PR #29326 完整报告

sgl-project/sglang

feat(hicache): Add shared memory allocator for host KV cache

合并时间：2026-07-25 12:05

原文链接：http://prhub.com.cn/sgl-project/sglang/pull/29326

执行摘要

- 一句话：为 Host KV Cache 添加共享内存分配器支持
- 推荐动作：该 PR 值得仔细阅读，尤其是 get_allocator_type 的设计模式（将分配器选择与存储后端配置解耦）以及 ShmHostTensorAllocator 的 fd 追踪方法。讨论中关于 memfd_create 与文件回退、madvise 对比 MAP_POPULATE 的技术选型也值得参考。对于需要理解 HiCache 存储层扩展机制的工程师是必读材料。

功能与动机

在分层缓存 (HiCache) 架构中，为了支持零拷贝数据卸载到动态存储后端（如基于 Unix Domain Socket 的卸载进程），需要在共享内存中分配 host 侧的 KV cache 池。通过共享文件描述符，可以避免数据在 SGLang 进程和存储后端之间的 CPU 内存复制，显著降低二级 KV cache 命中时的检索延迟（测试显示减少约 17.7%）。

实现拆解

1. 新增共享内存分配函数：在 mmap_allocator.py 中新增 alloc_shm 函数，通过 memfd_create（优先）或回退到 /dev/shm 文件创建共享内存映射，返回张量及其文件描述符。同时改进 alloc_mmap 使用 madvise(MADV_POPULATE_WRITE) 增强预取。
2. 引入共享内存分配器：在 pool_host/common.py 中新增 ShmHostTensorAllocator 类，继承自 HostTensorAllocator，重写 allocate 方法以调用 alloc_shm，并维护文件描述符列表以便在析构时自动关闭。同时新增 get_allocator_type 函数，从 server_args 中解析出分配器类型（支持 shm 后端或通过 dynamic 配置的 allocator 字段）。
3. 注册 dummy 存储后端：创建 storage/shm/hicache_shm.py 模块，实现 HiCacheShm 类作为 dummy 后端（因为 shm 是本地分配，不需要实际存储传输），并在 backend_factory.py 中注册 shm 选项。
4. 集成到分层缓存构建流程：在 hybrid_pool_assembler.py、hiradix_cache.py 和 decode_kvcache_offload_manager.py 中使用 get_allocator_type 替代直接引用 server_args.hicache_storage_backend，使各缓存池构建时能根据后端动态选择分配器。
5. 补充测试：新增 test_mmap_allocator.py 和扩展 test_mem_pool_host.py，覆盖共享内存分配、unlink 行为、hugepage fallback、设备断言以及 ShmHostTensorAllocator 的 fd 管理。

关键文件：

- python/sglang/srt/mem_cache/pool_host/common.py（模块 Host 内存池；类别 source；类型 dependency-wiring；符号 ShmHostTensorAllocator, init, fd, mm）：核心文件：新增 ShmHostTensorAllocator 和 get_allocator_type 函数，是分配器选择逻辑的心脏。
- python/sglang/srt/mem_cache/storage/mmap/mmap_allocator.py（模块 内存分配器；类别 source；类型 rename-or-move；符号 alloc_shm）：底层分配函数：新增 alloc_shm 函数并优化 alloc_mmap 的预取策略。
- test/registered/unit/mem_cache/test_mmap_allocator.py（模块 测试；类别 test；类型 test-coverage；符号 TestMmapAllocator, test_alloc_mmap, test_alloc_shm, test_shm_host_tensor_allocator）：新增共享内存分配器单元测试，覆盖基本功能、unlink、hugepage fallback 和设备断言。
- python/sglang/srt/mem_cache/storage/shm/hicache_shm.py（模块 HiCache 存储；类别 source；类型 core-logic；符号 HiCacheShm, init, get, batch_get）：新增 dummy 存储后端 HiCacheShm，用于注册 shm 分配器类型以兼容现有后端工厂。
- python/sglang/srt/mem_cache/storage/mmap/__init__.py（模块 内存分配器；类别 source；类型 dependency-wiring）：新模块初始化文件，导出 alloc_mmap 和 alloc_shm 供外部调用。
- python/sglang/srt/mem_cache/hybrid_cache/hybrid_pool_assembler.py（模块 缓存池构建；类别 source；类型 core-logic；符号 _get_allocator_type）：使用 get_allocator_type 重构分配器选择，使各缓存池能根据后端动态切换。

