

PR #39697 完整报告

sgl-project/sglang

ci: fix always-failing coverage job, add by-GPU-count view

合并时间: 2026-09-16 16:13

原文链接: <http://prhub.com.cn/sgl-project/sglang/pull/39697>

执行摘要

- 一句话: 修复始终失败的覆盖率 Job, 新增按 GPU 数统计视图
- 推荐动作: 值得精读, 尤其是长期维护 CI 的人。重点看两处设计决策: 一是用 `set -o pipefail + ll echo` 把「非门禁」从意外行为改成显式契约, 以及随之而来的问题发现责任转移; 二是 `_SUITE_ACCEL_COUNT_RE` 用 `(?:^I-)` / `(?:-I$)` 锚定来区分 runner 规模与 `-tp4`、`qwen3-235b` 这类同名数字, 这类「从命名反解元数据」的做法在新增套件命名不规范时会静默失真, 建议后续补一个覆盖 `stress`、`mm-gen-*`、`-tp4` 等边界的单测。可略读 `by-gpu-count job` 与 `JSON` 字段部分, 属机械扩展。

功能与动机

PR body 明确指出问题不是测试失败而是超时: *The Unit Test Code Coverage job has been red on every scheduled run since 2026-08-18. It's a step timeout, not a test failure: the unit suite finished in 9m45s on the last green day, just under the 10-minute cap, and has since grown to ~11k tests needing ~22 minutes.* 第二层动机是掩盖失败: *That last green run also reported 127 failed, 6294 passed as a success, because pytest l tee takes its exit status from tee.* 即退出码取自 `tee`, 测试失败被静默吞掉。第三层动机来自容量规划需求: *Second half adds a runner-size view ... Current numbers: 914 enabled 1-GPU registrations (590 CUDA), 146 2-GPU, 223 4-GPU, 200 8-GPU.*, 需要在不展开每个套件的前提下回答「我们到底有多少 1-GPU 测试」。

实现拆解

1. 解除超时导致的假红: `.github/workflows/ci-coverage-overview.yml` 中 `unit-test-coverage job` 的 `timeout-minutes` 从 30 提到 60, `Run unit tests with coverage step` 的从 10 提到 45。原因是套件从 7.7k 收集项涨到约 11k, 单进程跑完需约 22 分钟, 10 分钟上限会在 46% 处直接 `kill` 掉进程。
2. 把退出码语义显式化: 在 `pytest` 命令前加 `set -o pipefail`, 并在管道末尾接 `ll echo "pytest exited $? -- reported below, not gating this job"`。前者修掉 `l tee` 从 `tee` 取退出码的缺陷, 后者刻意保留 `job` 非门禁属性——因为单进程跑全树存在正式 CI 不存在的交叉污染 (重复进程组初始化、泄漏默认 `device`)。
3. 让 `step summary` 不再静默留白: 把 `grep -E '^=+.*passed' coverage_output.txt >> $GITHUB_STEP_SUMMARY ll true` 改为 `ll echo "pytest did not reach its summary line (step killed or crashed)" >> $GITHUB_STEP_SUMMARY`, 被 `kill` 或崩溃时

