 `rank_offset > 0`，建组会从「全局 barrier」静默切换为「本 cohort barrier」，故障形态可能从显式 hang 变成更隐蔽的通信组不一致。
 3. Mooncake 预留开销：`pg_active_size = max_world_size` 会按 `max_ep_size` 分配 `pg_active_ranks` (device) 与 `pg_active_ranks_cpu` (CPU) 两份 int32 掩码，并交给 MooncakeBackendOptions。当 `max_ep_size` 远大于当前 world size 时存在少量显存 / CPU 元数据开销，量级小但属于长期驻留。
 4. 后端验证面不完整：`use_local_synchronization` 同时作用于 gloo 与默认后端路径，但作者只报告了 Mooncake 的 EP4→EP6 验证；PR body 里 AMD ROCm 10 任务标记为失败，讨论中未见原因澄清，属于未闭环的兼容性风险。
 5. 附带修复缺乏独立验证：DCP 与 self_pp 组的 `rank_offset/max_world_size/recovered_rank` 补参方向正确，但属于同 PR 内顺带修的路径，没有单独用例证明 joiner 场景下

这些子组的 rank 列表确实落在全局 rank 空间。

- 影响：

- 用户侧：使用弹性 EP scale-up (Mooncake 后端，典型如 EP4→EP6) 的部署直接受益，扩容时新 cohort 不再因建组等待老 rank 而阻塞；已按 PR body 的 GSM8K 62.89% 说明扩容后服务精度可维持。
- 系统侧：变更集中在 rank_offset > 0 的 joiner 分支，rank_offset == 0 的常规部署行为不变，理论上回归面很小；但 parallel_state.py 是所有并行初始化 (TP/PP/DP/EP/DCP) 的公共路径，任何误改都会影响全部模型，因此仍属高敏感文件。
- 契约侧：本 PR 明确了「WORLD 可扩张、模型并行组固定」的长期契约，后续弹性扩容相关 PR 需要沿用 max_world_size=None 表示固定组、非 None 表示可扩张组的约定。
- 团队 / CI 侧：PR 时间线上的 8 次 rerun 与 flaky 的 registered/hicache/test_hicache_storage.py 再次暴露 CI 稳定性成本，与近期 #39368、#39545 的 CI 减负 / 等待显存释放属同一治理方向。
- 风险标记：核心路径变更，缺少测试覆盖，AMD ROCm 验证缺失，同步语义依赖 rank_offset 正确性，手工验证为主

关联脉络

- PR #39487 [Fix][DCP] Localize widened KV ids in MLA retraction CPU backup/restore: 同属 DCP 路径的问题域：本 PR 也修改了 DCP 组 (group_name="dcp") 的建组参数，补齐 rank_offset 与 max_world_size，两者的改动区域在「DCP 与分布式组初始化」上交叉。
- PR #39423 [NPU][Bugfix] Disable pinned memory to fix DeepSeek-V2 DP-attention hang: 同属「分布式 / 进程组级别挂死」类 bugfix，思路一致：通过限制跨 rank 同步或资源行为来消除挂起，可与本 PR 的 use_local_synchronization 对照阅读。
- PR #31804 [EPLB] Drop defensive getattr for ep_dispatch_algorithm: 同为 MoE/EP 方向的参数契约收紧类改动，且其「补齐测试夹具 (test/registered/unit/eplb/test_waterfill_eplb.py)」的做法正是本 PR 缺失的测试配套，可作为补测的参考模板。
- PR #39368 [CI][PD] Skip the flaky decode HiCache file-backend disaggregation test: 与本 PR 讨论中的 flaky 测试 (registered/hicache/test_hicache_storage.py) 及多次 /rerun-failed-ci 属同一 CI 稳定性治理脉络，都反映 HiCache / PD 相关测试的不稳定成本。