PR #33171 完整报告

sgl-project/sglang

test: recover the config-namespace-migration deferrals

合并时间：2026-08-01 23:59

原文链接：http://prhub.com.cn/sgl-project/sglang/pull/33171

执行摘要

- 一句话：恢复配置迁移期跳过的 15 个单测文件，退役 deferral ratchet
- 推荐动作：值得精读。两类读者收益最大：(1) 正在做大规模内部 API 迁移的团队——deferral ratchet（钉住基线 + 双向断言）是管理迁移测试债务的极佳实践，本 PR 展示了它的完整生命周期（建立、履行、退役）；(2) SGLang runtime context 相关开发者——本 PR 是 override_server_args、bag 断言、各命名空间访问器用法的活教材。radix cache 用例的改造尤其值得看：它展示了当测试在"跳过期"长大、且与生产语义发生冲突时，如何重新构造等价状态而非强行适配。

功能与动机

PR body 明确说明：15 个单测文件在 namespace 迁移期间被 module-skip，原因是它们的 fixture 注入配置的方式对 namespace accessors 不可见（伪造 get_server_args 模块绑定、post-publish 往 ServerArgs 实例写字段）。deferral ratchet 将这 15 个文件钉死在基线上，使债务不能无声增长；本 PR 全部恢复并退役 ratchet——"its job done — is retired"。

实现拆解

1. 批量恢复 14 个 deferral 文件（Commit 1，a309036）：删除 pytestmark = skip 与 setUpModule() 跳过块，改用三种模式恢复——(a) 配置发布：通过 get_context().override_server_args(...).install() 配合 addCleanup(override.restore) 把被测代码读取的配置发布进 runtime context，作用域仅限本次测试，涉及 test_deepseek_v4_shared_expert_fusion.py、test_register_to_bootstrap.py、test_eagle_worker_v2_topk1_fastpath.py、test_fused_moe_triton_config.py、test_tbo_filter_batch_marker.py 等；(b) bag 断言：原断言 server_args 实例字段写穿（如 server_args.disable_shared_experts_fusion）改为断言 bag 状态（如 get_exec().moe.disable_shared_experts_fusion），与 declare_load_time_override 的 bag-only 语义一致；(c) 桩字段扩充：test_pool_configurator.py 的 model runner 桩补 context_len、max_total_tokens，test_max_running_requests.py 引入 _published() 发布辅助并以 ps.attn_dp_size 替代裸 dp_size，test_register_to_bootstrap.py 用 kv_cache_dtype_str 替代 server_args.kv_cache_dtype（对应生产代码 KVArgs payload 字段迁移）。另有 5 个文件（test_mm_process_config.py、test_tokenizer_manager_rid_cleanup.py、test_dllm_fdfo_kv_reuse.py、test_grammar_manager.py、test_fp32_lm_head.py、

test_priority_scheduling_disaggregation.py）仅删跳过块即可恢复。

2. 单独恢复 radix cache 文件（Commit 2，78318da）：test_unified_radix_cache_unittest.py 在跳过期间长大，且两个用例与迁移后语义冲突——test_cache_finished_req_strips_thinking（19 个参数化类）原直写 get_server_args().strip_thinking_cache，改为 get_serving().override(strip_thinking_cache=True) 的 serving bag 作用域覆盖；test_shallower_crossing_backs_up_above_backuped_middle 原通过 insert_host 构造"备份链中间断裂"状态，而 write-through 下 insert_host 会故意丢弃未备份节点下方的 refill（host_insert_dropped），改为显式 _write_backup + writing_check + evict 构造"中间节点已备份后被设备驱逐"的等价状态。
3. 退役 deferral ratchet：删除 test/registered/unit/test_migration_deferral_ratchet.py（基线 15 → 0）。该文件此前双向守卫：deferral 数超过基线即失败（防新 skip 混入），低于基线也失败（逼恢复者同步下调基线锁定成果）。
4. 验证与配套：全部 15 个文件按文件粒度（CI 语义）与单进程全量各跑一遍，证明无跨文件泄漏；radix 文件单文件 1039 个用例通过；全量套件双跑新增 120 个通过用例。评审中 2 个 P1（MLX 配置发布与 server_args 占位）与 3 个自审建议均在最终 head 修复。