关键符号：alloc_shm, ShmHostTensorAllocator.allocate, get_allocator_type, HiCacheShm.init

关键源码片段

python/sglang/srt/mem_cache/pool_host/common.py

核心文件：新增 ShmHostTensorAllocator 和 get_allocator_type 函数，是分配器选择逻辑的心脏。

```
class ShmHostTensorAllocator(HostTensorAllocator):
    def __init__(self):
        super().__init__()
        self.fds = [] # 记录所有分配的文件描述符，以便析构时关闭
        self.mms = []

    @property
    def fd(self):
        # 返回第一个分配的文件描述符
        return self.fds[0] if self.fds else None

    @property
    def mm(self):
        return self.mms[0] if self.mms else None

    def allocate(self, dims: tuple, dtype: torch.dtype, device: str) -> torch.Tensor:
        # 确保设备为 CPU，共享内存只支持 CPU 分配
```

```
        assert device == "cpu", f"ShmHostTensorAllocator only supports CPU allocations; got
        device={device!r}"
        self.dtype = dtype
        self.dims = dims
        from sglang.srt.mem_cache.storage.mmap import alloc_shm
        tensor, fd, mm = alloc_shm(dims, dtype) # 分配共享内存并返回 fd 和 mmap 对象
        self.fds.append(fd)
        self.mms.append(mm)
        return tensor

    def __del__(self):
        # 析构时关闭所有记录的文件描述符，避免泄漏
        for fd in getattr(self, "fds", []):
            if fd is not None:
                try:
                    os.close(fd)
                except OSError:
                    pass
        self.fds = []

def get_allocator_type(server_args) -> str:
    # 从 server_args 中推导出分配器类型字符串
    backend = getattr(server_args, "hicache_storage_backend", None)
    if backend == "shm":
        return "shm"
    if backend == "dynamic":
        extra_config_str = getattr(server_args, "hicache_storage_backend_extra_config", None)
        if extra_config_str:
            try:
                config = json.loads(extra_config_str)
                if config.get("allocator") == "shm":
                    return "shm"
            except Exception:
                pass
    return backend or "default"
```

python/sglang/srt/mem_cache/storage/mmap/mmap_allocator.py

底层分配函数：新增 alloc_shm 函数并优化 alloc_mmap 的预取策略。

```
def alloc_shm(dims: tuple, dtype: torch.dtype) -> tuple[torch.Tensor, int, mmap.mmap]:
    # 通过共享内存分配张量，返回 (tensor, fd, mm) 元组
    n_bytes = math.prod(dims) * torch.empty([], dtype=dtype).element_size()
    hugepage_size = (envs.SGLANG_HUGEPAGE_SIZE.get() or "").strip().upper()
    if hugepage_size != "":
        logger.warning("Hugepages are not supported with SHM allocator. Falling back to plain
        page-size mmap.")

    page_size = mmap.PAGESIZE
    alloc_bytes = math.ceil(n_bytes / page_size) * page_size
```

```
    # 优先使用 memfd_create（内核 3.17+）
    fd = None
    try:
        fd = os.memfd_create(
            f"sglang_host_pool_{uuid.uuid4().hex}",
            flags=getattr(os, "MFD_CLOEXEC", 1),
        )
    except (AttributeError, OSError):
        # 回退到 /dev/shm 中的临时文件
        shm_path = f"/dev/shm/sglang_host_pool_{uuid.uuid4().hex}.mmap"
        try:
            fd = os.open(shm_path, os.O_CREAT | os.O_RDWR | os.O_TRUNC, 0o600)
            os.unlink(shm_path) # 立即 unlink，OS 回收内存
        except Exception as e:
            raise OSError(f"Failed to create shm file: {e}")

    os.ftruncate(fd, alloc_bytes)
    mm = mmap.mmap(fd, alloc_bytes, flags=mmap.MAP_SHARED | _MAP_POPULATE, prot=
    mmap.PROT_READ | mmap.PROT_WRITE)
    # MADV_POPULATE_WRITE 改进了预取行为（内核 5.14+）
    try:
        mm.madvise(_MADV_POPULATE_WRITE)
    except OSError:
        # 旧内核回退到 MAP_POPULATE
        pass

    tensor = torch.frombuffer(mm, dtype=dtype, count=math.prod(dims)).reshape(dims)
    return tensor, fd, mm
```

