

PR #54014 完整报告

vllm-project/vllm

[Bugfix][KV Offload] Check cgroup memory before SHM allocation

合并时间: 2026-09-16 14:17

原文链接: <http://prhub.com.cn/vllm-project/vllm/pull/54014>

执行摘要

- 一句话: SHM 分配前新增 cgroup 内存预检, 余量不足只告警不阻断
- 推荐动作: 值得精读, 但只需读三处: vllm/utils/cpu_resource_utils.py 的读取器拆分与 check_cgroup_memory_available()、shm_broadcast.check_shm_free_space() 的控制流改写, 以及 Etelis 的实测评论。最值得借鉴的设计决策有两个: 一是用「缓存静态 limit、实时读 usage」同时满足稳定性与时效性, 避免了简单移除 @cache 带来的重复 sysfs 读取; 二是在度量语义不精确 (memory.current 含可回收 cache) 时选择「诊断而非门禁」, 用实验数据推翻了自己最初 raise 的设计。若你负责容器化部署, 建议同时核对全仓是否还有按元组解包 get_cgroup_memory_limit() 的调用点, 并评估低余量 warning 在自家负载下的噪音水平。

功能与动机

PR body 明确指出问题根因: “/dev/shm free space and cgroup memory are separate constraints. An allocation can fit the SHM filesystem while the container has insufficient cgroup headroom. Populating the pages can then trigger a cgroup OOM kill without an application-level diagnostic.” 即 CPU KV offload 在 /dev/shm 上 ftruncate 并 MADV_POPULATE_WRITE 时, 文件系统空间充足但容器 cgroup 配额不足, 会直接触发 cgroup OOM kill, 现场没有任何应用层线索。作者同时声明该 PR 不重复 #55247 (后者处理 CPU KV-cache 自动 sizing 与 NUMA/ 非 NUMA 的 cgroup 钳制), 本 PR 只聚焦 SHM 分配预检诊断与 CPU KV offload 共享内存 sizing。

实现拆解

1. 拆分 cgroup 读取器 (vllm/utils/cpu_resource_utils.py) : get_cgroup_memory_limit() 由返回 tuple[int | None, int | None] 改为只返回 int | None, 保留 @cache (容器生命周期内 limit 静态); 新增 get_cgroup_memory_usage() 专门读 memory.current / memory.usage_in_bytes, 刻意不加缓存, 保证每次预检拿到实时水位; 两者都保留 cgroup v2 优先、v1 回退, 以及 max 与 1 << 62 哨兵值的判无限逻辑。
2. 新增预检函数: check_cgroup_memory_available(required_bytes, allocation_name) 计算 cgroup_available = max(0, limit - usage) 与 remaining_bytes, 余量充足走 logger.debug、不足走 logger.warning; 任一读数缺失 (非 Linux、文件不存在) 直接 return 跳过。函数不抛异常, 因为 cgroup usage 含可回收 file cache, 硬失败会误伤实际可成功的分配。

3. 接线 SHM 预检入口 (vllm/distributed/device_communicators/shm_broadcast.py) :
check_shm_free_space() 增加仅关键字参数 allocation_name (默认 "shared-memory allocation") , 控制流由早返回改为嵌套判断, 使 /dev/shm 不存在时也仍执行 cgroup 检查; 文件系统不足时抛出的 RuntimeError 文案带上 allocation 名称。
check_shm_free_space 的语义从纯文件系统校验扩展为「文件系统抛错 + cgroup 记录日志」。
4. 调用点透传分配名: vllm/v1/kv_offload/cpu/shared_offload_region.py 的 creator 路径传入 allocation_name="CPU KV offload shared region in /dev/shm";
vllm/model_executor/model_loader/weight_utils.py 的 _get_available_ram_bytes() 按新签名分两次调用 get_cgroup_memory_limit() 与 get_cgroup_memory_usage(), 间接修复了此前 usage 与 limit 一起被缓存导致的陈旧读数。
5. 测试与配套: 新增 tests/utils/_test_cpu_resource_utils.py (189 行) , 用 _stub_files/fake_open 以 StringIO 或 OSError 桩化 builtins.open, 配合 autouse 的 _clear_cgroup_cache fixture 覆盖 v2、v1、v2 unlimited 回退 v1、v1 哨兵、无文件、非 Linux、limit 被缓存、usage 不被缓存、warning/debug 分支、快照缺失跳过等 11 个用例; tests/distributed/test_shm_broadcast.py 保留一个委派测试
test_check_shm_free_space_checks_cgroup 断言参数透传;
tests/v1/kv_offload/cpu/test_shared_offload_region.py 把原先的 lambda 抛错 mock 改为 MagicMock, 并断言 allocation_name 被正确传入。无 schema、部署或模型输出侧改动。