关键文件：

- test/registered/unit/test_migration_deferral_ratchet.py（模块 迁移护栏；类别 test；类型 deletion；符号 TestMigrationDeferralRatchet, test_deferred_test_files_match_the_baseline）：整个守卫文件被删除，基线 15 → 0，是迁移债务清零的标志性动作。理解它才能理解本 PR 的治理框架。
- test/registered/unit/mem_cache/test_unified_radix_cache_unittest.py（模块 前缀缓存；类别 test；类型 test-coverage；符号 setUpModule, test_cache_finished_req_strips_thinking, test_shallower_crossing_backs_up_above_backuped_middle）：恢复最复杂的文件：在跳过期间长大了一批 hicache/insert-walk 用例，两个核心用例因 write-through 语义变化需要重构测试场景而非简单恢复 fixture。
- test/registered/unit/models/test_deepseek_v4_shared_expert_fusion.py（模块 融合策略；类别 test；类型 test-coverage；符号 setUpModule, setUp, tearDown, TestDeepseekV4SharedExpertFusionPolicy）：最能体现 bag-only 断言模式的恢复样本：从断言 ServerArgs 实例写穿改为断言 get_exec() 命名空间，与 declare_load_time_override 语义对齐。
- test/registered/unit/hardware_backend/mlx/test_max_running_requests.py（模块 并发上限；类别 test；类型 test-coverage；符号 setUpModule, _published, _stub, _hybrid_stub_for_initialize）：评审交锋最集中的文件（2 个 P1 均在此）：展示了从 stub.server_args 直写迁移到 _published() 发布辅助 + ps 命名空间的完整模式。
- test/registered/unit/spec/test_eagle_worker_v2_topk1_fastpath.py（模块 投机解码；类别 test；类型 test-coverage；符号 setUpModule, setUp, TestEagleWorkerV2Topk1FastPath, TestEagleWorkerV2BackendFallback）：展示了 setUp 粒度发布配置的恢复模式，并含一条有价值的自审讨论（cuda_graph_max_bs_decode 无效 seed 的处理）。

关键符号：get_context().override_server_args, _published, TestMigrationDeferralRatchet.test_deferred_test_files_match_the_baseline,

TestEagleWorkerV2Topk1FastPath.setUp, TestRegisterToBootstrap.setUp,test_cache_finished_req_strips_thinking, test_shallower_crossing_backs_up_above_backuped_middle

关键源码片段

test/registered/unit/test_migration_deferral_ratchet.py

整个守卫文件被删除，基线 15 → 0，是迁移债务清零的标志性动作。理解它才能理解本 PR 的治理框架。

```
# 该文件随本 PR 整体删除：它把"迁移期间必须跳过的测试"钉死在基线 15 个，
# 职责完成后退役。以下为删除前的完整逻辑，供理解迁移债务管理机制参考。

"""Ratchet guard: config-namespace-migration test deferrals may only decrease."""

from pathlib import Path

# 从 test/registered/unit/ 向上两级得到 test/（测试根目录）
_TEST_ROOT = Path(__file__).resolve().parents[2]

# 在测试文件里搜索的"延期标记"：凡含此文案的文件即视为迁移 deferral
_MARKER = "config-namespace migration"

# 基线 15：与当时被 module-skip 的文件数严格相等
_BASELINE = 15


class TestMigrationDeferralRatchet(CustomTestCase):
    def test_deferred_test_files_match_the_baseline(self):
        # 全量扫描测试根目录，统计仍带 deferral 标记的文件
        deferred = sorted(
            p.relative_to(_TEST_ROOT).as_posix()
            for p in _TEST_ROOT.rglob("*.py")
            if p.name != Path(__file__).name and _MARKER in p.read_text(errors="ignore")
        )
        count = len(deferred)
        # 双向守卫：过多 -> 有新的 skip 悄悄混入；过少 -> 恢复的文件没有同步
        # 下调基线。两条路都会 fail，逼着"恢复一个、锁定一个"。
        if count > _BASELINE:
            self.fail(
                f"deferred (module-skipped) migration tests grew: {count} > "
                f"baseline {_BASELINE}: {deferred}. Do not add new deferrals."
            )
        if count < _BASELINE:
            self.fail(
                f"deferred migration tests shrank: {count} < baseline {_BASELINE}. "
                "Lower the baseline in this file to lock in the recovery."
            )
```

