PR #34406 完整报告

sgl-project/sglang

TP/PP Consensus checker

合并时间：2026-08-21 01:36

原文链接：http://prhub.com.cn/sgl-project/sglang/pull/34406

执行摘要

- 一句话：新增 PP/TP 秩间共识检查器，提前定位分歧根因
- 推荐动作：值得精读，重点看 rank_consensus_checker.py 的装饰器设计：选择器归一化、self / cls 跳过、关闭时 import 阶段短路实现零开销、后台线程 + 专用 gloo 组的比对模型。建议与 #27010（HiCache PP 一致性修复）一起阅读，理解 PP 分歧的根因与验证链路。后续可关注 hzh0425 提出的 *e2e* 测试 follow-up。

功能与动机

PR body 明确指出：PP/TP 分歧发生时（如服务器挂起），拿到的 stacktrace 并不是分歧发生的真实时间点。例如 kv cache 可用内存因软件 bug 在各 rank 不同，同一请求在 TP0 被接受、TP1 被拒绝，batch size 开始分歧，最终服务器挂起——但该请求只是受害者而非触发器，必须找到触发器请求才能定位根因。因此引入 consensus checker 做早期分歧检测。此外，该检查器被用于复现 PP 场景 + HiCache L3 的分歧问题，并在几分钟内检测到 PP0 / PP1 的 check_prefetch_progress 结果不一致，验证了 #27010 的修复。

实现拆解

实现按以下 5 个步骤拆解：

1. 新增核心模块 python/sglang/srt/utils/rank_consensus_checker.py（新增 432 行）：提供 rank_consensus 装饰器与 assert_same 编程式接口。装饰器支持 same_params / same_results 的布尔值或字段选择器（如 "a.req_id"、"result.full_kv_hit_length"），通过 _normalize_selector 归一化，_build_payload 把函数名、参数或返回值序列化为可比较字符串；assert_same 将事件推入队列，后台工作线程消费事件并借助 configure 传入的专用 gloo 进程组与其他 rank 对齐比对，不一致即中止（os._exit）。
2. 环境开关与零开销设计：在 python/sglang/srt/environ.py 增加 SGLANG_ENABLE_RANK_CONSENSUS_CHECKER（默认 False）。装饰器在 import 阶段即短路返回原函数，保证关闭时零运行时开销；PR 内三组对比（基线 / 开启 / 关闭）验证无性能回退。
3. 调度器挂载：python/sglang/srt/managers/scheduler.py 新增 init_rank_consensus_checker()，在 Scheduler 初始化流程中按 attn_cp_group / attn_tp_group / tp_group / pp_group 收集分组并调用 rank_consensus_checker.configure(groups)；在 release_host_resources 中调用

shutdown() 做退出清理。

4. HiCache 关键路径接入：python/sglang/srt/mem_cache/unified_radix_cache.py 对 match_prefix（校验 params 与 full_kv_hit_length / swa_host_hit_length）、check_prefetch_progress（全参数全结果）、staged_prefetch_swa_tokens（全参数）、release_aborted_request（全参数）挂上装饰器，覆盖 PP 分歧最先出现的预取进度与前缀命中路径。
5. 测试与文档配套：新增 test/registered/cpu/test_rank_consensus_checker.py（586 行），用 torch.multiprocessing spawn 多进程 + gloo 后端在纯 CPU 环境模拟 PP/TP 拓扑，覆盖实例 / 类 / 静态方法、字段选择器求值、分歧触发 os._exit(1)、正常调用 os._exit(0) 等场景；docs/docs/references/environment_variables.mdx 补充环境变量说明。端到端测试按 review 意见留待 follow-up PR。

