PR #27364 完整报告

sgl-project/sglang

[perf] reduce radix cache match overhead by changing the match algorithm

合并时间：2026-06-07 06:40

原文链接：http://prhub.com.cn/sgl-project/sglang/pull/27364

执行摘要

- 一句话：指数搜索优化 RadixCache.match 减少逐元素比较
- 推荐动作：值得精读。指数搜索 + 二分查找的模式通用性强，可推广到其他线性扫描场景。PR 对类型兼容性问题处理果断，测试完备，可放心合入。建议后续关注 million-token 级别的实测数据。

功能与动机

原始实现使用逐元素 zip 循环比较 token_ids，每次迭代都涉及 Python 对象的装箱拆箱开销（见原有 TODO 注释）。随着上下文长度增加（百万 tokens 级别，见 #26943 讨论），这部分开销成为性能瓶颈。PR 通过 slice 级比较将大部分比较下沉到 C 层面。

实现拆解

1. 修改 RadixKey.match 方法（radix_cache.py）：在获取两个 token_ids 后增加类型断言 assert type(t0) is type(t1) 确保 slice 比较正确性；随后采用指数搜索（galloping）定位第一个差异的窗口——每次尝试比较跨度翻倍的 slice，若不等则在该窗口内二分查找精确定位第一个差异的 token 位置（matched_tokens）。
2. 统一返回值计算：根据 is_bigram 和 page_size 调整返回值。删除原先针对 bigram、page_size=1 及 page_size>1 的三套独立循环，全部基于 matched_tokens 计算。
3. 补充单元测试（test_radix_cache_unit.py）：新增 _assert_match 辅助方法，封装 RadixKey 构造与 match 调用；添加 test_match_page_size_1（覆盖全等、部分、无匹配、前缀、空键等边界）、test_match_page_size_gt_1_rounds_down（确认 page_size>1 结果对齐到页边界）、test_match_long_keys_exponential_search（2000 tokens 深度测试 galloping 窗口在不同分歧位置的正确性）、test_match_bigram（验证 bigram 模式下 token 匹配数与 bigram 数的转换）。

关键文件：

- python/sglang/srt/mem_cache/radix_cache.py（模块 缓存层；类别 source；类型 core-logic；符号 RadixKey.match）：核心性能优化：将 match 方法从逐元素循环改为指数搜索 + 二分查找，减少 Python 开销。
- test/registered/unit/mem_cache/test_radix_cache_unit.py（模块 测试；类别 test；类型 test-coverage；符号 _assert_match, test_match_page_size_1, test_match_page_size_gt_1_rounds_down, test_match_long_keys_exponential_search）

：新增全面单元测试，覆盖各种 page_size、bigram 模式以及长键的指数搜索路径。

关键符号：RadixKey.match

关键源码片段

python/sglang/srt/mem_cache/radix_cache.py

核心性能优化：将 match 方法从逐元素循环改为指数搜索 + 二分查找，减少 Python 开销。

```
# python/sglang/srt/mem_cache/radix_cache.py — RadixKey.match 方法（优化后完整实现）

def match(self, other: "RadixKey", page_size: int = 1) -> int:
    """
    计算与 other 共享的逻辑单元前缀长度，结果按 page_size 向下取整。
    使用指数搜索（galloping）快速定位第一个差异 token 的位置。
    """
    self._check_compatible(other)
    t0, t1 = self.token_ids, other.token_ids
    # 类型断言：确保 slice 比较不因类型混用而失效
    assert type(t0) is type(t1), (type(t0), type(t1))
    n = min(len(t0), len(t1))

    # 指数搜索：从位置 0 开始，窗口大小倍增，每次用 C 级 slice 比较
    # 找到第一个差异窗口后，在窗口内二分查找精确定位
    matched_tokens = n
    lo = 0
    step = 1
    while lo < n:
        hi = lo + step if lo + step < n else n
        if t0[lo:hi] != t1[lo:hi]:
            # 在 [lo, hi) 区间内二分查找第一个不等元素
            while hi - lo > 1:
                mid = (lo + hi) // 2
                if t0[lo:mid] == t1[lo:mid]:
                    lo = mid
                else:
                    hi = mid
            matched_tokens = lo
            break
        lo = hi
        step *= 2

    # 根据 bigram 标记调整返回值
    if self.is_bigram:
        matched = max(0, min(matched_tokens - 1, len(self), len(other)))
        return (matched // page_size) * page_size if page_size > 1 else matched

    if page_size == 1:
        return matched_tokens
```

```
    return (matched_tokens // page_size) * page_size
```

