PR #35081 完整报告

sgl-project/sglang

Using unified radix tree by default for all case

合并时间：2026-08-21 10:45

原文链接：http://prhub.com.cn/sgl-project/sglang/pull/35081

执行摘要

- 一句话：统一 radix 树设为默认，弃用 SGLANG_ENABLE_UNIFIED_RADIX_TREE
- 推荐动作：值得精读，尤其关注 registry.py 中工厂选择链的收敛以及 unified_radix_cache.py 新增的兼容接口。该 PR 展示了如何将实验性能力转正为默认路径，并保持特例旁路，适合作为 SGLang 缓存架构的入门文档。建议合入后关注纯 full-attention 模型的性能回归数据。

功能与动机

PR body 明确指出：'Unified RadixTree is set as the default tree, which has previously cover SWA/Mamba/HiCache. Now officially includes all cases including Full Only. Deprecate SGLANG_ENABLE_UNIFIED_RADIX_TREE env.' 即统一树已经过多个场景验证，希望收敛默认行为，降低用户配置成本，并为后续缓存能力（如 HiCache、存储后端）提供统一入口。

实现拆解

1. 精简缓存工厂选择链（python/sglang/srt/mem_cache/registry.py）：在 default_radix_cache_factory 中删除 SGLANG_ENABLE_UNIFIED_RADIX_TREE 判断、hybrid SWA/SSM 分支、enable_hierarchical_cache 下的 HiRadixCache 分支以及最终的 RadixCache 兜底，所有常规路径统一收敛到 _create_unified_radix_cache；仅保留纯 SWA 模型、LMCache、FlexKV 等显式特例。这样减少了多套实现并存带来的行为漂移。
2. 补齐 UnifiedRadixCache 兼容接口（python/sglang/srt/mem_cache/unified_radix_cache.py）：新增 query_storage_hit_length 用于同步探测 L3 存储可复用前缀长度，内部对 TP 组执行 all-reduce MIN 并做 page 对齐；新增 is_load_back_event_done 镜像 HiRadixCache 的加载事件状态，供 disagg decode 恢复状态机（DecodeHiCacheTransferMixin）门控使用。
3. 扩展 HostKVCache 聚合转发（python/sglang/srt/mem_cache/memory_pool_host.py）：为多池聚合类增加 get_size_per_token 和 get_split_heads_page_buffer_meta，转发到底层 anchor pool，保证 UnifiedRadixCache 对 host pool 接口的调用可用。
4. 清理启动校验与环境变量（python/sglang/srt/server_args.py、python/sglang/srt/environ.py）：移除 MiMoV2 对 SGLANG_ENABLE_UNIFIED_RADIX_TREE 的强制要求；environ 中相应配置键标记为弃用，避免误导。

5. 同步测试与文档：test_registry.py 将“设置 env 才用 unified”改为“默认就是 unified”，新增 hierarchical + full attention、hybrid 等场景测试，删除 fallback_to_radix_cache 测试；scripted_runtime 增加 resolve_node / to_node_handle 以兼容统一树的 NodeId 句柄；components/README.md 更新说明。

关键文件：

- python/sglang/srt/mem_cache/registry.py（模块 缓存工厂；类别 source；类型 dependency-wiring）：缓存工厂选择链的核心修改，决定了所有模型的默认缓存实现。
- python/sglang/srt/mem_cache/unified_radix_cache.py（模块 统一树；类别 source；类型 core-logic；符号 query_storage_hit_length, is_load_back_event_done）：新增 query_storage_hit_length 和 is_load_back_event_done，补齐统一树对外接口。
- python/sglang/srt/mem_cache/memory_pool_host.py（模块 内存池；类别 source；类型 core-logic；符号 get_size_per_token, get_split_heads_page_buffer_meta）：为 HostKVCache 聚合转发 get_size_per_token 和 get_split_heads_page_buffer_meta，支撑统一树的接口需求。
- python/sglang/srt/server_args.py（模块 启动参数；类别 source；类型 core-logic）：移除 MiMoV2 对 SGLANG_ENABLE_UNIFIED_RADIX_TREE 的强制校验，适配默认切换。
- python/sglang/srt/environ.py（模块 环境配置；类别 source；类型 core-logic）：环境变量配置调整，标记 SGLANG_ENABLE_UNIFIED_RADIX_TREE 弃用。
- test/registered/unit/mem_cache/test_registry.py（模块 测试；类别 test；类型 test-coverage；符号 test_unified_radix_cache_is_the_default, test_unified_radix_cache_when_hierarchical, test_hi_radix_cache_when_hierarchical, test_unified_radix_cache_when_hierarchical）：测试默认行为从 env 驱动改为默认即 unified，并覆盖 hierarchical 与 hybrid 场景。
- python/sglang/test/scripted_runtime/context/radix.py（模块 脚本运行时；类别 test；类型 test-coverage；符号 resolve_node, to_node_handle）：新增 resolve_node / to_node_handle，统一树使用 NodeId 句柄需要转换。
- python/sglang/test/scripted_runtime/req_handle.py（模块 脚本运行时；类别 test；类型 test-coverage）：适配 node 句柄解析方式。
- python/sglang/test/scripted_runtime/context/lock_ref_exhauster.py（模块 脚本运行时；类别 test；类型 test-coverage）：适配统一树的锁引用接口。
- python/sglang/srt/mem_cache/unified_cache/components/README.md（模块 文档；类别 docs；类型 documentation）：文档更新，说明统一树成为默认。