关键文件：

- python/sglang/srt/utils/rank_consensus_checker.py（模块 调试工具；类别 source；类型 new-module；符号 rank_consensus, assert_same, configure, shutdown）：PR 的核心新增模块（432 行），实现 @rank_consensus 装饰器、assert_same 编程接口、后台线程与跨 rank gloo 比对机制，是整套共识检查能力的载体。
- test/registered/cpu/test_rank_consensus_checker.py（模块 单元测试；类别 test；类型 test-coverage；符号 run_distributed_test, _cpu_init_model_parallel_group, _DummyClass, _MethodHost）：586 行 CPU-only 分布式单测，通过 spawn 多进程 + gloo 在无 GPU 环境模拟 PP/TP 拓扑（CUDA_VISIBLE_DEVICES=99 配合 patch 禁用 pynccl 与自定义 allreduce），覆盖实例 / 类 / 静态方法、字段选择器求值与分歧检测退出码约定。
- python/sglang/srt/managers/scheduler.py（模块 调度器；类别 source；类型 dependency-wiring；符号 init_rank_consensus_checker）：检查器在真实服务中的挂载点：启动时 init_rank_consensus_checker 收集 tp/pp/attention 组并 configure，退出路径调用 shutdown，是共识检查与调度线程生命周期的衔接处。
- python/sglang/srt/mem_cache/unified_radix_cache.py（模块 缓存层；类别 source；类型 dependency-wiring；符号 match_prefix, check_prefetch_progress, staged_prefetch_swa_tokens, release_aborted_request）：HiCache 关键路径接入点，对 match_prefix、check_prefetch_progress、staged_prefetch_swa_tokens、release_aborted_request 挂装饰器，覆盖 PP 分歧最先出现的预取进度与前缀命中路径，是 PR body 中实际抓到 PP0/PP1 分歧的位置。
- python/sglang/srt/environ.py（模块 环境配置；类别 source；类型 configuration；符号 SGLANG_ENABLE_RANK_CONSENSUS_CHECKER）：新增 SGLANG_ENABLE_RANK_CONSENSUS_CHECKER 环境变量（默认 False），是检查器开关的唯一定义处，决定装饰器是否在 import 阶段短路。
- docs/docs/references/environment_variables.mdx（模块 文档；类别 docs；类型 documentation）：为新增环境变量补充用户文档，说明其用途（检查 PP/TP 分歧并在分歧时中止服务）与默认值。

关键符号：rank_consensus, assert_same, configure, shutdown, init_rank_consensus_checker, match_prefix, check_prefetch_progress,

staged_prefetch_swa_tokens, release_aborted_request

关键源码片段

python/sglang/srt/utils/rank_consensus_checker.py

PR 的核心新增模块（432 行），实现 @rank_consensus 装饰器、assert_same 编程接口、后台线程与跨 rank gloo 比对机制，是整套共识检查能力的载体。

```
# rank_consensus_checker.py 核心设计：
# 被检查函数在调度线程被调用时，会把调用事件（函数名 + 参数 / 返回值的选定字段）
# 序列化为字符串，经由队列交给后台线程；后台线程通过专用 gloo 进程组
# 与其他 rank 对齐比对，一旦事件序列不一致，立即判定 divergence 并中止服务。

import functools
import inspect
from typing import Any, Callable, Optional

from sglang.srt.environ import envs


def rank_consensus(func=None, *, same_params=None, same_results=None, **kwargs):
    """标记一个函数，要求它在 PP / TP 各 rank 的调度线程中以相同顺序被调用，
    并可选择校验参数与返回值的一致性。

    @rank_consensus(same_params=True, same_results=True)
    def check_prefetch_progress(self, req_id: str) -> bool: ...

    # 也支持只校验部分参数或参数的某个字段，例如：
    # same_params=["len(foo)", "operation.req_id"]
    """
    if kwargs:
        raise TypeError(
            f"rank_consensus() got unexpected keyword argument(s): {list(kwargs)}"
        )

    # 把布尔值与字段选择器统一归一化；None 表示该维度不开启检查
    params_selector = _normalize_selector(same_params, "same_params")
    results_selector = _normalize_selector(same_results, "same_results")

    def decorator(func: Callable) -> Callable:
        # 装饰器在 import 时执行：检查器默认关闭，此时直接返回原函数，
        # 保证线上零运行时开销（PR 内基线对比验证无回退）
        if not envs.SGLANG_ENABLE_RANK_CONSENSUS_CHECKER.get():
            return func

        # 先解开 classmethod / staticmethod 描述符，统一对原始函数签名处理，
        # 并记住描述符类型，最后重新包装以保持类体描述符协议不变
        if isinstance(func, (classmethod, staticmethod)):
            raw_func = func.__func__
```

```
        descriptor_type = type(func)
    else:
        raw_func = func
        descriptor_type = None
    sig = inspect.signature(raw_func)

    # 对实例方法 / 类方法跳过第一个 self / cls 参数：
    # 其字符串形式可能包含内存地址，会让各 rank 误判为分歧
    skip_name: Optional[str] = None
    if _is_method_with_receiver(func) and len(sig.parameters) > 0:
        skip_name = next(iter(sig.parameters))

    @functools.wraps(raw_func)
    def wrapper(*args: Any, **kwargs: Any) -> Any:
        params_payload = "<no check>"
        if params_selector is not None:
            # 只在需要校验时才 bind，并应用默认值，
            # 使按名选择参数时与位置 / 关键字传参方式无关
            bound = sig.bind(*args, **kwargs)
            bound.apply_defaults()
            arguments = dict(bound.arguments)
            params_payload = _build_payload(
                "call", params_selector, arguments, skip_name
            )
        assert_same("%s called params=%s", raw_func.__name__, params_payload)

        result = raw_func(*args, **kwargs)

        result_payload = "<no check>"
        if results_selector is not None:
            # 返回值以 result 为名注入求值作用域，
            # 因此选择器可写 "result.full_kv_hit_length" 这类字段表达式
            result_scope = {"result": result}
            result_payload = _build_payload(
                "return", results_selector, result_scope,
            )
        assert_same("%s returns result=%s", raw_func.__name__, result_payload)
        return result

    # 按原描述符类型重新包装，classmethod / staticmethod 语义保持不变
    #  （_normalize_selector / _is_method_with_receiver / _build_payload /
    # assert_same 等内部函数的实现从略）
    if descriptor_type is not None:
        return descriptor_type(wrapper)
    return wrapper

# 同时支持裸用 @rank_consensus 与带参数 @rank_consensus(...) 两种形式
return decorator(func) if func is not None else decorator
```