关键文件:

- vllm/utils/cpu_resource_utils.py (模块 资源探测; 类别 source; 类型 core-logic; 符号 get_cgroup_memory_limit, get_cgroup_memory_usage, check_cgroup_memory_available, get_memory_node_info) : 本 PR 的核心: 把 cgroup limit/usage 读取拆成缓存与不缓存两条路径, 并新增不抛异常的预检函数, 是后续所有接线的能力来源。
- tests/utils/_test_cpu_resource_utils.py (模块 资源探测; 类别 test; 类型 test-coverage; 符号 _clear_cgroup_cache, _stub_files, fake_open, test_cgroup_v2_limit_and_usage) : 本 PR 新增的聚焦测试文件, 用 sysfs 桩覆盖 v1/v2 分支、缓存语义与日志级别, 是验证「limit 缓存、usage 不缓存」契约的唯一入口。
- vllm/distributed/device_communicators/shm_broadcast.py (模块 共享内存; 类别 source; 类型 dependency-wiring; 符号 check_shm_free_space, SHM_PATH) : SHM 预检的统一入口: check_shm_free_space 新增 allocation_name 参数并改为嵌套控制流, 使 cgroup 检查在 /dev/shm 缺失时也生效。
- vllm/model_executor/model_loader/weight_utils.py (模块 权重加载; 类别 source; 类型 data-contract; 符号 _get_available_ram_bytes) : 非 KV offload 侧的既有调用点, 按新签名改用 limit/usage 两次读取, 顺带修复了 usage 与 limit 一起被缓存导致的陈旧估算。
- vllm/v1/kv_offload/cpu/shared_offload_region.py (模块 卸载区域; 类别 source; 类型 core-logic; 符号 SharedOffloadRegion.init) : 本次预检的真实目标场景: creator 路径在 ftruncate 与 populate 之前调用 check_shm_free_space, 并透传可读的分配名用于诊断。

- tests/distributed/test_shm_broadcast.py (模块 共享内存; 类别 test; 类型 test-coverage; 符号 test_check_shm_free_space_checks_cgroup, test_check_shm_free_space_passes_when_sufficient) : 只保留委派测试 test_check_shm_free_space_checks_cgroup, 验证 allocation_name 被原样传给 cgroup 预检, 其余聚焦用例按 reviewer 要求迁出。
- tests/v1/kv_offload/cpu/test_shared_offload_region.py (模块 卸载区域; 类别 test; 类型 test-coverage; 符号 test_insufficient_space_raises_clear_error) : 把原先一次性 lambda 抛错 mock 换成 MagicMock, 并断言 allocation_name 被透传, 保证诊断名称不会被静默改掉。

关键符号: get_cgroup_memory_limit, get_cgroup_memory_usage, check_cgroup_memory_available, check_shm_free_space, get_memory_node_info, _get_available_ram_bytes, SharedOffloadRegion.init, _stub_files