summary 里能直接看到原因，而不是空白。

4. 新增 runner 规模解析层 (scripts/ci/utils/ci_coverage_report.py) : 新增模块级正则 `_SUITE_ACCEL_COUNT_RE = re.compile(r"(?:^|I-)(\d+)-(?:gpuInpu)(?:-I$)")`，以及 `get_accel_count()`、`accel_count_label()`、`sorted_accel_counts()` 三个符号。锚定 `(?:^|I-)` 与 `(?:-I$)` 是为了避免把 `-tp4` 之类的并行度后缀或 `qwen3-235b` 之类的模型名误判为 runner 规模。
5. 把规模维度接入报告数据流: `organize_test_data()` 增加 `by_accel` 分组并写入返回 dict; `generate_summary_section()` 新增一个可折叠的 GPU Count Summary 表格 (行 = 规模, 列 = 各活跃后端 + Total + Disabled) ; 新增 `generate_by_gpu_count_section()` 提供逐后端、逐套件的钻取表 (Suite / Enabled / Disabled / Est. Time / Nightly 或 Per-Commit) ; `generate_markdown_report()` 在 `--section all` 中追加该小节，并接受新的 `--section by-gpu-count`。
6. JSON 导出扩容: `generate_json_report()` 新增 `gpu_count_summary`, key 与 Markdown 标签一致 ("1-GPU"... "Unsize"), value 含各后端计数、gpus、enabled、disabled 与 suites 列表，供下游脚本消费，属增量 schema 变更。
7. 补齐 workflow 入口: 新增独立的 `by-gpu-count` job (ubuntu-latest + Python 3.10) 运行 `python scripts/ci/utils/ci_coverage_report.py --section by-gpu-count`。
8. 测试与配置配套: 本 PR 未新增或修改任何测试，也没有为新的解析函数补单测; 验证方式依赖下一次定时运行或手动执行 `--section by-gpu-count`。

关键文件:

- `scripts/ci/utils/ci_coverage_report.py` (模块 覆盖率报告; 类别 infra; 类型 infrastructure; 符号 `get_accel_count`, `accel_count_label`, `sorted_accel_counts`, `generate_by_gpu_count_section`) : 报告生成主脚本: 新增 runner 规模解析层 (正则 + `get_accel_count` / `accel_count_label` / `sorted_accel_counts`)、`organize_test_data` 的 `by_accel` 分组、`generate_by_gpu_count_section` 钻取小节、summary 中的 GPU Count Summary 表格、JSON 的 `gpu_count_summary` 字段，以及 `--section by-gpu-count` 选项，是整个容量视图能力的落点。
- `.github/workflows/ci-coverage-overview.yml` (模块 workflow配置; 类别 infra; 类型 configuration) : CI 门禁侧的全部行为改动: 抬高 `unit-test-coverage` 的 job / step 超时上限，用 `set -o pipefail` 与显式非门禁 `|| echo` 修正被 `tee` 吞掉的退出码，让 summary 在未跑到 `pytest` 汇总行时给出显式提示，并新增 `by-gpu-count` job 作为新报告的入口。

关键符号: `get_accel_count`, `accel_count_label`, `sorted_accel_counts`, `generate_by_gpu_count_section`, `organize_test_data`, `generate_summary_section`, `generate_markdown_report`, `generate_json_report`

关键源码片段

`scripts/ci/utils/ci_coverage_report.py`

报告生成主脚本: 新增 runner 规模解析层 (正则 + `get_accel_count` / `accel_count_label` / `sorted_accel_counts`)、`organize_test_data` 的 `by_accel` 分组、`generate_by_gpu_count_section` 钻取小节、summary 中的 GPU Count Summary 表格、

JSON 的 `gpu_count_summary` 字段，以及 `--section by-gpu-count` 选项，是整个容量视图能力的落点。

```
# 套件名用 `
```

```
def get_accel_count(test: CIRegistry) -> Optional[int]:
    """测试所属套件占用的 GPU / NPU 数量；名字里没有规模信息时返回 None。

    返回 None 的三类套件：没有规模 token 的 `stress`、`nightly-amd-vlm`
    和 CPU 套件；以及合成出来的 `mm-gen-*` —— multimodal_gen 把同一个套件
    摊到 1-gpu 与 2-gpu 两种 runner 上，没有唯一答案可报。
    """

    match = _SUITE_ACCEL_COUNT_RE.search(test.effective_suite or "")
    return int(match.group(1)) if match else None
```

```
def accel_count_label(count: Optional[int]) -> str:
    """把加速器数量映射为表格行标签，Markdown 与 JSON 共用同一套 key。"""
    return f"{count}-GPU" if count is not None else "Unsize"
```

```
def sorted_accel_counts(by_accel: dict) -> list:
    """加速器数量升序排列，`None`（未标注规模）排到最后。"""
    return sorted(by_accel.keys(), key=lambda c: (c is None, c if c is not None else 0))
```

```
def generate_by_gpu_count_section(data: dict) -> str:
    """生成 “All Tests by GPU Count” 小节。
```

