PR #30164 完整报告

sgl-project/sglang

[1/N] elastic-ep: Add runtime EP scale-up

合并时间：2026-07-17 06:53

原文链接：http://prhub.com.cn/sgl-project/sglang/pull/30164

执行摘要

- 一句话：运行中 MoE EP 规模动态扩展，不重启添加 GPU ranks
- 推荐动作：值得精读，尤其关注 ElasticEPStateManager 状态机、maybe_join_ep_ranks 中的生命周期管理、以及数据平面（NIXL、EPLB、DP attention）的协同更新。但需注意当前存在的遗留限制（空集群阻塞、超时非集体等），建议跟踪后续 PR 的修复。设计上复用现有弹性 EP 恢复机制是一个好的演进方向。

功能与动机

SGLang 目前固定 EP 拓扑，添加 GPU 需要停止服务器、重建分布式拓扑、重载模型。PR 复用已有的弹性 EP rank 恢复机制实现运行时扩展，使部署能根据负载动态调整规模而不中断服务。详见 RFC #22788。

实现拆解

1. 状态管理与 API 层：在 ElasticEPState 中新增 effective_ep_size、pending_ep_size、scale_phase 等字段，ElasticEPStateManager 提供 begin_scale、request_scale、try_admit_scale_ranks 等方法。新增 HTTP 入口 POST /scale_elastic_ep（位于 entrypoints/elastic_ep.py）将 target 写入 scheduler 的 IPC 通道；GET /is_scaling_elastic_ep 暴露当前状态。
2. Rank 接纳与生命周期：启动时使用 --max-ep-size 预留缓冲区 size，active_ranks 中尾部 slots 置零。Joiner 启动后通过 register_scale_cohort 将其目标（ep_join_rank_offset + tp_size）写入全局 TCPStore。Primary 在 maybe_join_ep_ranks 中（每个 forward 调用）检测是否所有 joiner 就绪，若就绪则调用 try_admit_scale_ranks 提交，再通过 _finalize_scale_up 激活扩展后的运行时状态。
3. 数据平面更新：提交时依次执行：on_scale 调度 NIXL 新连接（nixl.py）；_expand_eplb_metadata_for_scale 扩展 expert 位置元数据（expert_location.py）；update_dp_attention_post_scale 切换 DP 收集到 WORLD group（dp_attention.py）；add_elastic_workers 激活 DataParallelController 中的新 scheduler 进程（data_parallel_controller.py）。
4. 测试与配置：新增手动端到端测试 test_elastic_scale.py（模拟多 GPU 扩展场景）；server_args.py 新增 --elastic-ep-join-mode、--ep-join-rank-offset 等参数；修复原有 elastic_ep_rejoin 相关代码。

关键文件：

- python/sglang/srt/model_executor/model_runner.py（模块 模型执行器；类别 source；类型 data-contract；符号 _initialize_elastic_ep_joiner, _expand_eplb_metadata_for_scale, _elastic_global_rank, _report_elastic_scale_failure）：核心协调者，负责 joiner 初始化和 scale-up 生命周期。新增 _initialize_elastic_ep_joiner、maybe_join_ep_ranks、_finalize_scale_up 等方法，是扩展提交的入口。
- python/sglang/srt/elastic_ep/elastic_ep.py（模块 弹性 EP 管理；类别 source；类型 dependency-wiring；符号 register_scale_cohort, get_scale_cohort_target, _init_joiner_state, request_scale）：弹性 EP 状态管理器核心，定义了 scale 状态机、cohort 注册 / 查询、以及 lifecycle 辅助函数。新增 register_scale_cohort、begin_scale、request_scale 等。
- python/sglang/srt/managers/data_parallel_controller.py（模块 数据并行控制器；类别 source；类型 entrypoint；符号 add_elastic_workers, _refresh_active_workers, _joiner_local_tp_span, _joiner_slot_offset）：数据并行控制器，管理 worker 激活和负载分发。新增 add_elastic_workers、_refresh_active_workers，支持运行时增加 scheduler 进程。