test/registered/cpu/test_rank_consensus_checker.py

586 行 CPU-only 分布式单测，通过 spawn 多进程 + gloo 在无 GPU 环境模拟 PP/TP 拓扑（CUDA_VISIBLE_DEVICES=99 配合 patch 禁用 pynccl 与自定义 allreduce），覆盖实例 / 类 / 静态方法、字段选择器求值与分歧检测退出码约定。

```
# 让分布式测试在无 GPU 的 CPU 环境运行：
# 父进程把 CUDA_VISIBLE_DEVICES 设为 "99"（不存在的设备），
# 子进程重新 import 后 is_cuda_alike() 为 False，GroupCoordinator 自动选 CPU；
# 但 pynccl / 自定义 allreduce 等 CUDA 通信器无法在无 GPU 时构建，
# 因此这里 patch 掉 init_model_parallel_group，强制关闭这两类通信器。

def run_distributed_test(rank, world_size, pp_size, tp_size, master_port, fn) -> None:
    """子进程入口：初始化 gloo 分布式环境后运行 fn。

    退出码约定：
      * 0  -> fn 正常结束
      * 1  -> 检查器检测到分歧，其工作线程调用 os._exit(1)
      * 2  -> fn 抛异常（测试搭建 / 场景 bug）
    """
    ps.set_custom_all_reduce(False)

    def _cpu_init_model_parallel_group(
        *args, _orig=ps.init_model_parallel_group, **kwargs
    ):
        # initialize_model_parallel 没有禁用 pynccl 的开关，
        # 这里统一强制关闭，同时关闭自定义 allreduce
        kwargs.setdefault("use_pynccl", False)
        kwargs.setdefault("use_custom_allreduce", False)
        return _orig(*args, **kwargs)

    with patch.object(ps, "init_model_parallel_group", _cpu_init_model_parallel_group):
        try:
            os.environ["RANK"] = str(rank)
            os.environ["WORLD_SIZE"] = str(world_size)
            os.environ["MASTER_ADDR"] = "localhost"
            os.environ["MASTER_PORT"] = str(master_port)
            os.environ["LOCAL_SIZE"] = str(world_size)
            init_distributed_environment(
                world_size=world_size, rank=rank,
                distributed_init_method="env://", local_rank=rank,
                backend="gloo",
            )
            initialize_model_parallel(
                tensor_model_parallel_size=tp_size,
                pipeline_model_parallel_size=pp_size,
                backend="gloo",
            )
            fn()
```

```
        except Exception as e:
            traceback.print_exc()
            os._exit(2)
        finally:
            # patch.object 自动恢复，包括 os._exit(2) 的错误路径
            if dist.is_initialized():
                dist.destroy_process_group()
```

python/sglang/srt/mem_cache/unified_radix_cache.py

HiCache 关键路径接入点，对 match_prefix、check_prefetch_progress、staged_prefetch_swa_tokens、release_aborted_request 挂装饰器，覆盖 PP 分歧最先出现的预取进度与前缀命中路径，是 PR body 中实际抓到 PP0/PP1 分歧的位置。

```
# unified_radix_cache.py —— HiCache 关键路径接入 rank 共识检查：
# 这些装饰器只在 SGLANG_ENABLE_RANK_CONSENSUS_CHECKER=1 时生效，
# 用于在 PP 场景下尽早发现各 rank 前缀匹配与预取进度不一致的请求。

from sglang.srt.utils.rank_consensus_checker import rank_consensus


class UnifiedRadixCache:
    @rank_consensus(
        same_params=["params"],
        same_results=["result.full_kv_hit_length", "result.swa_host_hit_length"],
    )
    def match_prefix(self, params: MatchPrefixParams) -> MatchResult:
        # 各 rank 的命中长度必须一致：若某个 rank 因 L3 存储长度不同
        # 而命中不同前缀，这里会被检查器第一时间抓到
        result = self.session.try_match_prefix(params)
        if result is not None:
            return result
        if self.disable:
            return self.tree_core.empty_match_result
        result = self.tree_core.match_prefix(params)
        # 在 finalizer 之前应用 walk 产生的动作（如 split 引发的写穿透搬迁）
        self._apply_cache_actions(result.cache_actions)
        for component in self._components_tuple:
            result = component.finalize_match_result_in_cache(params, result)
        # finalizer 不允许再产生动作，walk 的动作已在上面应用
        assert not result.cache_actions
        return result

    @rank_consensus(same_params=True, same_results=True)
    def check_prefetch_progress(self, req_id: str) -> bool:
        # PR 描述中实际抓到 PP0 / PP1 分歧的函数：
        # 当某个请求在某 rank 被接受、另一 rank 被拒绝时，
        # batch 尺寸开始分歧并最终导致服务器挂起
        if req_id not in self.ongoing_prefetch:
            return True
```

```
    # ...（后续预取进度检查逻辑从略）
```

