PR #33925 完整报告

sgl-project/sglang

config: route DCP topology reads through get_parallel()

合并时间：2026-08-08 05:53

原文链接：http://prhub.com.cn/sgl-project/sglang/pull/33925

执行摘要

- 一句话：DCP 拓扑读取全面迁移到 get_parallel() 实时来源
- 推荐动作：值得精读。本 PR 是 " 配置种子 → 实时拓扑 " 渐进迁移的样板：no-assert 访问器三元组（dcp_enabled / attn_dcp_size / attn_dcp_rank）的设计、" 哪些读取可以迁移、哪些必须留在 server_args" 的边界判断，以及自审式 review 中对命名属性、注释和局部变量的删减，都是可复用的工程实践。

功能与动机

33170 已将 106 个并行配置叶读取改经 get_parallel()，但故意留下 5 个 live-shadowed 拓扑尺寸（tp/pp/dcp/attn_cp/moe_dp_size）在 server_args 上，因为实时 property 在访问器上胜出，而条件初始化的组会在无条件调用点 fail loud。DCP 恰是典型：_DCP 只在 dcp_size > 1 时安装，DCP 关闭时直接读 get_parallel().dcp_size 会断言。本 PR 用 attention 后端已用的 no-assert 三元组化解，让 13 个分布式初始化后运行的读取点安全读取实时拓扑而非配置种子。

实现拆解

1. 访问器契约：迁移的调用点统一使用 dcp_enabled（布尔门控）、attn_dcp_size（KV 池加宽系数与聚合边界）、attn_dcp_rank（复制权重索引）三个 no-assert 访问器；DCP 关闭或分布式初始化前分别安全返回 False / 1 / 0，避免 get_parallel().dcp_size 的断言。
2. 调用点迁移（10 个源码文件）：attention_registry.py 的 dcp_size > 1 → dcp_enabled；dsa/utils.py 的 dcp_size == 1 → not dcp_enabled；base_runner.py 的 fi_a2a 工作空间预分配门控改为 dcp_enabled and dcp_comm_backend == "fi_a2a"；pool_configurator.py 的 dflash cell-size 缩放改用 attn_dcp_size；allocation.py 的页大小分支改用 dcp_enabled；disaggregation/common/conn.py 改用 parallel.attn_dcp_size / attn_dcp_rank。每个位置只换数据源，不改决策逻辑。
3. model_runner.py 初始化顺序调整：self.dcp_size / self.dcp_rank 从 __init__ 开头挪到 init_torch_distributed() 之后、从实时拓扑读取；原 ps.tp_rank % dcp_size 算术随之删除

——DCP group 按每个 TP group 的连续 dcp_size 分片构建，tp_rank % dcp_size 恰好等于 attn_dcp_rank。q-proj 复制门控也改为 dcp_enabled and dcp_replicate_q_proj。

4. 调度器与 KV 缓存聚合边界：scheduler.py 的两处 max_total_num_tokens * dcp_size（init_pool_stats_observer 与 init_req_max_new_tokens 的请求长度封顶）改读 get_parallel().attn_dcp_size；kv_cache_configurator.py 的 4 处（loc_space_scale、分配器选型分支、PagedTokenToKVPoolAllocator 的容量与页大小加宽）同步迁移。
5. 清理与配套：invariant_checker.py 删除 server_args dataclass 字段及其两个消费点；allocation.py 失去 test_global_config_read_ratchet.py 中的 config-intent 豁免；测试更新 4 个文件：test_scheduler_init_req_max_new_tokens.py 删除 SimpleNamespace(dcp_size=1) stand-in、test_disaggregation_wire.py 删除失效的 dcp_size=1 字段、test_dcp_layout_unit.py 改为驱动真实 get_parallel() 路径、ratchet 基线复核（direct 0 / alias 12 不变）。