面对的是和 by-suite 小节同一批注册项，只是把透视轴从套件名换成 runner 规模：哪个后端吃 1-GPU 机器最多、8-GPU 的负载压在哪里，一眼可见。

```
"""
```

```
lines = []
by_accel = data["by_accel"]
```

```
lines.append("# All Tests by GPU Count\n")
```

```
# sorted_accel_counts 保证 `None`（Unsize）排在最后，数字规模升序，便于跨天对比
for count in sorted_accel_counts(by_accel):
    accel_tests = by_accel[count]
    a_disabled = sum(1 for t in accel_tests if t.disabled)
    a_enabled = len(accel_tests) - a_disabled
```

```
lines.append("<details>")
lines.append(
```

```

        f"<summary><h2>{accel_count_label(count)} "
        f"({a_enabled} enabled, {a_disabled} disabled)</h2></summary>\n"
    )

    accel_by_backend = defaultdict(list)
    for t in accel_tests:
        accel_by_backend[t.backend.name].append(t)

    # 只遍历展示顺序里的后端；该规模下没有注册项的后端整块跳过，
    # 避免表格里出现一整排 0
    for backend in BACKEND_DISPLAY_ORDER:
        backend_tests = accel_by_backend.get(backend, [])
        if not backend_tests:
            continue

        b_disabled = sum(1 for t in backend_tests if t.disabled)
        b_enabled = len(backend_tests) - b_disabled
        lines.append(f"### {backend} ({b_enabled} enabled, {b_disabled} disabled)\n")
        lines.append("Suite | Enabled | Disabled | Est. Time | Type |")
        lines.append("-----|-----|-----|-----|-----|")

        # 套件内再聚合：估算时长只累加 enabled 项，
        # Type 取「只要有一个 nightly 即算 Nightly」
        backend_suites = defaultdict(list)
        for t in backend_tests:
            backend_suites[t.effective_suite].append(t)

        for suite in sorted(backend_suites.keys()):
            suite_tests = backend_suites[suite]
            s_disabled = sum(1 for t in suite_tests if t.disabled)
            s_enabled = len(suite_tests) - s_disabled
            s_est_time = sum(t.est_time for t in suite_tests if not t.disabled)
            is_nightly = any(t.nightly for t in suite_tests if not t.disabled)
            suite_type = "Nightly" if is_nightly else "Per-Commit"
            lines.append(
                f"| {suite} | {s_enabled} | {s_disabled} | "
                f"{s_est_time:.0f}s | {suite_type} |"
            )

        lines.append("")

    lines.append("</details>\n")

    return "\n".join(lines)

```

评论区精华

本 PR `comments_count = 0`、`review_comments_count = 0`、只有 1 个 commit 且 author 与 merged_by 均为 alisonshao，因此不存在 review 交锋。可提炼的「讨论」全部来自 PR

body 的自述设计决策：其一，明确把覆盖率 job 定位为 advisory——This adds set -o pipefail and an explicit non-gating ll echo, so the job stays advisory on purpose and the counts show up in the step summary.; 其二，对 127 个失败的定性——Those failures are cross-test contamination from running the whole tree in one process (duplicate process-group init, leaked default device). CI proper runs one file per process, and the same files are green in base-b-test-1-gpu-small. Worth fixing separately., 即根因修复被显式推迟到后续 PR。

- 覆盖率 job 保持非门禁：pipefail 之后为什么还要 ll echo (testing): 决策为：修掉 l tee 吞退出码的缺陷，但刻意不让 pytest 失败导致 job 变红，改为把计数推到 step summary；根因修复留待后续 PR (Worth fixing separately)。风险是真正回归时也不会自动红灯。
- 从 suite 名反解 runner 规模的正则边界与 Unsized 口径 (design): 解析层设计已自洽并规避了两类典型误判，但命名约定之外的套件会静默落入 Unsized，缺少告警或单测兜底。