评论区精华

- 设计争议：hzh0425 最初反对新增 --host-kvcache-allocator 参数，要求改为复用 --hicache-storage-backend 的 shm 后端来驱动分配器选择。开发者采纳该建议，实现了 get_allocator_type 推导逻辑。
- 实现细节：guymguym 提出使用 memfd_create 替代手动创建 /dev/shm 文件，以及使用 madvise(MADV_POPULATE_WRITE) 替代 MAP_POPULATE，从而提高跨进程共享和预取效率。开发者均采纳。
- 模块划分：hzh0425 建议将 HiCacheShm 从 hicache_storage.py 分离到独立的 storage/shm/hicache_shm.py 文件中，以保持模块结构清晰。
- 兼容性确认：hzh0425 询问 ShmHostTensorAllocator 是否兼容 hybrid pool（如 Mamba/SWA 等），开发者确认每个池拥有独立的分配器实例，并通过列表跟踪多个 fd，兼容性没问题。
  - 将 shm 作为 storage backend 而非新增参数 (design): 采纳建议，将 shm 作为 --hicache-storage-backend 的一个选项，并通过 get_allocator_type 动态推导分配器类型。

- 使用 memfd_create 替代 /dev/shm 文件 (performance): 采用 memfd_create 作为主要路径，并在不支持时回退到 /dev/shm。
- 使用 madvise(MADV_POPULATE_WRITE) 改进预取 (performance): 在 alloc_mmap 和 alloc_shm 中均已添加 madvise 调用，并在不支持时静默回退。
- 将 HiCacheShm 分离到独立文件 (design): 移动 HiCacheShm 到 storage/shm/hicache_shm.py。
- ShmHostTensorAllocator 与 hybrid pool 的兼容性 (correctness): 作者确认每个池拥有单独的 allocator 实例，并通过 fds 列表管理多个 fd，兼容所有池类型。

风险与影响

- 风险：
  1. 共享内存泄漏：/dev/shm 中的文件在创建后立即 unlink，由 OS 自动回收；但若 fd 在 ShmHostTensorAllocator 析构前丢失，可能泄漏 fd（已通过 __del__清理）。低风险。
  2. Hugepage 回退静默：对 hugepage 请求，alloc_shm 会发出警告并回退到普通页面，可能影响性能（测试已覆盖）。
  3. 与 Hybrid Pool 集成验证不充分：虽然声称兼容，但多池场景（如 Full + Indexer）下 fd 管理可能有未释放风险，需要更多端到端测试。
  4. 旧内核兼容性：MADV_POPULATE_WRITE 在 <5.14 内核上回退到 MAP_POPULATE，不影响功能但预取弱化。 - 影响：用户可通过 --hicache-storage-backend shm 启用共享内存分配，但依赖 Linux memfd_create（内核 3.17+）或 /dev/shm 存在。零拷贝 offloading 将基于此特性实现，是该功能链路上的基础模块。代码结构重构（mmap_allocator 搬迁至 storage/mmap/，新增 storage/shm/），团队后续 offloading PR 需基于此接口。 - 风险标记：共享内存泄漏，hugepage 静默回退，混合池兼容需验证，旧内核预取降级

关联脉络

- 暂无明显关联 PR