test/registered/unit/mem_cache/test_unified_radix_cache_unittest.py

恢复最复杂的文件：在跳过期间长大了一批 hicache/insert-walk 用例，两个核心用例因 write-through 语义变化需要重构测试场景而非简单恢复 fixture。

```
# test_cache_finished_req_strips_thinking 的恢复模式：
# 原代码直接写 get_server_args().strip_thinking_cache = True（对 namespace
# accessors 不可见），改为通过 serving bag 的作用域 override 发布配置。

# cache_finished_req 读取 get_serving().strip_thinking_cache
with get_serving().override(strip_thinking_cache=True):
    avail_before = allocator.available_size()
    cache.cache_finished_req(
        req,
        is_insert=True,
        kv_len_to_handle=req.effective_kv_committed_len(),
    )
    start_p, end_p = req.effective_kv_committed_len(), req.kv.kv_allocated_len
# override 上下文退出后自动还原，无需手动置 False



# test_shallower_crossing_backs_up_above_backuped_middle 的场景重构：
# 原方案用 insert_host 在"未备份的顶层节点"下方插入备份节点以破坏备份连续性；
# 但 write-through 下 insert_host 会故意丢弃未备份节点下方的 refill
#  （host_insert_dropped），因此改为显式 backup + device eviction 构造等价状态。

self._insert(cache, allocator, req_to_token_pool, [1, 2, 3, 4])
top = next(iter(cache.root_node.children.values()))
# 先插入更深的一层，再显式备份中间的节点
self._insert(cache, allocator, req_to_token_pool, list(range(1, 9)))
middle = next(iter(top.children.values()))
self.assertGreater(_write_backup(cache, middle, write_back=True), 0)
cache.writing_check(write_back=True)
# 把中间的备份节点从设备驱逐，形成"已备份但已被驱逐"的断裂链
cache.evict(EvictParams(num_tokens=4))
self.assertTrue(middle.evicted)
self.assertTrue(middle.backuped)
self.assertFalse(top.backuped)
```

test/registered/unit/models/test_deepseek_v4_shared_expert_fusion.py

最能体现 bag-only 断言模式的恢复样本：从断言 ServerArgs 实例写穿改为断言 get_exec() 命名空间，与 declare_load_time_override 语义对齐。

```
class TestDeepseekV4SharedExpertFusionPolicy(unittest.TestCase):
    """共享专家融合的禁用决策是 load-time 解析：结果落在已发布的配置
    bag 上（declare_load_time_override 只写 bag，ServerArgs 实例保持原样）。"""

    def _make_model(self, n_shared_experts=1):
        return SimpleNamespace(
            config=SimpleNamespace(n_shared_experts=n_shared_experts)
        )
```

```
    def _publish(self, enforce):
        # 通过 runtime context 发布配置：不再跳过模块、不再往 ServerArgs
        # 实例上写属性，而是发布生产代码真正读取的配置命名空间
        override = get_context().override_server_args(
            enforce_shared_experts_fusion=enforce
        )
        override.install() # 激活本次 override
        self.addCleanup(override.restore) # 测试结束自动还原，避免跨用例泄漏

    def test_disables_shared_fusion_without_enforce(self):
        self._publish(enforce=False)
        model = self._make_model()

        DeepseekV4ForCausalLM.determine_num_fused_shared_experts(model)

        self.assertEqual(model.num_fused_shared_experts, 0)
        # bag-only 断言：不再读 server_args 实例字段，改读 get_exec() 的
        # moe 命名空间，与生产代码的读取路径保持一致
        self.assertTrue(get_exec().moe.disable_shared_experts_fusion)

    def test_enables_shared_fusion_when_enforced(self):
        self._publish(enforce=True)
        model = self._make_model()

        DeepseekV4ForCausalLM.determine_num_fused_shared_experts(model)

        self.assertEqual(model.num_fused_shared_experts, 1)
        self.assertFalse(get_exec().moe.disable_shared_experts_fusion)
```

评论区精华

评审主要交锋集中在“恢复后的 fixture 是否真的发布了生产代码所读的配置”：Codex 机器人提出 2 个 P1（MLX resolver 测试缺配置发布与 ps.attn_dp_size、initialize() 桩缺 server_args 占位导致 AttributeError），作者均修复并注明占位字段“整对象求值但从不读取”；作者自审发现 EAGLE setUp 中 cuda_graph_max_bs_decode=8 是无效 seed（生产读 get_exec().graph.cuda_graph_config.decode.max_bs，快速字段不会被 override_server_args 重新折叠），修复为删除无效字段并加注释；另有 2 个关于注释与 bag/raw 读取语义的 nit 类线程，作者逐一核对 #33170 栈上代码后确认或撤回，最终 r3 为零开放问题。

