PR #35707 完整报告

sgl-project/sglang

[diffusion] read the cgroup this process is actually in

合并时间：2026-08-21 08:48

原文链接：http://prhub.com.cn/sgl-project/sglang/pull/35707

执行摘要

- 一句话：修复嵌套 cgroup 下内存上限误读，避免 OOM
- 推荐动作：值得精读。核心看点：一是容器内外 cgroup 路径不对齐问题的"渐进缩短后缀"解法，简洁、无依赖且语义清晰；二是"取最紧 cap + 对应 usage"的选择，正确反映了 cgroup 层级继承；三是测试用 tmp_path 构造嵌套 cgroup 树的方式，值得作为同类 sysfs 探测逻辑的测试模板。建议后续补充 v2 嵌套场景用例，并关注混合 v1/v2 挂载下的边界行为。

功能与动机

cgroup_memory_limit_bytes 原实现固定读取 /sys/fs/cgroup/memory.max（v2）与 /sys/fs/cgroup/memory/memory.limit_in_bytes（v1），即 mount root 的上限。对直接位于该 cgroup 的进程（普通容器）这是正确的，但对进程位于更深的 cgroup（systemd scope 带 MemoryMax、容器以 --cgroup-parent 启动、或任何嵌套更紧的 cgroup）时，读到的是父级更大的数字。pinned host memory 便按进程实际无法拥有的内存来规划——这正是该模块要防止的过度承诺——最终以 OOM kill 收场，而不是优雅回退到 pageable 内存。PR body 明确要求从 /proc/self/cgroup 解析进程自身 cgroup 并沿链到 mount root 取最紧 cap。

实现拆解

1. 解析进程自身 cgroup：在 host_memory_budget.py 中新增 _own_cgroup_path()，读取 _PROC_SELF_CGROUP 常量（/proc/self/cgroup），按冒号切分为 hierarchy、controller 列表、路径三段；优先选择 v2 行（controller 字段为空）或 controller 列表包含 memory 的 v1 行，返回第三个字段；文件读取失败或格式异常返回空串，作为后续查找的基准。
2. 构造目录链：新增 _cgroup_dirs(mount)。将 /proc 报告路径按 / 分段，从最长后缀开始尝试 os.path.join(mount, *parts[start:])，用 os.path.isdir 找到第一个真实存在的目录作为叶子；容器内外路径不对齐时（例如 /proc 报 /docker/8e10... 而容器内 mount 下没有 docker 目录）会自然回退到 mount 本身；随后从叶子逐级 os.path.dirname 回溯到 mount，得到 [leaf, ..., mount] 的目录链。
3. 取最紧 cap：重写 cgroup_memory_limit_bytes()。常量从固定文件路径 _CGROUP_V2/_CGROUP_V1 改为 _CGROUP_MOUNTS（v2 优先、v1 兜底的 (mount root, limit 文件名 , usage 文件名) 三元组）。对每个 mount，遍历目录链读取每层 limit ，跳过无上限（None 或 >= 1<<62）的层，保留 limit 最小（最紧）的 cap 及其对应目录

的 usage。

4. 测试配套：test/unit/test_host_memory_budget.py 中 _point_at 重构为在 tmp_path 下构造 v2/v1 双 mount 目录树，新增 own_path 参数模拟 /proc/self/cgroup 内容、nested 参数模拟嵌套 cgroup（按 own_path 分段在 v1 mount 下逐级建目录）；新增 _write_cgroup 辅助函数；新增 TestNestedCgroup 的 4 个用例覆盖嵌套更紧、嵌套更松、路径不存在回退 mount、报告路径后缀命中四种场景。原有 TestCgroupLimit 用例适配新常量结构。
5. 无配置与部署改动；仅 2 个文件（源码 + 测试），单提交完成。

