

PR #56747 完整报告

vllm-project/vllm

[CI] Collect GPU memory telemetry for MIG slices

合并时间: 2026-09-14 12:30

原文链接: <http://prhub.com.cn/vllm-project/vllm/pull/56747>

执行摘要

- 一句话: MIG 切片 CI 内存遥测补齐, 复用 NVML 绑定采集 used/total。
- 推荐动作: 值得精读, 尤其适合 CI/ 可观测性方向的同学。三个设计点可以复用: 一是用 `importlib.util.find_spec + spec_from_file_location` 复用捆绑绑定而不引入重型运行时依赖; 二是把 CUDA 驱动交互放进带超时的一次性子进程, 避免在常驻采样进程里创建 CUDA 上下文; 三是采样失败永远保持 `unknown` 而非合成零值, 保证 dashboard 语义不会被污染。

功能与动机

MIG CI 任务 (如 H200 18GB、35GB 切片) 此前只产生 GPU 事件而没有内存读数: `nvidia-smi` 在 MIG 容器中要么因权限无法查询父 GPU, 要么父设备内存不能代表分配给该任务的切片。PR body 明确目标为“Record memory used/total from each assigned MIG instance through vLLM's standalone NVML binding, using the existing event format”, 并强调“不枚举 NVML 兄弟实例、不替代父内存、不可用读数保持 `unknown`”, 避免把父设备内存误标到切片上污染 dashboard 数据。这与关联 Issue #55481 的运行时 NVML 回退同属 MIG 内存可见性问题链, 但改动对象是独立的 CI 遥测收集器。

实现拆解

1. 显式 MIG 可见性解析: 新增 `visible_mig_devices()`, 处理 `NVIDIA_VISIBLE_DEVICES` 与 `CUDA_VISIBLE_DEVICES` 的组合映射——CUDA 数字索引换算成 NVIDIA 侧的 MIG-UUID 列表, 去重并截断到 `MAX_DEVICES`; 任何非 MIG 值、越界或混合取值都返回空列表, 从而安全回退到旧 `nvidia-smi` 路径。
2. 独立加载 NVML 绑定: 新增 `load_nvml()`, 用 `importlib.util.find_spec("vllm")` 定位已安装 `vllm` 包但不执行其 `__init__`, 直接以 `ci_nvml` 名义加载 `third_party/pynvml.py`, 保证测试镜像 (wheel 安装 + 单独 tests checkout) 也能使用, 且不触发 Torch/CUDA 导入。
3. CDI 场景的 UUID 发现: 新增 `cuda_visible_mig_devices()`, 在一次性子进程中用 `ctypes.CDLL("libcuda.so.1")` 对 CUDA 可见设备调用 `cuDeviceGetUuid_v2` 获取 compute instance UUID, 再经 NVML `nvmlDeviceGetHandleByUUID + nvmlDeviceIsMigDeviceHandle` 校验后才采纳; 不创建 CUDA 上下文, 绝不枚举 NVML 兄弟设备。
4. MIG 内存采样与主循环切换: 新增 `query_mig_samples()`, 按 UUID 拿句柄、确认是 MIG 实例后读取 `nvmlDeviceGetMemoryInfo`, 产出 `gpu.memory.used/total` 与

gpu.status=mig_memory_only; 查询异常时保留 unavailable_mig_memory 状态，绝不合成零值。collect() 入口先走显式解析，命中即切到 python __file__--query-mig 子进程；未命中且 nvidia-smi 输出出现 unsupported_mig 时一次性尝试 --discover-mig，成功后无缝切换采样方式。

5. 测试与文档配套：tests/test_ci_otel.py 新增 7 组测试，覆盖可见性重映射、CDI 发现只读 CUDA 可见 UUID、父设备拒绝、权限失败、超时边界、无 vLLM/Torch 导入的绑定加载；README.md 同步说明 MIG 内存采样行为与 unknown 语义。