风险与影响

- 风险：
 1. 门禁与 advisory 的张力：ll echo 让 pytest 的非零退出码永远不失败 job。如果未来出现真正的回归（而非已知交叉污染），job 依旧全绿，只能靠人去看 step summary 里的计数，问题发现从「自动红」退化为「依赖人工巡检」。
 2. 超时上限放宽 4.5 倍：step 从 10 分钟到 45 分钟、job 从 30 分钟到 60 分钟，1-gpu-h100 runner 的最长占用翻倍以上。若套件继续按 body 中的速度增长，45 分钟同样会被吃满并再次被 kill，届时失败形态又回到「红 job」，只是这次至少会打出提示。
 3. 解析逻辑强依赖命名约定：get_accel_count() 完全依赖 suite 名中的 <n>-gpu / <n>-npu token。一旦新增套件写成 gpu1、1gpu、gpu-1 或不带规模（如 stress、nightly-amd-ml、CPU 套件、合成的 mm-gen-*），会静默归入 Unsized 而不是报错，统计口径出现偏差时不易察觉，直接影响 GPU Count Summary 与 gpu_count_summary 的数字可信度。
 4. 根因未修：单进程跑全树的交叉污染（重复进程组初始化、泄漏默认 device）仍在，报告里的 passed / failed 计数在此期间不能当作质量门禁指标使用；body 已声明 Worth fixing separately。
 5. 缺少测试覆盖：scripts/ci/utis/ci_coverage_report.py 新增 155 行、含正则解析与两层聚合逻辑，却没有对应单测；accel_count_label 的标签契约同时被 Markdown 和 JSON 复用，一旦改动需同步两处。
 6. 报告体积增长：generate_by_gpu_count_section() 对每个规模 × 每个后端 × 每个套件都展开 <details> 与表格，--section all 的输出会随套件数线性膨胀，可能影响 GitHub summary 渲染速度。- 影响：对开发者：覆盖率 job 从「每次定时运行必红的噪音」变成可用信号，step summary 中能看到真实 passed / failed 计数、TOTAL 行以及被 kill 时的显式提示，减少误判为测试回归的时间成本。对系统与 CI 容量：1-gpu-h100 上单 job 的最长占用从 30 分钟放宽到 60 分钟，调度占用增加，但换来不再被 46% 处截断。对团队决策：新的 GPU Count Summary 给出 fleet 需求分布（914 个启用的 1-GPU 注册项，其中 590 为 CUDA；146 个 2-GPU、223 个 4-GPU、200 个 8-GPU），是 runner 池扩容与测试切分（例如把 1-GPU 套件拆到更多机器）的量化输入。对下游工

具：JSON 报告新增 `gpu_count_summary` 字段属增量变更，既有 key 不变，兼容性风险低。 - 风险标记：CI 门禁有意放行失败，超时上限大幅放宽，解析依赖套件命名约定，单进程跑全树交叉污染未修，新增脚本逻辑缺少单测

关联脉络

- PR #39295 [SRT] Clean up no-op compiler pass, dead helpers, and migration tests: 该 PR 删除 `test/registered/unit/` 下的迁移测试与 `ratchet` 测试，正是本覆盖率 job 收集的那棵测试树，套件规模的增减直接决定 job 时长与超时上限是否够用。
- PR #39293 [Diffusion] Clean up obsolete worker plumbing, dead helpers, and tests: 同样删除 `multimodal_gen` 下的单测，而本 PR 的 `get_accel_count()` 专门为 `mm-gen-*` 这类「一个套件摊到多种 runner 规模」的情况返回 `Unsize`d，两者在测试树结构上有直接关联。
- PR #34712 [Fix] Spawn, don't fork, the benchmark server process: 同为 CI / 测试基础设施修复，处理 `fork` 与进程组初始化导致的崩溃，与本 PR 中「单进程跑全树引发重复进程组初始化」的交叉污染根因同源，是可参考的修复范式。
- PR #39713 [Router] Bound the e2e worker memory budget so prefill graph capture stops OOMing: 同属 `run-ci / ci / infra` 线的 job 资源边界调整（该 PR 限定显存预算，本 PR 放宽时间上限），体现 CI 资源参数在套件增长下反复校准的演进脉络。