PR #34328 完整报告

sgl-project/sglang

[AMD][CI] CI: fix AMD 2-GPU multimodal-gen partition-count abort

合并时间：2026-08-13 08:32

原文链接：http://prhub.com.cn/sgl-project/sglang/pull/34328

执行摘要

- 一句话：修复 AMD 2-GPU diffusion CI 分区中止，改为 LPT 均衡分片
- 推荐动作：值得精读。原因：(1) 这是一个典型的 CI 结构性 bug 修复，展示了“修计数 vs 解耦”的设计取舍，以及作者如何通过统一两条分区路径来防止代码漂移；(2) 对 “把独立测试文件与参数化用例混排”带来的副作用（fast-fail 语义变化）有完整分析，并配套了有针对性的回归测试；(3) PR body 对哪些车道行为变化、哪些不变做了逐条清单，是 CI 变更的好示范。关注点：_run_partition_assignment 的统一执行路径、assign_partition 的确定性切片设计、以及 PR 末尾披露但刻意未修的 diffusion_case_parser.py 预存在 bug（CUDA standalone 测试可能已静默停跑数月），值得后续单独跟进的 issue。

功能与动机

PR body 明确指出：Every “multimodal-gen-test-2-gpu-amd” shard on both “pr-test-amd” and “pr-test-amd-rocm720” fails before running a single test，错误为 Error: total_partitions (3) must be >= standalone files (7)。根因是 run_suite.py 在 CI 未提供预计算分区计划时，为每个 standalone 文件预留一个完整分区，一旦 standalone 文件数超过 --total-partitions 就退出；AMD 车道硬编码 --total-partitions 3，而 #33725 和 #33775 把 2-gpu 套件的 standalone 文件增加到 7 个。此前关闭的 #33879 通过提高硬编码计数来修复，但 PR 认为那只是推迟问题且每个额外 AMD 分区都是串行 2-GPU 任务，成本高。PR 还发现同一守卫已静默消灭 AMD 车道的参数化测试覆盖：当 standalone 文件恰好 3 个时 parametrized_partitions 为 0。因此选择移除耦合而非继续加分区。

实现拆解

1. 新增分区工具函数：在 python/sglang/multimodal_gen/test/partitioning.py 中新增 assign_partition(items, partition_id, num_partitions)，其内部调用既有确定性 LPT 算法 partition_items_by_lpt()，返回当前分片拥有的 items 切片；分片越界或分区数为 0 时返回空列表。这是整个方案的最小公共原语，预计算计划路径与本地计算路径共用同一 LPT 逻辑，避免两套代码漂移。
2. run_suite.py 本地分片构建与统一执行路径：新增 build_local_partition_assignment(suite, partition_id, total_partitions)，将参数化用例（_get_dynamic_suite_cases）和 STANDALONE_FILES[suite] 统一构造成 PartitionItem 列表，按估算耗时通过 assign_partition 分片；删除旧的基于索引的 auto_partition() 与已死的 _get_standalone_file()。_run_dynamic_suite() 现在根据是否传入 --partition-plan-json

选择 parse_partition_plan() 或 build_local_partition_assignment()，随后统一交给新提取的 _run_partition_assignment() 执行，原守卫 parametrized_partitions < 0 被移除。

3. 修复失败用例吞掉 standalone 文件的问题：旧代码在参数化 pytest 失败且未传 --continue-on-error 时提前 return，导致同分片 standalone 文件不再执行；新 _run_partition_assignment 只记录 exit_code，继续执行 standalone 文件。这是把 standalone 文件与用例混排后必须处理的回归（AMD 1-GPU 车道上的 test_generate_zimage_turbo_cli.py 此前独占分片、不受影响，且 AMD 车道无 coverage 检查，跳过将是静默的）。
4. 配套调整：gen_diffusion_ci_outputs.py 改为直接从 partitioning 模块导入 PartitionItem 和 partition_items_by_lpt（因 run_suite 不再 re-export）；两个 AMD workflow 只更新 stale 注释，分区计数不变；新增 test/unit/test_suite_partitioning.py，覆盖任意分片数下每用例和 standalone 文件恰好调度一次、越界分片返回空、以及失败用例不跳过同分片 standalone 文件的回归场景。