关键符号：default_radix_cache_factory, _create_unified_radix_cache, query_storage_hit_length, is_load_back_event_done, get_size_per_token, get_split_heads_page_buffer_meta, resolve_node, to_node_handle

关键源码片段

python/sglang/srt/mem_cache/registry.py

缓存工厂选择链的核心修改，决定了所有模型的默认缓存实现。

```
def default_radix_cache_factory(ctx: TreeCacheBuildContext) -> BasePrefixCache:
    """Built-in Radix Cache selection chain.

    本 PR 后，常规路径默认统一走 UnifiedRadixCache，仅保留少量显式特例。
    """
    server_args = ctx.server_args
    params = ctx.params

    # 禁用 radix cache 但启用了 host_pool 回退时，仍需 unified 树来支撑
    if (
        ctx.disable_radix_cache
        and get_disagg().disaggregation_decode_retraction_backup == "host_pool"
    ):
        return _create_unified_radix_cache(ctx, server_args, params)

    # chunked prefill + 禁用 radix cache 时使用 ChunkCache（含 SWA 变体）
    if ctx.effective_chunked_prefill_size is not None and ctx.disable_radix_cache:
        if not ctx.is_hybrid_swa:
            from sglang.srt.mem_cache.chunk_cache import ChunkCache
            return ChunkCache(params)
        if ctx.full_tokens_per_layer == 0:
            from sglang.srt.mem_cache.chunk_cache import PureSWAChunkCache
            return PureSWAChunkCache(params)
        from sglang.srt.mem_cache.chunk_cache import SWAChunkCache
        return SWAChunkCache(params)

    # 实验性的 C++ radix tree 仍然保留开关
    if envs.SGLANG_EXPERIMENTAL_CPP_RADIX_TREE.get():
        # lazy import 避免 JIT 开销
        from sglang.srt.mem_cache.radix_cache_cpp import RadixCacheCpp
        logger.info("Using experimental C++ radix tree implementation.")
        return RadixCacheCpp(params=params, server_args=server_args)

    # 纯 SWA 模型（无 full attention 层）继续使用专用缓存
    if ctx.is_hybrid_swa and ctx.full_tokens_per_layer == 0:
        from sglang.srt.mem_cache.pure_swa_radix_cache import PureSWARadixCache
        return PureSWARadixCache(params=params)

    # LMCache / FlexKV 等外部后端仍然优先
    if get_memory().enable_lmcache:
        from sglang.srt.mem_cache.storage.lmcache.lmc_radix_cache import LMCRadixCache
        return LMCRadixCache(
            params=params,
            model_config=ctx.model_config,
            tp_size=ctx.tp_size,
            rank=ctx.tp_rank,
            tp_group=ctx.tp_group,
        )
```

```
    if get_memory().enable_flexkv:
        import os
        from sglang.srt.mem_cache.storage.flexkv import _flexkv_factory
        # 将 CLI 配置转发给 FlexKV 实际读取的环境变量
        if get_memory().flexkv_config_file and not os.environ.get("FLEXKV_CONFIG_PATH"):
            os.environ["FLEXKV_CONFIG_PATH"] = get_memory().flexkv_config_file
        return _flexkv_factory(ctx)

    # 其余所有情况默认使用 UnifiedRadixCache
    return _create_unified_radix_cache(ctx, server_args, params)
```