关键源码片段

vllm/utils/cpu_resource_utils.py

本 PR 的核心: 把 cgroup limit/usage 读取拆成缓存与不缓存两条路径, 并新增不抛异常的预检函数, 是后续所有接线的能力来源。

cgroup 读取器拆分与预检函数 (vllm/utils/cpu_resource_utils.py)

```
# 静态的 cgroup limit 缓存一次即可: 容器生命周期内 limit 基本不变,
# 避免每次 SHM 分配都重复读取 sysfs。
@cache
def get_cgroup_memory_limit() -> int | None:
    """Return the cgroup memory limit in bytes, or None."""
    if sys.platform != "linux":
        return None

    # cgroup v2 统一层级: memory.max 为 "max" 时 _read_int_file 返回 None,
    # 继续向下尝试 v1, 兼容 mixed 场景。
    v2_limit = _read_int_file("/sys/fs/cgroup/memory.max")
    if v2_limit is not None:
        return v2_limit

    # cgroup v1: 无限制时内核写入接近 PAGE_COUNTER_MAX 的哨兵值,
    # 用 1 << 62 作为阈值把「无限」与真实 limit 区分开。
    v1_limit = _read_int_file("/sys/fs/cgroup/memory/memory.limit_in_bytes")
    if v1_limit is not None:
        if v1_limit >= (1 << 62):
            return None
        return v1_limit

    return None
```

```
def get_cgroup_memory_usage() -> int | None:
```

```
"""Return the current cgroup memory usage in bytes, or None.
```

刻意不加 @cache: memory.current 在进程运行期间持续变化,
缓存会让 preflight 读到过期水位 (Review 中 orozery 明确要求拆分)。

```
"""
```

```
if sys.platform != "linux":
```

```
    return None
```

```
# cgroup v2 统一层级: 先确认 memory.max 可读 (即处于 v2), 再读 current.
```

```
if _read_int_file("/sys/fs/cgroup/memory.max") is not None:
```

```
    return _read_int_file("/sys/fs/cgroup/memory.current")
```

```
# cgroup v1: 哨兵值刻意不算有效 limit, 避免给出错误余量。
```

```
v1_limit = _read_int_file("/sys/fs/cgroup/memory/memory.limit_in_bytes")
```

```
if v1_limit is not None and v1_limit < (1 << 62):
```

```
    return _read_int_file("/sys/fs/cgroup/memory/memory.usage_in_bytes")
```

```
return None
```

```
def check_cgroup_memory_available(required_bytes: int, allocation_name: str) -> None:
```

```
    """Log cgroup memory headroom for an upcoming allocation.
```

只记录诊断信息, 不拒绝分配: memory.current 含可回收的 file cache,

硬失败会误伤实际能成功的分配 (Etelis 在 1 GiB cgroup 中的实测结论)。

```
"""
```

```
cgroup_limit = get_cgroup_memory_limit()
```

```
cgroup_usage = get_cgroup_memory_usage()
```

```
# 任一读数缺失 (非 Linux、文件不存在) 时静默跳过, 不产生误导性结论。
```

```
if cgroup_limit is None or cgroup_usage is None:
```

```
    return
```

```
cgroup_available = max(0, cgroup_limit - cgroup_usage)
```

```
mib = 1 << 20
```

```
remaining_bytes = cgroup_available - required_bytes
```

```
# 余量充足走 debug 避免正常启动刷屏; 不足时升级为 warning 便于排障。
```

```
log_fn = logger.debug if remaining_bytes >= 0 else logger.warning
```

```
status = (
```

```
    "current headroom meets the requested allocation"
```

```
    if remaining_bytes >= 0
```

```
    else "current headroom is below the requested allocation; allocation "
```

```
    "will still be attempted because cgroup usage may be reclaimable"
```

```
)
```

```
log_fn(
```

```
    "Cgroup memory preflight for %s: %.0f MiB required, %.0f MiB current "
```

```
    "usage, %.0f MiB available under %.0f MiB limit, %.0f MiB remaining "
```

```
    "after allocation based on current usage; %s.",
```

```
    allocation_name,
```

```
    required_bytes / mib,
```

```

        cgroup_usage / mib,
        cgroup_available / mib,
        cgroup_limit / mib,
        remaining_bytes / mib,
        status,
    )