- MLX 回归测试恢复需发布配置并补齐 parallel 状态 (correctness): 作者新增 _published() 辅助，围绕每个 resolve/initialize 调用发布配置；stub 改携带 _ 前缀字段与 ps = SimpleNamespace(attn_dp_size=...)。
- initialize 桩缺 server_args 占位导致 AttributeError (correctness): 作者恢复为空 SimpleNamespace 占位，并注释“整对象作为调用参数求值但从不读取”。
- EAGLE setUp 中 cuda_graph_max_bs_decode 是无效 seed (correctness): 作者删除无效字段，注释说明 dummy-boundary publish 不解析，cuda_graph_config 为 None，

max_running_requests 单独决定预分配尺寸。

- register_to_bootstrap setUp 注释与生产读取语义的核对 (question): 作者核实后撤回：conn.py 在 #33170 已翻转为 bag 读取，无 self.server_args 读取残留，删除死的 MagicMock 块；保留有用的显式 seed（load_balance_method + port=30000）。
- tbo filter_batch 注释中 raw-vs-bag 的说法 (documentation): 作者在 stack 上验证：two_batch_overlap.py 中 attention_backend 走 get_server_args()（per-runner fork 字段保持 raw）、moe_dense_tp_size 走 get_parallel()（#33170 翻转），注释与代码匹配，r2 作者承认误报并撤回。

风险与影响

- 风险：
  1. MLX 平台验证盲区：test_max_running_requests.py 依赖 mlx 安装，CI 的 CPU 套件不会实际执行；_hybrid_stub_for_initialize 中空 server_args 占位是按“整对象求值但从不读取”的假设维护的，若生产代码后续真的读取该字段，表现为 AttributeError（fail 而非误通过），方向安全但依赖 MLX CI 真机覆盖。
  2. radix 用例语义改造：test_shallower_crossing_backs_up_above_backuped_middle 从 insert_host 路径改为显式 backup + evict 构造，若 _write_backup 与真实 write-through 语义有偏差，该用例对“备份连续性破坏”的回归覆盖可能与生产行为不完全等价。
  3. 上游 stack 依赖：多个断言（如 get_parallel().moe_dense_tp_size、get_exec().moe.disable_shared_experts_fusion）依赖 #33170 已合入的 bag 读取语义，栈内后续 PR 若调整命名空间，本批测试需同步跟进。整体为纯测试变更，无生产代码、无运行时行为风险。 - 影响：对 CI：约 120 个此前从未运行的用例重新进入全量套件，覆盖调度（priority scheduling）、radix cache、EAGLE 投机解码、TBO 批重叠、MoE Triton 配置、disaggregation 注册、MLX 并发上限等多模块，等于给迁移后的配置读取路径补上了缺失的回归保护。对团队：ratchet 退役标志迁移债务清零，后续新测试不得再以“迁移期间”为由 module-skip，必须走 override_server_args 发布模式。对用户：无直接运行时影响。 - 风险标记：测试恢复覆盖面大，MLX 平台验证受限，依赖上游 stack 语义，radix 用例语义改造

关联脉络

- PR #33170 Base PR of the config-namespace follow-up stack: 本 PR 的 base，已将 register_to_bootstrap 等生产代码翻转为 bag 读取；本 PR 多个断言与注释的语义前提均来自它。
- PR #33013 config: read resolved config via namespace accessors: config 读取迁移主 PR，本 PR 恢复的 15 个 deferral 测试正是其迁移期间逃逸的回归覆盖。
- PR #33011 config: preserve resolved config across nested publishes + mutation ratchets: 与本 PR 退役的 deferral ratchet 同属迁移护栏体系，后者建立 ratchet 模式，本 PR 展示其退役。
- PR #33172 runtime_context: per-role namespace enforcement behind SGLANG_ROLE_NAMESPACES: 同栈 Part 3（RFC #30696 follow-up），本 PR 恢复的

测试为其提供了配置读取路径的回归保障。

- PR #33179 [CI] Fix runtime context setup in flat logprob tests: 同类“迁移后测试收尾”PR，修复 flat logprob 测试在 runtime context 迁移后的失败，与本 PR 共同构成迁移完成态。