test/registered/unit/mem_cache/test_radix_cache_unit.py

新增全面单元测试，覆盖各种 page_size、bigram 模式以及长键的指数搜索路径。

```
# test/registered/unit/mem_cache/test_radix_cache_unit.py — 新增的 match 测试代码

def _assert_match(self, a, b, page_size, expected, is_bigram=False):
    key_a = RadixKey(array("q", a), is_bigram=is_bigram)
    key_b = RadixKey(array("q", b), is_bigram=is_bigram)
    self.assertEqual(key_a.match(key_b, page_size=page_size), expected)

def test_match_page_size_1(self):
    """match() with page_size=1: full, partial, none, prefix, and empty keys."""
    self._assert_match([1, 2, 3, 4], [1, 2, 3, 4], 1, 4) # identical
    self._assert_match([1, 2, 3, 4], [1, 2, 9, 9], 1, 2) # diverge at index 2
    self._assert_match([9, 2, 3], [1, 2, 3], 1, 0) # diverge at index 0
    self._assert_match([1, 2, 3, 4], [1, 2, 3], 1, 3) # other is a prefix
    self._assert_match([], [1, 2], 1, 0) # empty self
    self._assert_match([1, 2], [], 1, 0) # empty other
    self._assert_match([], [], 1, 0) # both empty

def test_match_page_size_gt_1_rounds_down(self):
    """match() with page_size>1 rounds the shared length down to a page."""
    self._assert_match([1, 2, 3, 4, 5, 6, 7, 8], [1, 2, 3, 4, 5, 6, 9, 8], 4, 4)
    self._assert_match([1, 2, 3, 4], [1, 9, 3, 4], 4, 0) # diverge inside first page
    self._assert_match([1, 2, 3, 4, 5, 6, 7, 8], [1, 2, 3, 4, 9, 6, 7, 8], 4, 4)
    self._assert_match([1, 2, 3, 4, 5, 6, 7, 8], [1, 2, 3, 4, 5, 6, 7, 8], 4, 8)
    self._assert_match([1, 2, 3], [1, 2, 3], 4, 0) # shorter than one page

def test_match_long_keys_exponential_search(self):
    """Deep divergences exercise the doubling gallop windows + binary search."""
    base = list(range(2000))
    for div in (1, 2, 63, 64, 65, 127, 128, 511, 512, 513, 1234, 1999):
        b = base[:]
        b[div] = -1
        for page_size in (1, 4, 64):
            with self.subTest(div=div, page_size=page_size):
                self._assert_match(base, b, page_size, (div // page_size) * page_size)
    # Full match of a long key: the gallop must reach the end.
    self._assert_match(base, base[:], 64, (2000 // 64) * 64)

def test_match_bigram(self):
    """is_bigram: L matching raw tokens imply L-1 matching bigrams."""
    self._assert_match([1, 2, 3, 4, 5], [1, 2, 3, 9, 5], 1, 2, is_bigram=True)
    self._assert_match([1, 2, 3, 4, 5], [1, 2, 3, 4, 5], 1, 4, is_bigram=True)
    self._assert_match([1, 2], [1, 2], 1, 1, is_bigram=True)
    # Raw diverge at token 70 -> 69 matching bigrams -> rounded down to 64.
    long_a = list(range(130))
```

```
long_b = list(range(130))
long_b[70] = -1
self._assert_match(long_a, long_b, 64, 64, is_bigram=True)
```

评论区精华

- gemini-code-assist[bot]指出：若 t0 和 t1 类型不同（如 list vs array），t0[lo:hi] != t1[lo:hi] 将始终返回 True，完全破坏优化效果。作者在代码中添加了 assert type(t0) is type(t1) 作为防御措施。
- ispobock要求添加覆盖边界情况和 page_size>1 的单元测试，作者回应“done”，随后在测试文件中实现了相关用例。
- hzh0425批准并建议继续添加更多单元测试。
- 类型兼容性导致 slice 比较失效 (correctness): 作者增加 assert type(t0) is type(t1) 确保类型一致，若断言失败则抛出异常，避免静默错误。
- 需要增加单元测试覆盖边界和 page_size>1 (testing): 作者在测试文件中实现了 test_match_page_size_gt_1_rounds_down 等用例，并回复“done”。

风险与影响

- 风险：类型断言 assert type(t0) is type(t1) 会在类型不匹配时抛出 AssertionError，而原实现可以正常运行（尽管性能较差）。如果某个调用方偶然混用了不同容器类型，将出现新错误。不过生产环境中 token_ids 类型通常一致，风险较低。指数搜索在完全匹配时会遍历到末尾（gallop 到 n 并退出），额外开销约为 2*log2(n) 次 slice 比较，对于长键可忽略，但对于极短键（<4 tokens）可能略慢于原逐元素循环。测试覆盖了短键和边界情况，但未覆盖百万 tokens 量级，超大前缀下的实际性能表现需实测验证。无安全与兼容性风险。
- 影响：影响范围：所有使用 RadixCache 的前缀匹配路径（如 KV cache 重用、前缀命中检测），属于核心调度路径。优化效果在长公共前缀场景下显著，短前缀场景下几乎无退化。配套测试较完善，降低了回归风险。对用户透明，无需任何配置变更。团队应关注后续可能的类型不一致问题，并考虑在调试模式外移除 assert 或降级为 warning。
- 风险标记：类型断言可能引发新错误，核心路径变更，短键场景轻微退化

关联脉络

- PR #26943 [RadixKey] improve slicing for long contexts: Issue 评论中提及，该 PR 也是优化 RadixKey 操作，与本 PR 的目标一致，共同改善 radix cache 在长上下文场景下的性能。