关键文件：

- python/sglang/srt/managers/scheduler.py（模块 调度器；类别 source；类型 core-logic；符号 Scheduler.init_pool_stats_observer, Scheduler.init_invariant_checker, Scheduler.init_req_max_new_tokens）：核心调度路径：两处 max_total_num_tokens * dcp_size 聚合边界（池统计观测与请求 max_new_tokens 封顶）改读实时 attn_dcp_size，并移除传给 invariant_checker 的 server_args 参数。
- python/sglang/srt/model_executor/model_runner.py（模块 模型运行；类别 source；类型 data-contract；符号 ModelRunner.init, ModelRunner.init_attention_backends）：dcp_size / dcp_rank 从构造器开头移到 init_torch_distributed() 之后并读取实时拓扑，删除 tp_rank % dcp_size 手工算术；q-proj 复制门控改用 dcp_enabled。
- python/sglang/srt/managers/scheduler_components/invariant_checker.py（模块 不变量检查；类别 source；类型 dependency-wiring；符号 SchedulerInvariantChecker, SchedulerInvariantChecker._check_full_pool）：删除 server_args dataclass 字段及其两个消费点，两处防御性 getattr(self.server_args, "dcp_size", 1) 改为 get_parallel().dcp_enabled，是迁移中依赖清理最彻底的文件。
- python/sglang/srt/mem_cache/kv_cache_configurator.py（模块 KV 缓存；类别 source；类型 core-logic；符号 KVCacheConfigurator.loc_space_scale, KVCacheConfigurator._build_token_to_kv_pool_allocator）：DCP KV 池加宽的 4 处核心计算（loc_space_scale、分配器选型、PagedTokenToKVPoolAllocator 容量与页大小）全部改读实时拓扑，是影响显存布局的关键路径。
- python/sglang/srt/model_executor/runner/base_runner.py（模块 模型运行；类别 source；类型 data-contract；符号 BaseRunner._pre_initialize_fi_a2a_workspace）：fi_a2a DCP 工作空间预分配门控从 server_args 迁移到实时拓扑，且 dcp_comm_backend 移入并行包后能反映 publish-time 覆盖。
- python/sglang/srt/mem_cache/allocation.py（模块 内存分配；类别 source；类型 dependency-wiring；符号 allocation._alloc_page_size）：页大小分支改经实时 dcp_enabled，同时失去 test_global_config_read_ratchet.py 中的 config-intent 豁免，是 ratchet 基线变化的关键文件。

- python/sglang/srt/disaggregation/common/conn.py（模块 断开连接；类别 source；类型 core-logic）：断开连接公共连接层改用 parallel.attn_dcp_size / attn_dcp_rank，与 #36025 的改动在同一文件接续。
- python/sglang/srt/model_executor/pool_configurator.py（模块 内存池；类别 source；类型 data-contract）：dflash cell-size 缩放改经 attn_dcp_size，参与模型内存池布局决策。
- python/sglang/srt/layers/attention/attention_registry.py（模块 注意力；类别 source；类型 dependency-wiring）：attention 后端注册的 DCP 门控改经 dcp_enabled，与 DCP 注意力后端选择直接相关。
- python/sglang/srt/layers/attention/dsa/utils.py（模块 注意力；类别 source；类型 core-logic；符号 dsa.utils.should_remap_pd_dsa_seed_to_local_slots）：DSA 重映射决策从 server_args.dcp_size == 1 改为 not dcp_enabled，并删除了冗余 docstring。
- test/registered/unit/managers/test_scheduler_init_req_max_new_tokens.py（模块 测试；类别 test；类型 test-coverage）：Fixture 不再需要 SimpleNamespace(dcp_size=1) stand-in，直接删除，验证了迁移后测试契约简化。
- test/registered/dcp/test_dcp_layout_unit.py（模块 测试；类别 test；类型 test-coverage）：分配器加宽测试改为驱动真实 get_parallel() 路径，反映全局单例耦合下的测试调整策略。
- test/registered/disaggregation/test_disaggregation_wire.py（模块 测试；类别 test；类型 test-coverage）：删除 server_args stand-in 中已失效的 dcp_size=1 字段，与 conn.py 迁移配套。
- test/registered/unit/config/test_global_config_read_ratchet.py（模块 测试；类别 test；类型 test-coverage）：allocation.py 失去 config-intent 豁免，ratchet 基线复核为 direct 0 / alias 12 不变。

关键符号：ModelRunner.init, ModelRunner.init_attention_backends, BaseRunner._pre_initialize_fi_a2a_workspace, Scheduler.init_pool_stats_observer, Scheduler.init_invariant_checker, Scheduler.init_req_max_new_tokens, SchedulerInvariantChecker._check_full_pool, KVCacheConfigurator.loc_space_scale, KVCacheConfigurator._build_token_to_kv_pool_allocator, allocation._alloc_page_size, dsa.utils.should_remap_pd_dsa_seed_to_local_slots