```

tests/utils_/test_cpu_resource_utils.py

本 PR 新增的聚焦测试文件，用 sysfs 桩覆盖 v1/v2 分支、缓存语义与日志级别，是验证「limit 缓存、usage 不缓存」契约的唯一入口。

用 sysfs 桩覆盖 v1/v2 与缓存语义 (tests/utils_/test_cpu_resource_utils.py)

```

@pytest.fixture(autouse=True)
def _clear_cgroup_cache():
    # limit 走 @cache，跨用例必须显式清空，否则会读到上一个用例的 sysfs 桩。
    cru.get_cgroup_memory_limit.cache_clear()
    yield
    cru.get_cgroup_memory_limit.cache_clear()

def _stub_files(monkeypatch, files: dict):
    """Stub ``open()`` for a fixed set of paths; ``None`` -> OSError."""
    real_open = open

    def fake_open(path, *args, **kwargs):
        if path in files:
            content = files[path]
            # 值为 None 表示「文件不存在」，用于验证 v2 -> v1 回退与跳过分支。
            if content is None:
                raise OSError(f"no such file: {path}")
            return StringIO(content)
        # 其余路径走真实实现，避免影响测试进程的其它 I/O。
        return real_open(path, *args, **kwargs)

    monkeypatch.setattr("builtins.open", fake_open)

def test_cgroup_usage_is_read_without_cache(monkeypatch):
    """usage 必须每次重新读文件，否则预检水位会长期过期。"""
    monkeypatch.setattr(sys, "platform", "linux")
    files = {
        _V2_LIMIT_PATH: f"{1 * GiB_bytes}\n",
        _V2_USAGE_PATH: f"{512 << 20}\n",
    }
    _stub_files(monkeypatch, files)

    assert cru.get_cgroup_memory_usage() == 512 << 20
    files[_V2_USAGE_PATH] = f"{768 << 20}\n"

```

```
assert cru.get_cgroup_memory_usage() == 768 << 20 # 同一进程内即时反映变化
```

```
def test_check_cgroup_memory_available_warns_on_low_headroom(monkeypatch):
    monkeypatch.setattr(cru, "get_cgroup_memory_limit", lambda: 1 << 30)
    monkeypatch.setattr(cru, "get_cgroup_memory_usage", lambda: 512 << 20)

    with (
        mock.patch.object(cru.logger, "debug") as log_debug,
        mock.patch.object(cru.logger, "warning") as log_warning,
    ):
        # 可用 512 MiB, 请求 600 MiB: 应触发 warning 且不抛异常。
        cru.check_cgroup_memory_available(600 << 20, "mmap")

    log_debug.assert_not_called()
    log_warning.assert_called_once()
    assert "current headroom is below" in log_warning.call_args.args[-1]
```

vllm/distributed/device_communicators/shm_broadcast.py

SHM 预检的统一入口: check_shm_free_space 新增 allocation_name 参数并改为嵌套控制流, 使 cgroup 检查在 /dev/shm 缺失时也生效。