关键符号：register_scale_cohort, get_scale_cohort_target, ElasticEPStateManager.begin_scale, ElasticEPStateManager.request_scale, ElasticEPStateManager.try_admit_scale_ranks, ElasticEPStateManager.commit_scale, ElasticEPStateManager._init_joiner_state, ModelRunner._initialize_elastic_ep_joiner, ModelRunner.maybe_join_ep_ranks, ModelRunner._finalize_scale_up, ModelRunner._expand_eplb_metadata_for_scale, DataParallelController.add_elastic_workers, DataParallelController._refresh_active_workers, Scheduler.handle_scale_elastic_ep, Scheduler._maybe_namespace_elastic_radix_cache, NixlTokenDispatcher.on_scale, NixlTokenDispatcher._connect_ranks, NixlTokenDispatcher._update_connections, ExpertLocationMetadata.init_trivial, append_trivial_expert_slots, set_global_expert_location_metadata, update_dp_attention_post_scale, world_dp_gather_enabled

关键源码片段

python/sglang/srt/model_executor/model_runner.py

核心协调者，负责 joiner 初始化和 scale-up 生命周期。新增 _initialize_elastic_ep_joiner、maybe_join_ep_ranks、_finalize_scale_up 等方法，是扩展提交的入口。

以下为 maybe_join_ep_ranks 和 _finalize_scale_up 的核心逻辑：

```
def maybe_join_ep_ranks(self) -> None:
    """每次 forward 调用，检测是否有待提交的 scale-up。"""
    inst = ElasticEPStateManager.instance()
    if inst is None or not inst.is_scaling():
        return

    # 如果 pending_size 非 None 且尚未有 joiner 注册时间超时
```

```
    if inst.pending_ep_size is not None:
        elapsed = time.monotonic() - inst.pending_since
        timeout = getattr(self.server_args, "elastic_ep_scale_timeout", 600)
        if elapsed > timeout:
            # 超时判定应为集体操作（TODO: 后续 PR 改为 allreduce）
            self._report_elastic_scale_failure("timeout")
            return

        # 通过 TCPStore 检查 cohort 是否就绪
        cohort_target = get_scale_cohort_target(inst.ep_join_rank_offset)
        if cohort_target is None or cohort_target < inst.pending_ep_size:
            return # joiner 尚未注册

    # 尝试通过 Mooncake 接纳新 ranks
    if not try_admit_scale_ranks(inst.pending_ep_size):
        return

    # 提交 scale-up：更新 NIXL、EPLB、DP attention、DataParallelController
    self._finalize_scale_up()


def _finalize_scale_up(self) -> None:
    """执行状态转换，使新 EP 大小生效。"""
    inst = ElasticEPStateManager.instance()
    old_size = inst.effective_ep_size
    new_size = inst.pending_ep_size

    # 1. 扩展 expert 位置元数据
    self._expand_eplb_metadata_for_scale(old_size, new_size)

    # 2. 通知 NIXL 连接新 ranks
    if self.server_args.moe_a2a_backend == "nixl":
        NixlTokenDispatcher.on_scale(old_size, new_size)

    # 3. 更新 DP attention 全局组
    update_dp_attention_post_scale(new_size, self.dcp_rank)

    # 4. 激活 DataParallelController 中的新 worker
    dp_controller = get_dp_controller()
    if dp_controller is not None:
        slot_offset = self.server_args.ep_join_rank_offset
        slot_count = new_size - old_size
        dp_controller.add_elastic_workers(slot_offset, slot_count)

    # 5. 提交状态
    inst.commit_scale(new_size)

    # 6. 重置 tree cache 命名空间
    self._invalidate_radix_cache_namespace()
```

注意：上述代码为重新整理后的逻辑片段，注释遵守盘古排版规则。

python/sglang/srt/elastic_ep/elastic_ep.py

弹性 EP 状态管理器核心，定义了 scale 状态机、cohort 注册 / 查询、以及 lifecycle 辅助函数。新增 register_scale_cohort、begin_scale、request_scale 等。