关键源码片段

python/sglang/srt/managers/scheduler.py

核心调度路径：两处 max_total_num_tokens * dcp_size 聚合边界（池统计观测与请求 max_new_tokens 封顶）改读实时 attn_dcp_size，并移除传给 invariant_checker 的 server_args 参数。

```
# scheduler.py —— 请求上限与池观测的聚合边界统一走实时拓扑。
# DCP 会把 KV pool 分片到各 rank，所以单请求上限与池统计都应以
# 聚合池（本 rank 份额 x attn_dcp_size）为准，而非本 rank 份额。

def init_pool_stats_observer(self) -> None:
    self.pool_stats_observer = SchedulerPoolStatsObserver(
```

```
        ...
        # DCP 开启时即聚合池大小；attn_dcp_size 在 DCP 关闭时返回 1，
        # 所以这里无需像旧代码那样读 server_args.dcp_size 配置种子
        max_total_num_tokens=self.max_total_num_tokens
        * get_parallel().attn_dcp_size,
        ...
    )

def init_req_max_new_tokens(self, req):
    ...
    req.sampling_params.max_new_tokens = max(
        0,
        min(
            max_new_tokens,
            self.max_req_len - input_len - 1,
            # 与 PrefillAdder 的准入预算保持一致：ceil_page(input_len)
            # + max_new_tokens + page_size 必须严格小于聚合的
            # max_total_num_tokens，否则请求进等待队列却永远无法被调度，
            # 最终导致健康检查失败
            self.max_total_num_tokens * get_parallel().attn_dcp_size
            - paged_input_len - self.page_size - 1,
        ),
    )
```

python/sglang/srt/model_executor/model_runner.py

dcp_size / dcp_rank 从构造器开头移到 init_torch_distributed() 之后并读取实时拓扑，删除 tp_rank % dcp_size 手工算术；q-proj 复制门控改用 dcp_enabled。

```
# model_runner.py —— DCP 拓扑读取从配置种子迁移到实时来源。
# 关键点：DCP group 只在分布式初始化后安装，所以 dcp_size / dcp_rank
# 必须放在 init_torch_distributed() 之后读取，否则会拿到 server_args 的
# 配置种子；而 DCP group 按 TP group 的连续 dcp_size 分片构建，
# 原 ps.tp_rank % dcp_size 恰好等于 rank 在 DCP group 内的索引，
# 所以直接用权威来源 attn_dcp_rank 即可。

def __init__(self, model_config, mem_fraction_static, gpu_id, ps, ...):
    self.ps = ps
    self.model_config = model_config
    ...
    # Mooncake TransferEngine 必须在分布式初始化前创建，以复用共享 TE
    self.init_shared_mooncake_transfer_engine()
    # 在这里之前 DCP group 不存在，提前读 attn_dcp_size 会静默拿到 1
    self.init_torch_distributed()

    # 从实时拓扑读取，而不是配置种子
    self.dcp_size = get_parallel().attn_dcp_size
    self.dcp_rank = get_parallel().attn_dcp_rank
    ...
    # q-proj 复制门控同样用 dcp_enabled（实时）而非 server_args.dcp_size：
```

```
    # DCP 关闭时 dcp_enabled 安全返回 False（经 get_dcp_group_no_assert）
    if get_parallel().dcp_enabled and get_parallel().dcp_replicate_q_proj:
        self._prepare_replicated_q_proj()
```

python/sglang/srt/managers/scheduler_components/invariant_checker.py

删除 server_args dataclass 字段及其两个消费点，两处防御性 getattr(self.server_args, "dcp_size", 1) 改为 get_parallel().dcp_enabled，是迁移中依赖清理最彻底的文件。

```
# invariant_checker.py —— 防御性 getattr 读取让位于 no-assert 实时访问器。
# 之前：getattr(self.server_args, "dcp_size", 1) > 1（配置种子 + 防御默认值）
# 现在：get_parallel().dcp_enabled（实时拓扑，无断言、无默认值兜底）

def _check_full_pool(self, ps: PoolStats, uncached: int = 0) -> Tuple[bool, str]:
    ...
    full_evictable_size = ps.full_evictable_size
    allocator = self.token_to_kv_pool_allocator
    if get_parallel().dcp_enabled and allocator.page_size > 1:
        # DCP 在加宽后的物理页里存逻辑 token：前缀缓存按逻辑 token 计数，
        # 而分配器按整页释放，所以要把缓存的 token 向上取整到物理页单位
        full_evictable_size = (
            (full_evictable_size + allocator.page_size - 1)
            // allocator.page_size
            * allocator.page_size
        )
    leak, msg = self._check_pool_invariant(
        "full",
        ps.full_available_size,
        full_evictable_size,
        protected,
        session_held,
        total,
        uncached,
    )
    if leak and get_parallel().dcp_enabled and allocator.page_size > 1:
        # 前缀缓存会计按逻辑 token，而 DCP full KV 分配按物理页进行，
        # 即使所有页都被持有，也可能残留页面级 slack，需要单独提示
        ...
```