SHM 预检入口的接线 (vllm/distributed/device_communicators/shm_broadcast.py)

```
SHM_PATH = "/dev/shm"
```

```
def check_shm_free_space(
    required_bytes: int,
    shm_path: str = SHM_PATH,
    *,
    allocation_name: str = "shared-memory allocation",
) -> None:
    """Raise if SHM cannot fit a shared segment and log cgroup headroom.

    Args:
        required_bytes: Size of the shared-memory segment to be created.
        shm_path: Mount point backing POSIX shared memory; its filesystem
            check is skipped if absent.
        allocation_name: Human-readable name used in errors and logs.

    Raises:
        RuntimeError: If the SHM filesystem has insufficient space.
    """
    # 文件系统维度: 容量不足直接抛错, 保持原有硬失败语义,
    # 只是错误文案里带上了 allocation 名称, 便于定位是谁在申请。
    if os.path.isdir(shm_path):
        free_bytes = shutil.disk_usage(shm_path).free
```

```

if required_bytes > free_bytes:
    mib = 1 << 20
    raise RuntimeError(
        f"Insufficient space in {shm_path} for {allocation_name}: "
        f"{required_bytes / mib:.0f} MiB required, "
        f"{free_bytes / mib:.0f} MiB free. Increase {shm_path} "
        "(e.g. --shm-size or --ipc=host)."
    )

# cgroup 维度：/dev/shm 容量与 cgroup 内存是两套独立约束，
# 文件系统放得下时仍可能因 MADV_POPULATE_WRITE 触顶导致 cgroup OOM kill。
# 注意这里刻意放在 isdir 判断之外：shm_path 不存在时仍要做预检。
check_cgroup_memory_available(
    required_bytes,
    allocation_name,
)

```

评论区精华

核心交锋是「预检应该 raise 还是只 warn」。Etelis 以 CHANGES_REQUESTED 提交了实测证据：在 1 GiB cgroup 中，早期版本（commit [0524c39](#)）以 307 MiB 余量拒绝了 512 MiB 的 SHM 分配，但 [MADV_POPULATE_WRITE](#) 在回收 208 MiB cache 后成功完成且未 OOM，因此主张“I'd warn here until we account for reclaimable memory, and adjust the existing docstring/test.”结论被采纳：最终实现改为 warning + 继续分配，docstring 与测试同步改写。

第二条主线来自 orozery 的审查：一是要求把聚焦测试挪到 [tests/utils_/test_cpu_resource_utils.py](#)（即 #55247 引入、符合 [tests/utils_/](#) 约定的文件）并只保留委派测试，理由是降低合并冲突；二是针对 [@cache](#) 移除，指出“keep @cache on the static limit and add a separate uncached reader for usage/available — that fixes the staleness without touching #55247's fixture”，这正是最终落地的拆分方案；三是把成功路径日志由 [logger.info](#) 降为 [logger.debug](#)，避免正常启动刷屏。

CodeRabbit 曾以“Critical”标记 warning 分支未抛 [RuntimeError](#)、与 docstring 契约不符，该结论在 warn-only 设计确定后即失效（属于旧版本的过时告警）。PR 评论区还有作者对 #54619 的表态：“I think this feature is helpful in production, and should not be blocked by #54619”，以及 orozery 推动 CI 运行（[/ci run](#)、[/ci retry](#)）的过程。

- 预检应 raise 还是只 warn：memory.current 含可回收 file cache (correctness)：采纳：check_cgroup_memory_available 只在低余量时打 warning 并继续分配，docstring 与测试同步改为 warn-only 语义；status 文案显式说明「allocation will still be attempted because cgroup usage may be reclaimable」。
- limit 保留 @cache、usage 另设无缓存读取器 (design)：采纳：get_cgroup_memory_limit 恢复 @cache 且只返回 limit；新增 get_cgroup_memory_usage 每次读取，返回类型由 tuple 改为 int | None，两个调用点同步适配。
- 聚焦测试落位与合并冲突控制 (testing)：采纳：新增 tests/utils_/test_cpu_resource_utils.py 承载 cgroup 读取与预检用例，shm 测试文件只留委派测试。

- 成功路径日志级别由 info 降为 debug (style): 采纳: 成功路径改为 debug, warning 仅保留给余量不足场景, 降低正常启动日志噪音。
- 警告分支未抛 RuntimeError (CodeRabbit Critical) (question): 该告警在 warn-only 设计确定后失效: docstring 与测试已改写为「低余量只记录日志」, 异常契约只保留文件系统容量维度; 后续 CodeRabbit 复审未再产生 actionable comment。

风险与影响

• 风险:

1. 公共工具函数签名破坏性变更: `get_cgroup_memory_limit()` 由返回二元组改为返回 `int | None`。本 PR 已同步 `get_memory_node_info()` 与 `weight_utils._get_available_ram_bytes()` 两个调用点, 但该函数位于 `vllm/utils`, 若仓库其它位置或下游集成仍按元组解包, 会静默拿到单个整数并在解包处报错; 需在合并后全仓确认调用点。
2. 只告警不拦截的语义落差: 当余量不足且 file cache 不可回收时, cgroup OOM kill 依然会发生, `check_shm_free_space()` 的 RuntimeError 契约只在文件系统容量维度成立, 调用方无法通过异常分支感知 cgroup 维度问题; 预检的价值限定在「事后可诊断」。
3. 读数一致性与额外开销: `get_cgroup_memory_usage()` 每次调用都重新读 `memory.max` 来判断 hierarchy, 比直接复用缓存 limit 多一次 sysfs 读取; 同时 limit 走缓存而 usage 实时读取, 理论上两者可能来自不同时刻的层级判断 (极端场景下才可见)。
4. 行为变化波及 checkpoint 加载路径: `_get_available_ram_bytes()` 现在每次读取实时 usage, 而此前 usage 与 limit 一起被 @cache 冻结, 长流程中 checkpoint 大小与可用 RAM 的比值会随 page cache 波动, 更准确但行为不再稳定可复现。
5. 告警噪音: 在 page cache 占比高的高负载容器中, 低余量 warning 可能频繁出现, 运维容易钝化对该日志的敏感度——这正是 Etelis 实测揭示的 `memory.current` 语义带来的副作用。
6. 测试覆盖边界: 测试通过 `monkeypatch.setattr("builtins.open", fake_open)` 全局替换 open, 属于单元级桩化, 未覆盖真实 cgroup + MADV_POPULATE_WRITE 的端到端 OOM 场景; PR body 亦声明“No model evaluation is applicable”。- 影响: 用户侧: 以容器 /Pod 形式设置 cgroup 内存上限、并使用 CPU KV offload (`vllm/v1/kv_offload/cpu/shared_offload_region.py`) 或多进程 SHM 广播的部署, 启动 / 首次分配时可获得包含 required、usage、available、limit、remaining 五元快照的日志, OOM kill 从「无解释」变为「可定位」; 正常部署多一条 debug 日志, 无功能变化。系统侧: `vllm/utils/cpu_resource_utils.py` 作为 CPU/ 容器资源探测的公共模块, 其 limit/usage 读取被 `check_shm_free_space`、CPU KV offload 与权重加载 RAM 估算三条路径复用, 改动面覆盖 distributed、model_executor、v1/kv_offload 三个子系统, 但均为小幅接线, 无性能关键路径改动。团队侧: 本 PR 与 #55247 共用测试文件与 cgroup 语义, 需在后续改动中保持 limit 缓存、usage 不缓存的约定; 测试文件落位约定 (`tests/utils/_/`) 也被明确了一次。整体影响等级为中低, 属可观测性增强而非行为修复。- 风险标记: 公共工具 API 签名变更, 只告警不阻断分配, 缓存语义变更, 缺少端到端 OOM 场景覆盖

关联脉络

- PR #55247 CPU KV-cache auto-sizing and cgroup clamping: PR body 明确声明两者不重复: 该 PR 处理 CPU KV-cache 自动 sizing 与 NUMA/ 非 NUMA 的 cgroup 钳制, 本 PR 聚焦 SHM 预检诊断; review 中 orozery 要求复用其引入的 tests/utils_/test_cpu_resource_utils.py 文件与 fixture, 且本 PR 的 @cache 调整刻意避免破坏该 PR 的测试夹具。
- PR #54619 被作者提及的前置 / 并行 PR: 作者在 Issue 评论中表示“I think this feature is helpful in production, and should not be blocked by #54619”, 说明本 PR 的落地节奏曾与该 PR 存在依赖或排期上的讨论。