以下为 ElasticEPState 新增字段和 _init_joiner_state 逻辑：

```
@dataclass
class ElasticEPState:
    active_ranks: Optional[torch.Tensor]
    last_active_ranks: Optional[torch.Tensor]
    active_ranks_cpu: Optional[torch.Tensor]
    # ---- scale-up 新增字段 ----
    effective_ep_size: int = 0 # 当前使用的 EP 大小
    pending_ep_size: Optional[int] = None # 待提交的目标 EP 大小
    scale_phase: str = "idle" # 状态机阶段
    last_error: Optional[str] = None
    pending_since: Optional[float] = None
    original_ep_size: int = 0 # launch 时 EP 大小
    has_scaled: bool = False
    ep_join_rank_offset: int = 0 # joiner 在全局 rank 空间中的偏移

    def reset(self):
        if self.active_ranks is not None:
            # 预留 slots 保持 inactive 直到对应 ranks 加入
            self.active_ranks.zero_()
            self.active_ranks[: self.effective_ep_size] = 1
            self.snapshot_active_to_last()
            self.sync_active_to_cpu()


class ElasticEPStateManager:

    @classmethod
    def _init_joiner_state(cls, inst: ElasticEPState, server_args: ServerArgs) -> None:
        """初始化 joiner rank 的状态。"""
        global_rank = torch.distributed.get_rank()
        inst.active_ranks.zero_()
        inst.active_ranks[global_rank] = 1
        inst.snapshot_active_to_last()
        inst.sync_active_to_cpu()

        if server_args.ep_join_mode == "scale":
            # scale 模式：joiner 的有效 EP 大小基于 offset + 本地 TP 大小
            inst.effective_ep_size = (
                server_args.ep_join_rank_offset + server_args.tp_size
            )
            inst.original_ep_size = (
```

```
                server_args.elastic_ep_initial_size or server_args.ep_join_rank_offset
            )
            inst.has_scaled = True
        else:
            # 普通 recovery 模式
            world_size = torch.distributed.get_world_size()
            inst.effective_ep_size = world_size
            inst.original_ep_size = world_size
```

python/sglang/srt/managers/data_parallel_controller.py

数据并行控制器，管理 worker 激活和负载分发。新增 add_elastic_workers、_refresh_active_workers，支持运行时增加 scheduler 进程。

以下为 add_elastic_workers 实现：

```
defadd_elastic_workers(self, slot_offset: int, slot_count: int): """激活预绑定的 worker slot 范围。"""  end = slot_offset + slot_count  if end > self.max_dp_size:  raise ValueError(  f"[Elastic EP] add_elastic_workers: slot_offset={slot_offset}+ "  f"slot_count={slot_count}exceeds max_dp_size={self.max_dp_size}. "  f"Restart with a larger --max-ep-size."  )  for slot in range(slot_offset, end):  self.dp_active[slot] = True  self.status[slot] = True  # 此处需要确保 worker socket 已在初始化时创建  if self.workers[slot] is None:  # 实际 worker 进程在 launch 时已经启动，这里只是激活通信  self.workers[slot] = self._make_worker_socket(slot)  self._refresh_active_workers()  logger.debug(  "[Elastic EP][DPC] activated workers: slots%d..%d, active_workers=%s", slot_offset,  end - 1,  self._active_workers,  )
```

注意：此函数在 _finalize_scale_up 中由 model_runner 调用，确保在 DP 级别新增的分发目标被激活。

评论区精华

- ishandhanani 指出 joiner 目标必须与 API 请求一致：若现有规模 EP=4，加入 TP=2 的 joiner 期望 EP=6，但 API 请求 new_ep_size=5 会导致死锁。zackyoray 已修复：在 handle_scale_elastic_ep 中对比 cohort 注册的 target 与请求值，不一致则拒绝。
- ishandhanani 指出 NIXL combine 路径每层调用 sync_active_to_cpu 造成性能回归：这会引入 GPU→CPU 同步。zackyoray 移除了该调用，改为只在 mask 变化时（forward 结束时）更新 CPU 副本。
- ch-wan 指出空集群无 forward 调用时扩展停滞：maybe_join_ep_ranks 只在 forward 路径中执行，若集群无请求则 joiners 永远不被提交。zackyoray 认为这是设计限制，需后续 PR 在 scheduler 事件循环中驱动。
- ch-wan 指出超时决策非集体性可能导致死锁：各 rank 本地判断超时，不同步则一部分退出而另一部分仍阻塞在 collective 中。zackyoray 承诺改为集体超时判定（通过 WORLD CPU group allreduce）。
- ch-wan 指出 post-scale 后任一 rank 故障导致持续报错：因为 has_scaled 为 true 且 is_scaling() 判定为 true，每次 forward 都会 raise RuntimeError。zackyoray 改为记录一次错误并通过状态暴露。