python/sglang/srt/mem_cache/unified_radix_cache.py

新增 query_storage_hit_length 和 is_load_back_event_done，补齐统一树对外接口。

```
  def query_storage_hit_length(
      self,
      last_host_node_id: NodeId,
      new_input_tokens: list[int],
      last_hash: Optional[str] = None,
      prefix_keys: Optional[list[str]] = None,
  ) -> int:
      """Synchronously probe L3 storage for the reusable prefix length.

      该接口用于替代异步 prefetch 的同步查询，返回可直接复用的前缀长度；
      若未启用存储、速率受限或 key 过短，则返回 0。
      """
      # 未启用存储或 cache controller 缺失时直接返回
      if (
          not self.enable_storage
          or self.cache_controller is None
          or self.cache_controller.prefetch_rate_limited()
      ):
          return 0

      # 构造与 prefetch 相同的 RadixKey，确保查询口径一致
      extra_key, cache_salt = self.tree_core.prefetch_anchor_info(last_host_node_id)
      prefetch_key = RadixKey(
          new_input_tokens,
          extra_key=extra_key,
          is_bigram=self.tree_core.is_eagle,
          cache_salt=cache_salt,
      ).page_aligned(self.page_size)
      if len(prefetch_key) < self.prefetch_threshold:
          return 0

      # 通过 controller 的存储命中查询接口获取命中数
      from sglang.srt.mem_cache.hybrid_cache.hybrid_cache_controller import (
          PrefetchOperation,
      )
```

```
        operation = PrefetchOperation(
            "__storage_hit_query__",
            prefetch_key,
            last_hash,
            prefix_keys,
        )
        _, storage_hit_count = self.cache_controller._storage_hit_query(operation)

        # 多卡环境下取所有 attention 组的最小值，保证各 rank 口径一致
        storage_hit_count_tensor = torch.tensor(storage_hit_count, dtype=torch.int)
        self._all_reduce_attn_groups(
            storage_hit_count_tensor, torch.distributed.ReduceOp.MIN
        )
        storage_hit_count = storage_hit_count_tensor.item()

        # 对齐到 page_size 边界，避免跨页引用不完整数据
        storage_hit_count -= storage_hit_count % self.page_size
        return storage_hit_count
```

评论区精华

PR 没有 review 评论；issue 中作者通过 /rerun-test 触发了几次 CI 重跑，包括 test_disaggregation_decode_radix_cache.py、test_hicache_storage_3fs_backend.py、test_scripted_runtime_core.py 等，最终均通过。其中一次失败的测试被作者判定为与本 PR 无关。主要关注点是确保默认切换后现有关键路径仍然绿。

- 默认切换后的 CI 回归测试 (testing): 重跑全部通过，其中一个失败被判定为与本 PR 无关。

风险与影响

- 风险：
  1. 默认路径变更：所有非特例模型的缓存实现从 RadixCache / HiRadixCache 切换到 UnifiedRadixCache，纯 full-attention 模型此前未经过大规模验证，可能存在前缀命中率或内存分配行为差异。
  2. 新增分布式调用：query_storage_hit_length 中执行 _all_reduce_attn_groups(..., ReduceOp.MIN)，每次调用都会引入一次 TP 组集合通信，若在高频查询路径上使用可能带来延迟开销。
  3. 事件索引风险：is_load_back_event_done 直接访问 layer_done_counter.events[consumer_index]，若 consumer_index 越界或事件未初始化会抛异常，需要调用方保证索引有效。
  4. 环境变量弃用：依赖 SGLANG_ENABLE_UNIFIED_RADIX_TREE 的部署脚本不再生效，需迁移到默认配置或显式 --radix-cache-backend 指定。 - 影响：用户影响：无需再设置环境变量即可获得统一树能力；但旧环境变量被弃用，设置后可能产生告警或未来版本移除。系统影响：默认缓存层变更会影响 prefill 命中率、显存管理以及 disagg 加载流程；统一树支持 Host 侧存储和 HiCache，为后续功能演进（如跨层复用、存储扩展）铺平道路。团队影响：缓存实现分支大幅减少，降低维护成本，但需要保证回归测试覆盖，尤

其是 full-attention、纯 SWA、LMCache/FlexKV 等特例。 - 风险标记：默认行为变更，核心缓存路径切换，弃用环境变量，集合通信开销

关联脉络

- PR #27770 [P/D disagg] Decode-side radix cache for SWA hybrid models (unified radix tree): 该 PR 为 SWA 混合模型启用了统一 radix 树的 decode 侧缓存复用，是统一树在混合模型场景落地的关键一步；本 PR 将其推广为所有场景的默认实现。