关键文件：

- python/sglang/multimodal_gen/runtime/managers/memory_managers/host_memory_budget.py（模块 内存预算；类别 source；类型 core-logic；符号 cgroup_memory_limit_bytes, _own_cgroup_path, _cgroup_dirs）：核心修复文件。重写 cgroup_memory_limit_bytes 为沿目录链取最紧 cap，新增 _own_cgroup_path 与 _cgroup_dirs 处理 /proc/self/cgroup 解析和容器内外路径对齐问题。
- python/sglang/multimodal_gen/test/unit/test_host_memory_budget.py（模块 单测配套；类别 test；类型 test-coverage；符号 _point_at, _write_cgroup, TestNestedCgroup, test_a_tighter_nested_cap_wins_over_the_root）：测试配套。_point_at 重构为构造 v2/v1 双目录树并支持 own_path/nested 参数，新增 TestNestedCgroup 4 个用例覆盖嵌套 cap 竞争、路径回退与后缀命中。

关键符号：_own_cgroup_path, _cgroup_dirs, cgroup_memory_limit_bytes, _point_at, _write_cgroup, TestNestedCgroup

关键源码片段

python/sglang/multimodal_gen/runtime/managers/memory_managers/host_memory_budget.py

核心修复文件。重写 cgroup_memory_limit_bytes 为沿目录链取最紧 cap，新增 _own_cgroup_path 与 _cgroup_dirs 处理 /proc/self/cgroup 解析和容器内外路径对齐问题。

```
def _cgroup_dirs(mount: str) -> list[str]:
    """进程实际所在 cgroup 的目录链：从叶子逐级回溯到 mount 根。

    核心难点：/proc/self/cgroup 报告的路径相对宿主 cgroup root，而容器内
    看到的 mount 已经是容器自己的 cgroup，二者不能直接拼接。实测某容器内
    /proc 报 /docker/8e10...，而 /sys/fs/cgroup 下没有 docker 目录。
    因此从最长后缀开始探测，第一个真实存在的目录就是容器视角下的叶子，
    找不到则回退到 mount 本身。
    """
    if not os.path.isdir(mount):
        return []
    parts = [part for part in _own_cgroup_path().split("/") if part]
    leaf = mount
    for start in range(len(parts)):
        candidate = os.path.join(mount, *parts[start:])
```

```
        if os.path.isdir(candidate):
            leaf = candidate
            break
    dirs = [leaf]
    while dirs[-1] != mount:
        dirs.append(os.path.dirname(dirs[-1]))
    return dirs


def cgroup_memory_limit_bytes() -> tuple[int, int] | None:
    """进程在其 cgroup 下的 (最紧 cap, 对应 usage)，无上限时返回 None。

    cgroup 的 limit 沿目录链向下生效，进程实际被限制在最紧的一层；
    systemd scope 带 MemoryMax、容器以 --cgroup-parent 启动时，mount root
    的数字比进程真实可用的内存大得多，按它规划 pinned host memory 就会
    过度承诺，最终以 OOM kill 收场而不是优雅回退到 pageable 内存。
    """
    for mount, limit_name, usage_name in _CGROUP_MOUNTS:
        tightest = None
        for directory in _cgroup_dirs(mount):
            limit = _read_int(os.path.join(directory, limit_name))
            if limit is None or limit >= _UNLIMITED_ABOVE:
                continue # 该层无上限，不参与"最紧"比较
            if tightest is not None and limit >= tightest[0]:
                continue # 只保留更紧的 cap
            tightest = (limit, _read_int(os.path.join(directory, usage_name)) or 0)
        if tightest is not None:
            return tightest # v2 读到有效 cap 直接返回，v1 仅兜底
    return None
```