关键文件：

- .buildkite/scripts/ci-otel/ci_gpu.py (模块 遥测采集；类别 source；类型 dependency-wiring；符号 visible_mig_devices, load_nvml, cuda_visible_mig_devices, query_mig_samples)：核心实现文件：新增 MIG 可见性解析、独立 NVML 绑定加载、CDI UUID 发现与 MIG 内存采样，并重构 collect 主循环以支持显式 MIG 与 CDI 发现两条路径。
- .buildkite/scripts/ci-otel/tests/test_ci_otel.py (模块 遥测测试；类别 test；类型 test-coverage；符号 test_mig_assignment_obey_cuda_visibility, test_mig_memory_uses_assigned_handle_without_parent_metrics, test_mig_memory_never_substitutes_parent_or_permission_failure, test_cdi_discovery_uses_only_cuda_visible_uuids)：测试配套：新增 7 组参数化单元 / 集成测试，覆盖 MIG 可见性重映射、CDI 发现边界、父设备拒绝、权限失败、超时与无导入加载，是验证核心实现正确性的关键。
- .buildkite/scripts/ci-otel/README.md (模块 遥测采集；类别 docs；类型 documentation)：同步更新 CI 遥测文档，明确 MIG 内存采样、CDI 发现与 unknown 语义，方便后续维护者理解事件字段变化。

关键符号：visible_mig_devices, load_nvml, cuda_visible_mig_devices, query_mig_samples, collect, main

关键源码片段

.buildkite/scripts/ci-otel/ci_gpu.py

核心实现文件：新增 MIG 可见性解析、独立 NVML 绑定加载、CDI UUID 发现与 MIG 内存采样，并重构 collect 主循环以支持显式 MIG 与 CDI 发现两条路径。

```
def load_nvml():
    """加载 vLLM 捆绑的独立 NVML 绑定，不导入 vLLM 或 Torch。"""
    # 测试镜像通常把 vLLM 装成 wheel，checkout 里只放 tests 和 helper。
    # 用 find_spec 定位顶层包路径，不会执行 vllm/__init__.py，避免连锁导入。
    package = importlib.util.find_spec("vllm")
    if package is None or package.origin is None:
        raise ImportError("Installed vLLM package unavailable")
    path = Path(package.origin).parent / "third_party/pynvml.py"
    spec = importlib.util.spec_from_file_location("ci_nvml", path)
    if spec is None or spec.loader is None:
        raise ImportError("Bundled NVML binding unavailable")
    module = importlib.util.module_from_spec(spec)
```

```

sys.modules[spec.name] = module
spec.loader.exec_module(module)
return module

```

```

def query_mig_samples(devices: list[str], timestamp_ns: int) -> list[dict]:
    # 直接复用 vLLM 自带的 NVML 绑定，全程不初始化 CUDA/Torch。
    pynvml = load_nvml()
    events = []
    pynvml.nvmlInit()
    try:
        for index, uuid in enumerate(devices[:MAX_DEVICES]):
            if not uuid.startswith("MIG-"):
                continue
            # 默认状态是 unavailable_mig_memory，查询失败时保留，不合成零值。
            attributes: dict[str, str | int | bool] = {
                "gpu.uuid": uuid,
                "gpu.index": index,
                "gpu.name": "NVIDIA MIG",
                "gpu.status": "unavailable_mig_memory",
            }
            try:
                # NVML 先用 UUID 拿句柄，并确认真的是 MIG 实例，避免误查父设备。
                handle = pynvml.nvmlDeviceGetHandleByUUID(uuid)
                if not pynvml.nvmlDeviceIsMigDeviceHandle(handle):
                    continue
                memory = pynvml.nvmlDeviceGetMemoryInfo(handle)
                used = _number(str(memory.used))
                total = _number(str(memory.total))
                if used is not None and total and used <= total:
                    attributes.update({
                        "gpu.memory.used": used,
                        "gpu.memory.total": total,
                        "gpu.status": "mig_memory_only",
                    })
                # NVML 不支持按 MIG 实例的利用率，保持缺失。
            except pynvml.NVMLError:
                pass
            events.append({
                "time_ns": timestamp_ns,
                "name": "ci.gpu.sample",
                "attributes": attributes,
            })
    finally:
        pynvml.nvmlShutdown()
    return events

def cuda_visible_mig_devices() -> list[str]:
    """在子进程中通过 CUDA UUID 发现 CDI 分配的 MIG 实例，不创建 CUDA 上下文。"""