评论区精华

本 PR 的 10 条 review 评论全部来自作者本人（kpham-sgl），是 Claude Code 生成代码后的自审模式，核心交锋：

- "Seems redundant?"（model_runner.py）：dcp_size / dcp_rank 读取上方的说明注释被判冗余后删除。
- "dont need to add redundant comment"（dsa/utils.py）：docstring 中解释 DCP 实时拓扑来源的段落被要求删掉。

- "its okay you can call get_parallel(), no need to spawn object"（base_runner.py）：parallel = get_parallel() 局部绑定被改为每个使用点直接调用。
- "just in-line get_parallel().attn_dcp_size later"（scheduler.py）：作者原本引入 aggregate_max_total_num_tokens 命名属性，自审后认为扩大 API 面，改为两处直接内联。
- "why is this needed?"（test_scheduler_init_req_max_new_tokens.py）：fixture 补 stub 的改动被追问，结论是 " 完全不需要 "——attn_dcp_size 在无 DCP group 时经 get_dcp_group_no_assert() 返回 1，且 get_parallel() 是模块级单例、没有上下文可注入，直接删行而非替换。
- aggregate_max_total_num_tokens 命名属性 vs 直接内联 (design): 放弃命名属性，两处调用点直接内联 get_parallel().attn_dcp_size，相应测试 stub 也一并删除。
- 测试 fixture 为何需要补 aggregate_max_total_num_tokens stub (testing): 完全不需要：attn_dcp_size 在无 DCP group 时经 get_dcp_group_no_assert() 返回 1，且 get_parallel() 是模块级单例、没有任何可注入的上下文，直接删除该行而非替换。
- get_parallel() 是否需要局部绑定 (style): 删除局部绑定，每个使用点直接调用 get_parallel()，减少多余变量。
- dsa/utils.py 冗余 docstring (style): 删除新增的 docstring 段落。
- model_runner.py DCP 读取上方的注释是否冗余 (style): 删除注释，保留代码本身的清晰性。

风险与影响

- 风险：
  1. model_runner.py 初始化顺序变更：self.dcp_size / self.dcp_rank 从构造器开头移到 init_torch_distributed() 之后，任何在分布式初始化前读取这两个属性的调用者会从 " 拿到配置种子 " 变为 AttributeError，需确认无此类调用点。
  2. 缺少 DCP e2e 覆盖：作者声明没有 GPU 机箱，最终依赖 CI 重跑（test_kimi_linear_pd_dcp4.py、test_dcp_layout_unit.py、test_dsv31_dcp8_gsm8k.py、test_reduce_scatter_along_dim.py）验证，多 GPU 验证仍薄弱。
  3. 测试与全局单例耦合：get_parallel() 是模块级单例、无法注入 stub，test_dcp_layout_unit.py 因此必须真实走 DCP 路径才能驱动分配器加宽逻辑，单测隔离性被削弱。
  4. publish-time 覆盖语义变化：dcp_comm_backend 移入并行包（resolvable=True）后开始反映发布后的覆盖值，任何仍从 server_args 实例读取该字段的代码可能短暂看到不一致值。
- 影响：
  1. 架构影响：跨 14 个文件统一 DCP 拓扑数据源，消除配置种子与实时拓扑的分叉，为后续 attn_cp / moe_dp_size 等拓扑尺寸迁移搭好模式。
  2. 用户与运维影响：publish-time 覆盖（如 dcp_comm_backend）现在能被正确反映，提升动态配置场景的一致性；预期无面向用户的可见行为变化。
  3. 团队影响：allocation.py 失去 ratchet 豁免、测试 fixture 契约简化，降低了未来改动的认知负担；但 model_runner.py 初始化顺序变更要求后续 PR 必须遵守 "DCP 读取必

须晚于分布式初始化 " 的隐式约束。 - 风险标记：模型初始化顺序变更，缺少 DCP *e2e* 覆盖，测试耦合全局单例，核心路径访问器契约

关联脉络

- PR #33170 route parallel config-leaf reads through get_parallel(): 本 PR 的直接前身，PR body 明确说明这是其 dcp_size 后续；#33170 路由了 106 个并行配置叶读取但故意留下 5 个 live-shadowed 拓扑尺寸，本 PR 完成其中 dcp_size 切片。
- PR #36025 [AMD][MORI] Deduplicate CP-replicated state transfers: 同样改动 disaggregation/common/conn.py 与 test_disaggregation_wire.py，同一文件上的接力改动，说明断开连接 /DCP 路径正在被多条并行主线打磨。