python/sglang/multimodal_gen/test/unit/test_host_memory_budget.py

测试配套。_point_at 重构为构造 v2/v1 双目录树并支持 own_path/nested 参数，新增 TestNestedCgroup 4 个用例覆盖嵌套 cap 竞争、路径回退与后缀命中。

```
def _point_at(monkeypatch, tmp_path, *, v2=None, v1=None, own_path="", nested=None):
    """把 cgroup 查询重定向到 tmp_path 下的目录树。

    own_path 模拟 /proc/self/cgroup 报告的路径；nested 是该路径对应的
    (limit, usage)，二者共同模拟"进程被挂在 mount root 之下"的嵌套场景。
    """
    roots = {}
    for key, files, values in (("v2", _V2_FILES, v2), ("v1", _V1_FILES, v1)):
        roots[key] = _write_cgroup(tmp_path / key, files, values)
    if nested is not None:
        # 按 own_path 的分段在 v1 mount 下逐级创建目录，模拟容器内的
        # systemd scope 或 --cgroup-parent 形成的嵌套 cgroup
        leaf = roots["v1"]
        for part in [part for part in own_path.split("/") if part]:
            leaf = leaf / part
```

```
        _write_cgroup(leaf, _V1_FILES, nested)

    monkeypatch.setattr(
        host_memory_budget,
        "_CGROUP_MOUNTS",
        ((str(roots["v2"]),) + _V2_FILES, (str(roots["v1"]),) + _V1_FILES),
    )
    proc = tmp_path / "proc_self_cgroup"
    proc.write_text(f"11:memory:{own_path}\n" if own_path else "")
    monkeypatch.setattr(host_memory_budget, "_PROC_SELF_CGROUP", str(proc))
```

评论区精华

该 PR 无 review 评论，设计决策完全由 PR body 中的实测数据支撑：psutil 报告 2015.7 GiB 而 cgroup 上限 1117.2 GiB（900 GiB 差距）；容器内 /proc/self/cgroup 报 11:memory:/docker/8e10...，而 /sys/fs/cgroup/memory 下拼接该路径不存在。作者据此提出“渐进缩短后缀直到找到存在的目录”的算法，并承诺“无论容器如何设置都能找到叶子”。这些测量直接决定了 _cgroup_dirs 的回退语义，是理解实现的关键。

- 暂无高价值评论线程

风险与影响

- 风险：
  1. 探测依赖目录存在性：_cgroup_dirs 用 os.path.isdir 判断，若 cgroup 文件系统仅暴露文件不暴露目录（罕见），会回退到 mount root，行为与旧版一致，不会比改前更糟。
  2. v1/v2 混合挂载边界：_own_cgroup_path 对 v1/v2 混合环境可能优先命中 v2 路径，而 v1 mount 与 v2 mount 下目录结构通常不同，v1 侧可能 miss 嵌套目录并回退 mount root；当前测试主要覆盖 v1 嵌套，v2 嵌套用例缺失。
  3. 最紧 cap 对应的 usage 取最紧层目录的 usage，cgroup usage 包含子层级，语义正确；但若最紧 cap 层与叶子层不同，usage 读数与进程实际消耗可能略有偏差，影响内存预算精度。
  4. /proc/self/cgroup 的非标准格式（多行、多控制器）可能导致路径解析不中，回退 mount root，等同旧行为，风险可控。
  5. 性能无风险：启动时读取少量小文件，对运行时无影响。 - 影响：影响范围集中在 sglang.multimodal_gen 的 host memory 预算模块：修复了 diffusion 服务在嵌套 cgroup 部署（K8s Pod 内 systemd scope、--cgroup-parent 容器等）下按错误上限规划 pinned host memory 导致 OOM kill 的问题。cgroup_memory_limit_bytes 签名不变，对调用方透明；普通容器根 cgroup 场景行为不变（回退到 mount root 即旧逻辑）。对团队而言，该修复的正确性测试模式（用 tmp_path 构造嵌套 cgroup 树 + monkeypatch /proc 路径）可复用于其他平台与后续 v2 嵌套覆盖。 - 风险标记：核心路径变更（启动内存规划），cgroup 双栈兼容（v1/v2），依赖 /proc/self/cgroup 格式

关联脉络

- 暂无明显关联 PR