关键文件：

- python/sglang/multimodal_gen/test/run_suite.py（模块 测试调度；类别 test；类型 core-logic；符号 auto_partition, build_local_partition_assignment, _get_standalone_file, _run_partition_assignment）：核心改动文件：新增 build_local_partition_assignment()，删除旧 auto_partition/_get_standalone_file 与 total_partitions >= standalone files 守卫，提取统一的 _run_partition_assignment() 执行路径，并修复失败用例跳过同分片 standalone 文件的问题。
- python/sglang/multimodal_gen/test/partitioning.py（模块 分区算法；类别 test；类型 core-logic；符号 assign_partition）：新增 assign_partition() 原语，包装确定性 LPT 切片逻辑，供 build_local_partition_assignment 与预计算计划共用，是方案的最小公分母。
- python/sglang/multimodal_gen/test/unit/test_suite_partitioning.py（模块 分区测试；类别 test；类型 test-coverage；符号 _items, _expected_work, test_assign_partition_covers_every_item_once, test_assign_partition_is_empty_outside_the_shard_range）：新增 22 个测试用例，锁定核心不变量：任意分片数下整个套件恰好调度一次、越界分片为空、失败用例不跳过同分片 standalone 文件。

关键符号：assign_partition, build_local_partition_assignment, _run_partition_assignment, partition_items_by_lpt

关键源码片段

python/sglang/multimodal_gen/test/run_suite.py

核心改动文件：新增 build_local_partition_assignment()，删除旧 auto_partition/_get_standalone_file 与 total_partitions >= standalone files 守卫，提取统一的 _run_partition_assignment() 执行路径，并修复失败用例跳过同分片 standalone 文件的问题。

```
# python/sglang/multimodal_gen/test/run_suite.py
# 本地分片构建：CI 未提供预计算分区计划（AMD 车道硬编码 --total-partitions）时，
```

```
# 将参数化用例和 standalone 文件混合后按估算耗时做 LPT 均衡，
# 而不是让每个 standalone 文件独占一个分片。
# 这样分片计数不再需要 >= standalone 文件数，
# 也就消除了之前 total_partitions (3) must be >= standalone files (7) 的 abort。
def build_local_partition_assignment(
    suite: str,
    partition_id: int,
    total_partitions: int,
) -> PartitionAssignment:
    """Assign this shard's work when CI did not precompute a partition plan.

    Lanes with a hardcoded ``--total-partitions`` (the AMD ones) cannot give
    every standalone file a shard of its own, so standalone files are LPT
    balanced together with the parametrized cases instead.
    """
    # 把两类工作统一建模成 PartitionItem，
    # 后续 assign_partition 的 LPT 切片对它们一视同仁
    items = [
        PartitionItem(kind="case", item_id=case.id, est_time=get_case_est_time(case.id))
        for case in _get_dynamic_suite_cases(suite)
    ]
    for standalone_file in STANDALONE_FILES.get(suite, []):
        items.append(
            PartitionItem(
                kind="standalone",
                item_id=standalone_file,
                est_time=get_standalone_file_est_time(suite, standalone_file)[0],
            )
        )

    # 每个分片独立调用 assign_partition，
    # 由于 LPT 是确定性的，各分片合起来恰好覆盖全部 item 一次
    my_items = assign_partition(items, partition_id, total_partitions)
    return PartitionAssignment(
        case_ids=[item.item_id for item in my_items if item.kind == "case"],
        standalone_files=[
            item.item_id for item in my_items if item.kind == "standalone"
        ],
    )


def _run_partition_assignment(
    args, target_dir: Path, assignment: PartitionAssignment
) -> int:
    # 统一执行入口：无论是预计算计划还是本地构建的 assignment 都走这里。
    # 旧代码在参数化用例失败且未传 --continue-on-error 时提前 return，
    # 会导致同分片的 standalone 文件被静默跳过；
    # 现在只记录 exit_code，standalone 文件仍会继续执行。
    ...
```

```
    if assignment.case_ids:
        ...
        exit_code, new_executed_cases, new_case_results = run_pytest(
            suite_files, filter_expr=filter_expr, junit_xml_path=junit_xml_path,
        )
        _merge_execution_results(executed_cases, case_results, new_executed_cases, new_case_
        results)
        # A failing case must not swallow this shard's standalone files: they
        # are separate pytest runs, and they only share a shard because the
        # shard count is fixed. --continue-on-error still decides whether a
        # failing standalone file stops the ones queued behind it.
        if exit_code != 0 and overall_exit_code == 0:
            overall_exit_code = exit_code

    if assignment.standalone_files:
        ...
```