```

```

pynvml = load_nvml()
driver = ctypes.CDLL("libcuda.so.1")
count = ctypes.c_int()
# cuInit/cuDeviceGetCount 失败时直接返回空，由调用方回退到旧 nvidia-smi 路径。
if driver.cuInit(0) or driver.cuDeviceGetCount(ctypes.byref(count)):
    return []
devices = []
pynvml.nvmlInit()
try:
    for index in range(min(count.value, MAX_DEVICES)):
        device = ctypes.c_int()
        identifier = (ctypes.c_ubyte * 16)()
        if driver.cuDeviceGet(ctypes.byref(device), index):
            continue
        # v2 入口在 MIG 模式下返回 compute instance UUID。
        if driver.cuDeviceGetUuid_v2(ctypes.byref(identifier), device):
            continue
        uuid = "MIG-" + str(uuid_module.UUID(bytes=bytes(identifier)))
        try:
            handle = pynvml.nvmlDeviceGetHandleByUUID(uuid)
            # 只认 MIG 实例句柄，绝不会把父设备当成切片。
            if pynvml.nvmlDeviceIsMigDeviceHandle(handle):
                devices.append(uuid)
        except pynvml.NVMLError:
            continue
finally:
    pynvml.nvmlShutdown()
return devices

```

评论区精华

本 PR 没有实质性的 review 技术讨论：唯一评论来自 claude[bot] 的自动提醒（“This repository is configured for manual code reviews”），不涉及代码内容。作者在 PR body 中主动交代了设计权衡与 canary 迭代：早期 canary 暴露了 CDI 可见性与已安装 wheel 布局两个问题，最终版本因此引入 `cuDeviceGetUuid_v2` 发现路径和 `find_spec` 定位绑定的方案；同时明确指出与 #50463（运行时 NVML 内存查找）和 #55481（调度器 NVML 回退）是不同改动。

- 暂无高价值评论线程

风险与影响

- 风险：
 - 加载路径耦合打包布局：`load_nvml()` 依赖 vllm wheel 内 `third_party/pynvml.py` 的相对位置，vLLM 打包结构调整会直接破坏 MIG 遥测，且失败时只有 `collect` 的兜底 `except Exception` 静默跳过。
 - 平台限制：`ctypes.CDLL("libcuda.so.1")` 仅适用于 Linux NVIDIA 环境；`--query-mig/--discover-mig` 子进程依赖 `sys.executable` 可用。

- 静默降级：CDI 发现只尝试一次，超时或失败后停留在老 `nvidia-smi` 路径且不重试；`visible_mig_devices()` 在环境变量组合误配时直接返回空，团队可能误以为功能正常却看不到内存数据。
- 事件语义扩展：新增 `gpu.status` 枚举值 `mig_memory_only` / `unavailable_mig_memory`，`dashboard` 查询与告警若按旧字段过滤会漏掉 MIG 序列。对运行时推理、调度、模型加载无影响；风险集中在 CI 遥测采集链路。
- 影响：影响范围集中在 Buildkite CI 的 GPU 遥测采集器：MIG 任务（H200 18GB、35GB 等切片）将开始上报 per-slice 的 `used/total` 内存，使容量规划、内存泄漏检测和多任务资源审计成为可能；全 GPU、CPU、AMD 与显式非 MIG 路径采样行为保持不变。端到端 canary 确认 155/155 数值样本、无截断、无父设备序列，说明对现有 `dashboard` 事件流是纯增量扩展。对 vLLM 运行时和用户侧功能无影响。
- 风险标记：依赖 `wheel` 内 `pynvml` 路径，仅 Linux 的 `libcuda` 加载，静默降级回 `nvidia-smi`，CDI 发现失败不重试，新 `telemetry` 字段需 `dashboard` 兼容

关联脉络

- PR #55481 [Bugfix][Scheduler] Fall back when NVML cannot report GPU memory: 同属 MIG 内存可见性问题链：该 PR 修运行时调度器默认值在 NVML 失败时的回退，本 PR 是 CI 遥测侧补 MIG 内存读数，二者独立且互补。
- PR #50463 [Bugfix] Use MIG handle for GPU memory lookup: PR body 明确指出该 PR 修复运行时 NVML 内存查找使用 MIG 句柄；本 PR 的 `query_mig_samples` 采用同样的“先拿 MIG 句柄再查内存”原则，是同一模式的 CI 侧镜像。