PR #29716 完整报告

sgl-project/sglang

feat(mem_cache): add client-side metadata cache for HiCacheFile storage

合并时间：2026-07-08 09:21

原文链接：http://prhub.com.cn/sgl-project/sglang/pull/29716

执行摘要

- 一句话：为 HiCacheFile 添加客户端元数据缓存
- 推荐动作：推荐阅读此 PR 的设计决策：如何通过回调将缓存失效与 LRU 驱逐器同步，以及 TTL 处理策略。测试覆盖完整，可作为模块级缓存的实现模板。

功能与动机

PR #29716 旨在解决多层级卸载中使用文件后端时的文件系统元数据瓶颈（PR body）。在 Lustre 等共享文件系统上，大量缓存文件导致 os.scandir 和 os.path.exists 调用阻塞事件循环，引入调度延迟。通过客户端内存正缓存直接避免这些文件系统调用。

实现拆解

1. 新增 MetadataCache 类（hicache_storage.py）：线程安全的 TTL 缓存，存储 key 到时间戳映射。contains() 惰性移除过期项，支持 -1 无限 TTL。
2. 环境变量配置（environ.py）：新增 SGLANG_HICACHE_FILE_BACKEND_ENABLE_METADATA_CACHE（默认 False）和 SGLANG_HICACHE_FILE_BACKEND_METADATA_TTL（默认 5.0 秒），可通过 extra_config 覆盖。
3. 集成到 HiCacheFile（hicache_storage.py）：根据配置创建 MetadataCache 实例，启用时调用 _scan_existing_files_to_metadata_cache() 预填充，并将 remove 回调传递给 LRUFileEvictor。
4. LRU 驱逐回调（lru_file_evictor.py）：新增 on_evict 参数，文件驱逐后调用确保缓存与磁盘一致。
5. 配套测试（test_hicache_file_lru_unit.py）：新增 TestMetadataCache 和 TestHiCacheFileMetadataIntegration，35 个测试覆盖基本功能、TTL、集成场景。

关键文件：

- python/sglang/srt/mem_cache/hicache_storage.py（模块 缓存层；类别 source；类型 dependency-wiring；符号 MetadataCache, init, add, remove）：核心变更：实现 MetadataCache 类并集成到 HiCacheFile 中，包括初始化、预扫描和回调传递。
- test/registered/unit/mem_cache/test_hicache_file_lru_unit.py（模块 测试；类别 test；类型 test-coverage；符号 TestMetadataCache, test_metadata_cache_basic, test_metadata_cache_ttl, test_metadata_cache_hard_ttl）：配套测试：新增 MetadataCache 单元测试和 HiCacheFile 集成测试，覆盖基本功能、TTL、启动扫描、写

后回填等。

- python/sglang/srt/mem_cache/storage/file/lru_file_evictor.py（模块 存储层；类别 source；类型 dependency-wiring）：新增 on_evict 回调，在 LRU 驱逐文件后调用 MetadataCache.remove 保持一致性。
- python/sglang/srt/environ.py（模块 配置管理；类别 source；类型 core-logic）：新增两个环境变量控制元数据缓存的启用和 TTL。

关键符号：MetadataCache.init, MetadataCache.add, MetadataCache.remove, MetadataCache.contains, MetadataCache.clear, HiCacheFile._scan_existing_files_to_metadata_cache