评论区精华

核心讨论集中在 rank_consensus_checker.py 的性能与正确性细节上：

- whybeyoung 提出两处热路径优化建议：_is_method_with_receiver(func) and next(iter(sig.parameters)) 应在 decorator(func) 阶段只计算一次而非每次调用；sig.bind 只在 params_selector is not None 时执行。作者均回复 Fixed，保证 wrapper 路径无冗余开销。
- ShangmingCai 抓到一个 f-string typo，作者确认修复。
- hzh0425 建议补充 rank checker 的端到端测试，作者同意放到 follow-up PR 中。
- 审阅结论：whybeyoung 先 CHANGES_REQUESTED 后 APPROVED，ShangmingCai、hzh0425 均 APPROVED，ShangmingCai 评价 "Really good feature for debugging!"。
- wrapper 热路径上的重复计算应上移到装饰器阶段 (performance): 作者回复 Fixed，两处均已调整，保证关闭检查器时零开销、开启时 wrapper 路径无冗余计算。
- TypeError 报错中的 f-string typo (style): 作者确认并修复（"Fixed. Nice catch."）。
- 缺少 rank checker 的端到端测试 (testing): 作者同意并计划在 follow-up PR 中补充，本 PR 仅含 CPU gloo 多进程单测。
- CI 失败是否由本 PR 引入 (question): 确认与本 PR 无关，主分支已修复。

风险与影响

- 风险：主要风险如下：

1. 误报即杀服务：检查器开启后，任何 payload 构造不一致（如未正确跳过 self / cls 的内存地址、集合迭代顺序不确定）都会让服务直接中止。PR 已通过跳过首个 receiver 参数缓解，但生产环境开启仍需谨慎。
2. 调度线程路径新增同步点：assert_same 在调度线程内入队，若后台线程消费或跨 rank gloo 通信出现异常，可能影响调度循环；shutdown() 的退出清理路径需要持续验证。
3. 装饰器描述符处理：对 classmethod / staticmethod 的解包与重新包装若与描述符协议不吻合，可能改变被装饰函数语义；match_prefix 是热点路径，回归影响面集中在 HiCache + PP 场景。
4. 测试依赖退出码约定：测试通过子进程 os._exit(1) / os._exit(2) 区分分歧与异常，CI 环境下（如 MUSA / AMD）行为可能不同。 - 影响：对线上用户与系统：默认关闭，零性能影响与零行为变更。对排查 PP/TP 挂起与 HiCache L3 分歧的团队而言，这是一套新的诊断手段，能把定位时间从「服务器挂起后翻 stacktrace」缩短到「分歧发生即中止并打印现场」。对代码库：新增了一个可复用的调试子系统（装饰器 + 同步组模型），未来可在更多调度线程函数上接入；对 #27010 这类分歧修复的验证效率有直接提升。团队协作上，该 PR 由 stepinto 主导、三位 reviewer 参与，讨论聚焦于性能细节与测试补全。 - 风险标记：默认关闭的调试开关，开启后分歧误报即中止服务，调度线程路径新增同步点，端到端测试待补全，依赖 gloo 同步组

关联脉络

- PR #27010 [HiCache] Fix PP inconsistency with HiCache L3 (#22607): PR body 明确说明：开启检查器后几分钟内检测到 PP 分歧，"The problem is fixed after patching #27010"，本 PR 正是用来复现、定位并验证该修复的调试工具。
- PR #35689 Skip empty linear-attention state buffers in PD transfer: 同属跨 rank 状态不一致类问题（PD 传输空状态 buffer 导致 Inkling 失败），与 PP/TP 一致性排查主题同源，可视为该调试工具后续继续发挥作用的场景。