python/sglang/multimodal_gen/test/partitioning.py

新增 assign_partition() 原语，包装确定性 LPT 切片逻辑，供 build_local_partition_assignment 与预计算计划共用，是方案的最小公分母。

```
# python/sglang/multimodal_gen/test/partitioning.py
# 在既有 partition_items_by_lpt 之上增加单分片视角：
# 每个分片只需拿到自己那一份 items，而不需要看到全局结果。
# 因为 LPT 是确定性的（按 (-est_time, kind, item_id) 排序后贪心放入最轻的分片），
# 只要所有分片传入相同的 items 和 num_partitions，
# 它们合起来就恰好覆盖整个套件一次，且无需分片间通信。
def assign_partition(
    items: list[PartitionItem], partition_id: int, num_partitions: int
) -> list[PartitionItem]:
    """Return the LPT slice of ``items`` owned by ``partition_id``.

    The LPT pass is deterministic, so shards that each call this with the same
    item list cover the list exactly once between them.
    """
    partitions = partition_items_by_lpt(items, num_partitions)
    # 越界（partition_id 超范围或 num_partitions 为 0）时返回空列表，
    # 调用方按空 assignment 处理，避免下标异常
    if partition_id < 0 or partition_id >= len(partitions):
        return []
    return partitions[partition_id]
```

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

新增 22 个测试用例，锁定核心不变量：任意分片数下整个套件恰好调度一次、越界分片为空、失败用例不跳过同分片 standalone 文件。

```
# python/sglang/multimodal_gen/test/unit/test_suite_partitioning.py
# 核心不变量测试：对每个 diffusion 套件、任意分片数（1/2/3/4/8），
# 把各分片的 build_local_partition_assignment 结果合并后，
```

```
# 必须恰好等于该套件的全部参数化用例 + 全部 standalone 文件。
# 这正是之前 AMD 2-GPU abort（standalone 文件数 > 分片数）场景的回归保护。
@pytest.mark.parametrize("suite", sorted(PARAMETRIZED_CASE_GROUPS))
@pytest.mark.parametrize("total_partitions", [1, 2, 3, 4, 8])
def test_suite_is_fully_scheduled_for_any_shard_count(suite, total_partitions):
    # 期望工作 = 套件里所有参数化用例 id + 所有 standalone 文件
    expected_case_ids, expected_standalone_files = _expected_work(suite)

    scheduled_case_ids: list[str] = []
    scheduled_standalone_files: list[str] = []
    for partition_id in range(total_partitions):
        assignment = build_local_partition_assignment(
            suite=suite,
            partition_id=partition_id,
            total_partitions=total_partitions,
        )
        scheduled_case_ids.extend(assignment.case_ids)
        scheduled_standalone_files.extend(assignment.standalone_files)

    # More standalone files than shards used to abort the run; they now share
    # shards with the parametrized cases instead.
    assert sorted(scheduled_case_ids) == sorted(expected_case_ids)
    assert sorted(scheduled_standalone_files) == sorted(expected_standalone_files)


# fast-fail 回归测试：参数化用例失败时，同分片的 standalone 文件必须仍被执行。
# 旧逻辑在用例失败且未传 --continue-on-error 时提前 return，
# AMD 车道没有 coverage 检查，跳过会是静默的。
def test_failing_cases_do_not_skip_the_shards_standalone_files(monkeypatch, tmp_path):
    """Standalone files used to own a shard, so cases could not block them."""
    standalone_rel = "../single_test_file/test_disagg_server.py"
    executed_standalone = []

    # 让参数化 pytest 返回失败（exit code 1），并记录 standalone 文件是否执行
    def fake_run_standalone_file(suite, rel, target_dir, extra_filter=None):
        executed_standalone.append(rel)
        key = f"standalone:{rel}"
        return 0, [key], {key: "pass"}, {"used_fallback_estimate": False}

    monkeypatch.setattr(
        run_suite, "run_pytest", lambda *a, **k: (1, ["a_case"], {"a_case": "fail"})
    )
    monkeypatch.setattr(run_suite, "_run_standalone_file", fake_run_standalone_file)
    monkeypatch.setattr(
        run_suite, "_get_parametrized_files_for_case_ids", lambda *a, **k: ["a_file.py"]
    )
    monkeypatch.setattr(run_suite, "write_execution_report", lambda **kwargs: "")

    args = SimpleNamespace(
```

```
        suite="2-gpu",
        partition_id=0,
        total_partitions=2,
        filter=None,
        continue_on_error=False,
    )
    exit_code = run_suite._run_partition_assignment(
        args,
        tmp_path,
        PartitionAssignment(case_ids=["a_case"], standalone_files=[standalone_rel]),
    )

    # 用例失败但 standalone 文件仍被执行，且整体 exit code 保持失败
    assert executed_standalone == [standalone_rel]
    assert exit_code == 1
```