- UNIDY2002 质疑 world_dp_gather_enabled() 和 enable_joiner_all_gather() 的简洁性：zackyoray 同意并内联了逻辑。
- gemini-code-assist[bot] 指出 model_runner 缺少 enable_elastic_ep 属性：但该属性其实在 __init__ 中有设置（self.enable_elastic_ep = server_args.elastic_ep_backend is not None），属误报。
  - Joiner target 与 API 请求必须一致 (correctness): zackyoray 修改 handle_scale_elastic_ep 在开始缩放前比较 cohort 注册的 target 与请求值，不一致则拒绝。
  - NIXL combine 路径每层 GPU 同步导致性能回归 (performance): zackyoray 移除了 combine 中的同步，改为只在 forward 结束时（mask 变化时）更新 CPU 副本。
  - 空集群无请求时扩展停滞 (correctness): zackyoray 承认这是限制，计划在后续 PR 中从 scheduler 事件循环驱动。
  - 超时决策非集体性可能导致死锁 (correctness): zackyoray 承诺改为集体超时判定（通过 WORLD CPU group allreduce）。
  - Post-scale 故障恢复不受支持且报错不断 (correctness): zackyoray 改为记录一次错误并通过状态暴露，不崩溃。
  - world_dp_gather_enabled 辅助函数必要性 (design): zackyoray 同意并内联逻辑，移除辅助函数。
  - model_runner.enable_elastic_ep 属性误判 (correctness): zackyoray 指出该属性在 __init__ 中已设置（self.enable_elastic_ep = server_args.elastic_ep_backend is not None），属误报。

风险与影响

- 风险：
  - 空集群扩展停滞（High）：maybe_join_ep_ranks 只在前向路径执行，若当前无请求则缩放请求永远不会被处理，joiners 可能超时。需后续 PR 在 scheduler 事件循环中主动推进。
  - 超时决策非集体性（Medium）：当前各 rank 独立基于 time.monotonic 判定超时并退出，可能导致部分 rank 继续执行而部分退出，引发 collectives 死锁。已计划改为集体 allreduce。
  - post-scale 恢复未支持（Medium）：扩展后若已有 rank 故障，is_scaling 持续返回 true 且报错，无法降级运行。需后续 PR。
  - 特定拓扑限制（Low）：当前仅支持 dp_size == tp_size 且 attn_cp_size == 1 的拓扑。文档中已说明。
  - 性能风险（已缓解）：NIXL combine 路径曾引入每层 GPU 同步，已移除。但仍需验证稳态 TPOT。
- 影响：
  - 用户：提供动态扩展能力，无需重启即可增加计算资源，降低运维成本。但当前仅支持扩展（scale-up），不支持收缩（scale-down）。

- 系统：增加 elastic_ep_scale_timeout、ep_join_mode 等配置项；ElasticEPState 数据结构复杂度提升；控制平面新增 IPC 消息类型 ElasticScaleUpdateReq。
- 团队：此为系列首 PR，后续还需收缩、拓扑扩展、自动恢复等。代码量较大（+2547 行），需仔细 review 状态机正确性。
- 风险标记：空集群扩展停滞，超时决策非集体性，post-scale 恢复不可用，特定拓扑限制（dp_size==tp_size）

关联脉络

- PR #22788 [RFC] Elastic EP Scale: 本 PR 实现该 RFC 的 scale-up 部分，定义了动机和整体架构。
- PR #15771 Elastic EP rank recovery: 提供基础弹性 EP 恢复机制（active_ranks、join_process_groups），本 PR 复用其模型扩展 scale-up。
- PR #30535 [hicache]: add mamba_io_kernel: 与本 PR 无直接代码依赖，但同为弹性 EP 生态中的传输优化。
- PR #31380 [Spec] Consolidate the verify step into eagle_worker_common.run_eagle_verify: 与本 PR 属不同功能线（speculative decoding vs elastic EP），但在 model_runner 中有文件冲突，合并时需协调。