关键源码片段

python/sglang/srt/mem_cache/hicache_storage.py

核心变更：实现 MetadataCache 类并集成到 HiCacheFile 中，包括初始化、预扫描和回调传递。

```
# MetadataCache 是一个线程安全的、基于 TTL 的正缓存，
# 用于缓存文件系统中已存在的文件键。
# 它只缓存“存在”的正向信息，避免重复的 os.path.exists 或 os.scandir 调用。
class MetadataCache:
    def __init__(self, ttl_seconds: float):
        self.ttl_seconds = ttl_seconds # TTL 秒数，-1 表示无限
        # cache 映射：key -> 添加时的单调时间戳
        self.cache: dict[str, float] = {}
        self.lock = threading.Lock()

    def add(self, key: str):
        """添加一个键到缓存（如果尚未存在）。"""
        with self.lock:
            if key not in self.cache:
                self.cache[key] = time.monotonic()

    def remove(self, key: str):
        """从缓存中移除一个键（若不存在则忽略）。"""
        with self.lock:
            self.cache.pop(key, None)

    def contains(self, key: str) -> bool:
        """
        检查键是否在缓存中且未过期。
        如果 TTL 为 -1，则永不过期。
        否则，在检查时如果发现过期则惰性移除并返回 False。
        """
        with self.lock:
            if key not in self.cache:
                return False
            if self.ttl_seconds == -1.0:
```

```
            return True
        if time.monotonic() - self.cache[key] > self.ttl_seconds:
            # 过期，惰性移除并返回 False
            del self.cache[key]
            return False
        return True

    def clear(self):
        """清空整个缓存。"""
        with self.lock:
            self.cache.clear()
```

test/registered/unit/mem_cache/test_hicache_file_lru_unit.py

配套测试：新增 MetadataCache 单元测试和 HiCacheFile 集成测试，覆盖基本功能、TTL、启动扫描、写后回填等。

```
class TestMetadataCache(CustomTestCase):
    def test_metadata_cache_basic(self):
        """测试基本的 add / contains / remove 流程。"""
        cache = MetadataCache(ttl_seconds=1.0)
        cache.add("k1")
        self.assertTrue(cache.contains("k1"))
        self.assertFalse(cache.contains("k2"))
        cache.remove("k1")
        self.assertFalse(cache.contains("k1"))

    def test_metadata_cache_ttl(self):
        """测试 TTL 过期：等待 TTL 后键应被移除。"""
        cache = MetadataCache(ttl_seconds=0.1)
        cache.add("k1")
        self.assertTrue(cache.contains("k1"))
        time.sleep(0.2) # 等待超过 TTL
        self.assertFalse(cache.contains("k1"))

    def test_metadata_cache_hard_ttl(self):
        """
        测试硬 TTL：再次 add 不会重置过期时间，
        过期时间基于第一次 add 的时间戳。
        """
        cache = MetadataCache(ttl_seconds=0.3)
        cache.add("k1")
        time.sleep(0.15)
        # 再次添加同一个 key，不应延长 TTL
        cache.add("k1")
        # 从第一次添加至今已 0.35s，因此应过期
        time.sleep(0.2)
        self.assertFalse(cache.contains("k1"))

    def test_metadata_cache_infinite_ttl(self):
```

```
    """测试无限 TTL（ttl_seconds=-1）：永远不会过期。"""
    cache = MetadataCache(ttl_seconds=-1.0)
    cache.add("k1")
    time.sleep(0.3)
    self.assertTrue(cache.contains("k1")) # 仍然存在
```

评论区精华

此 PR 的 review 没有产生实质性讨论；xiezhq-hermann 直接批准。PR body 和代码变更说明清晰，无争议。

- 暂无高价值评论线程

风险与影响

- 风险：
  1. 缓存一致性风险：正缓存可能返回已过期条目；外部删除文件后缓存仍认为存在，直到 TTL 到期。默认 5 秒 TTL 可缓解。
  2. 默认关闭：ENABLE_METADATA_CACHE 默认 False，现有用户无影响。
  3. 线程安全：MetadataCache 使用 Lock，需确保不在事件循环中长时间阻塞。
  4. 回调异常传播：若 on_evict 抛出异常，可能导致驱逐失败；当前调用简单 remove 风险低。
  5. 启动扫描性能：大量文件时初始化遍历目录可能增加启动时间，但不影响 TFFT。 - 影响：对用户：在共享文件系统上启用后可见显著性能提升（P99 TTFT -22%）。新增两个环境变量。对系统：248 行新增，测试覆盖完整。对团队：设计清晰，维护成本低。对其他模块：无影响，可选特性。 - 风险标记：缓存一致性依赖 TTL，默认关闭无回归风险，线程安全但需注意阻塞

关联脉络

- 暂无明显关联 PR