PR #26812 完整报告

sgl-project/sglang

Add a periodic full-radix-tree KV-canary sweep

合并时间：2026-05-31 09:56

原文链接：http://prhub.com.cn/sgl-project/sglang/pull/26812

执行摘要

- 一句话：为 KV-canary 添加定期全 radix 树扫描
- 推荐动作：本 PR 设计清晰，模块划分合理，单元测试覆盖全面，推荐详细了解 radix 树遍历和 SWA 索引映射的实现细节，这些设计模式在类似的可观测性模块中有借鉴意义。建议在合并后关注 long-running 场景下的性能表现。

功能与动机

常规 KV-canary 每步只验证当前请求的 HEAD/TAIL 槽位，但缓存在 radix 树中长期不活跃的 KV 槽位可能发生静默损坏而不被察觉。定期全树扫描填补了这一可观测性盲区，确保所有缓存 KV 的完整性得到周期性验证。

实现拆解

1. 配置入口（server_args.py）：新增 ServerArgs.kv_canary_sweep_interval 参数，通过 CanaryConfig.sweep_interval 传递到下游。
2. Radix 树遍历器（radix_cache_walker.py）：实现 walk_radix_cache_for_canary 函数，递归遍历 radix 树所有非空节点，输出平铺的 slot_indices、positions 和 prev_slot_indices 张量。支持 unlocked_only（跳过锁定节点）和 swa_resident_only（跳过 SWA 墓碑节点）两种过滤模式。
3. 扫描计划构造器（sweep_plan_builder.py）：build_verify_plan_radix_sweep 函数接收遍历结果，构造 VerifyPlan 用于 kernel 执行。处理 SWA 池的索引映射：通过 _swa_translate 将 full 池索引转换为 SWA 池索引，并保留 evicted 槽位（值为 0）作为 padding 哨兵。
4. 扫描编排器（runner/sweep.py）：SweepOrchestrator 管理扫描节奏。maybe_run_sweep 在每步 _post_ops_outside_graph 中被调用，检查自上次扫描是否已超过 sweep_interval 步，若是则遍历所有 buffer_group，为每个 group 构建扫描计划并发射 sweep kernel。attach_radix_cache 由调度器在 radix cache 构建完成后注入。
5. Kernel 启动路径扩展（endpoint.py、kernel_launcher.py）：新增 launch_sweep 和 launch_endpoints_sweep，并引入 SWEEP_K_FULL、SWEEP_V_FULL、SWEEP_K_SWA、SWEEP_V_SWA 等 CanaryLaunchTag 枚举值，确保扫描 kernel 的调用可追踪。
6. 测试（3 个单元测试文件）：test_self_unit_radix_walker.py 测试遍历器的各种树结构和过滤选项；test_self_unit_sweep_plan_builder.py 验证扫描计划的构建和 SWA 翻译；

test_self_unit_runner_sweep.py 验证扫描编排的节奏、kernel 启动和 plan 分配逻辑。test_self_unit_endpoint.py 增加了 sweep kernel 的专用测试用例。

关键文件：

- python/sglang/srt/kv_canary/radix_cache_walker.py（模块 Radix 遍历；类别 source；类型 core-logic；符号 RadixCacheWalkResult, walk_radix_cache_for_canary, _walk_radix_subtree, _node_is_unlocked_for_canary）：新增的 radix 树遍历器，核心功能：递归遍历所有非空节点，输出 slot、position 和 prev_slot 平铺张量，支持锁定过滤和 SWA 墓碑过滤。
- python/sglang/srt/kv_canary/runner/sweep.py（模块 扫描调度；类别 source；类型 core-logic；符号 SweepOrchestrator, init, sweep_passes, attach_radix_cache）：扫描编排器，管理扫描节奏：在每步 forward 后检查是否达到扫描间隔，若满足则触发所有 buffer group 的扫描计划构建和 kernel 发射。
- python/sglang/srt/kv_canary/sweep_plan_builder.py（模块 扫描计划；类别 source；类型 core-logic；符号 build_verify_plan_radix_sweep, _swa_translate）：扫描计划构造器，将 radix 遍历结果转换为 VerifyPlan，并执行 SWA 索引映射。
- test/registered/kv_canary/test_self_unit_radix_walker.py（模块 单元测试；类别 test；类型 test-coverage；符号 TestSelfUnitRadixWalker, setUp, test_single_node_chain_positions_increase, test_child_node_first_slot_prev_is_parent_last）：单元测试覆盖 radix 遍历器的各种树结构、位置计算和过滤选项。
- test/registered/kv_canary/test_self_unit_sweep_plan_builder.py（模块 单元测试；类别 test；类型 test-coverage；符号 TestSelfUnitSweepPlanBuilder, setUp, test_build_verify_plan_radix_sweep, test_radix_held_slot_still_swept）：验证扫描计划构建的正确性，包括空缓存、单链、SWA 翻译和 evicted 处理。
- test/registered/kv_canary/test_self_unit_runner_sweep.py（模块 单元测试；类别 test；类型 test-coverage；符号 _run_one_cycle, TestSelfUnitManagerSweep, test_sweep_every_n_cadence, _spy）：验证扫描编排器的节奏控制、kernel 启动和计划分配逻辑。

关键符号：SweepOrchestrator.maybe_run_sweep, walk_radix_cache_for_canary, build_verify_plan_radix_sweep, _walk_radix_subtree, launch_endpoints_sweep, _swa_translate, attach_radix_cache

关键源码片段

python/sglang/srt/kv_canary/runner/sweep.py

扫描编排器，管理扫描节奏：在每步 forward 后检查是否达到扫描间隔，若满足则触发所有 buffer group 的扫描计划构建和 kernel 发射。

```
from __future__ import annotations
import logging
from collections.abc import Callable
from typing import TYPE_CHECKING, Optional

from sglang.srt.kv_canary.buffer_group import CanaryBufferGroup, PoolKind
```

```
from sglang.srt.kv_canary.config import CanaryConfig
from sglang.srt.kv_canary.endpoint import CanaryEndpoint
from sglang.srt.kv_canary.runner.kernel_launcher import launch_endpoints_sweep
from sglang.srt.kv_canary.state import CanaryDeviceState
from sglang.srt.kv_canary.sweep_plan_builder import build_verify_plan_radix_sweep

if TYPE_CHECKING:
    from sglang.srt.mem_cache.base_prefix_cache import BasePrefixCache

logger = logging.getLogger(__name__)


class SweepOrchestrator:
    """扫描编排器，控制定期全 radix 树扫描的节奏与执行"""

    def __init__(
        self,
        *,
        config: CanaryConfig,
        device_state: CanaryDeviceState,
        buffer_groups: tuple[CanaryBufferGroup, ...],
        endpoints: tuple[CanaryEndpoint, ...],
        swa_window_size: int,
        outer_step_counter_getter: Callable[[], int],
    ) -> None:
        self._config = config
        self._device_state = device_state
        self._buffer_groups = buffer_groups
        self._endpoints = endpoints
        self._swa_window_size = swa_window_size
        self._outer_step_counter_getter = outer_step_counter_getter
        # radix cache 在调度器构建后才可用，通过 attach_radix_cache 注入
        self._radix_cache: Optional["BasePrefixCache"] = None

        self._last_sweep_step: int = -1 # 上次扫描时的 outer step
        self._sweep_passes: int = 0 # 成功扫描次数

    @property
    def sweep_passes(self) -> int:
        """返回已成功扫描的次数"""
        return self._sweep_passes

    def attach_radix_cache(self, radix_cache: "BasePrefixCache") -> None:
        """注入 radix cache 引用，由调度器在树缓存构建完成后调用"""
        self._radix_cache = radix_cache

    def maybe_run_sweep(self) -> None:
        """检查是否需要进行扫描，若是则执行"""
        # 扫描间隔为 0 表示禁用
```

```
if self._config.sweep_interval == 0:
    return
outer_step_counter = self._outer_step_counter_getter()
# 检查是否达到间隔步数（首次无条件执行）
if (
    self._last_sweep_step >= 0
    and outer_step_counter - self._last_sweep_step < self._config.sweep_interval
):
    return
self._last_sweep_step = outer_step_counter

# 尚未注入 radix cache 时跳过
if self._radix_cache is None:
    return

violation_log = self._device_state.violation_log
# 对每个 buffer group 构建扫描计划并发射 kernel
for group in self._buffer_groups:
    # SWA group 使用 SWA window size，非 SWA group 使用 0
    window = self._swa_window_size if group.kind is PoolKind.SWA else 0
    verify_plan = build_verify_plan_radix_sweep(
        radix_cache=self._radix_cache,
        swa_window_size=window,
        full_to_swa_index_mapping=group.swa_index_lut,
    )
    launch_endpoints_sweep(
        endpoints=self._endpoints,
        group=group,
        verify_plan=verify_plan,
        violation_log=violation_log,
    )

self._sweep_passes += 1
logger.info(
    "[canary] sweep succeeded %d times (last_step=%d)",
    self._sweep_passes,
    outer_step_counter,
)
```

评论区精华

本 PR 无实际 review 讨论，仅有一条来自 gemini-code-assist 的配额提醒，未涉及技术内容。

- 暂无高价值评论线程

风险与影响

- 风险：

- 性能：扫描 kernel 每 N 步运行一次，会增加 GPU 执行负载。间隔 N 可配置，建议生产环境中选择较大的 N 或在低峰期运行。
- 内存：遍历 radix 树可能产生较大的 VerifyPlan（与缓存槽位数线性相关），在大规模缓存场景下需关注显存占用峰值。
- 兼容性：新功能默认关闭（sweep_interval=0），不影响现有 KV-canary 行为。开启扫描需同时启用 --kv-canary，否则扫描被静默跳过。
- 耦合风险：SweepOrchestrator 与 radix cache 的具体实现（RadixCache 和 SWARadixCache）直接耦合，若未来引入新的缓存类型需同步修改遍历器。
- 影响：对用户：新增配置项 --kv-canary-sweep-interval，默认关闭。开启后会在指定间隔执行全缓存验证，增加可观测性但引入额外 GPU 开销。对系统：扩展了 KV-canary 的内部架构，新增模块化组件（遍历器、计划构造器、编排器），为后续更多验证模式（如增量扫描）提供了扩展点。对团队：需要通过性能测试确定合适的扫描间隔默认值，并维护新增组件的单元测试。
- 风险标记：新增配置项，性能影响（扫描负载），与 radix cache 耦合，需同步启用 --kv-canary

关联脉络

- PR #26821 Add periodic KV-canary stats logging and kernel-run-counter health check: 提供了健康检查和周期性统计，本 PR 在其基础上扩展了扫描功能
- PR #26820 Add a sliding-window-attention divergence reporter for the KV-canary: 添加了 SWA divergence reporter，本 PR 的扫描计划构建器处理了 SWA 索引映射
- PR #26818 Add token-id verification to the KV-canary: 增加了 token-id 验证，本 PR 的扫描验证与之共享相同的 VerifyPlan 和 kernel 框架
- PR #26817 Add real-data KV verification to the KV-canary: 增加了真实 KV 指纹验证，本 PR 的扫描使用相同的验证 kernel 和 violation 日志机制