评论区精华

主要 review 讨论集中在 bingxche 提出的三个问题：

- fast-fail 跳过 standalone 文件问题：bingxche 指出把 standalone 文件与用例混排后，参数化用例失败会导致同分片 standalone 文件被跳过。作者承认 1-gpu 是真正的回归点（HIP 上唯一的 standalone 文件 test_generate_zimage_turbo_cli.py 若被 flux_image_t2i/joyai_image_edit_ti2i 拖累会每次都被跳过且无 coverage 检查），最终选择直接删除提前 return 而非加条件守卫，并新增 test_failing_cases_do_not_skip_the_shards_standalone_files 锁定。
- 未读字段问题：estimated_time 与 missing_standalone_estimates 在本地构建的 assignment 中并没有被使用（_run_partition_assignment 运行时会重算），作者删除了它们及 used_fallback_estimate 管道。
- 变更范围界定：作者新增 “Which lanes change behavior” 小节，明确 AMD 2-GPU（由 abort 变为 3 个均衡分片）、AMD 1-GPU（3 个参数化分片 + 1 个独立分片变为 4 个混合分片）、CUDA multimodal-gen-test-1-5090（结果不变）、CUDA bcg-diffusion（仅报告 is_standalone 标志不同）的行为变化，其余车道字节级不变。kangwangamd 的离线验证确认旧逻辑在 7 standalone 文件 + 3 分区时计算 parametrized_partitions = -4 会 abort，新逻辑 LPT 均衡后 3 个分片各 4 项（约 805s/755s/755s），每个 item 恰好调度一次。
- 参数化用例失败会静默跳过同分片 standalone 文件 (correctness): 作者删除提前 return，改为记录 exit_code 后继续执行 standalone 文件；同时移除基于 not assignment.standalone_files 的条件守卫以避免重复 report 调用，并新增 test_failing_cases_do_not_skip_the_shards_standalone_files 回归测试。
- 本地 assignment 中未读字段的清理 (design): 作者在 3eb5538a 提交中删除这两个字段以及只服务于它们的 used_fallback_estimate 管道。
- 变更影响范围界定 (documentation): PR body 已补充完整的逐车道行为变化清单，并说明 NPU、MUSA、1-gpu-b200、unit 等路径不受影响。

风险与影响

- 风险：
  1. CUDA 车道行为回归风险：bcg-diffusion 套件的报告 is_standalone 标志从 false 变为 true（因为本地构建的 assignment 现在显式含 standalone 文件），PR 称没有消费者消费该 job 的报告，但这是一个未被验证的断言。
  2. AMD 1-GPU 车道组成变化：3 个参数化分片 + 1 个 standalone 分片变为 4 个混合分片，test_generate_zimage_turbo_cli.py 现在与用例共享分片、不再独占；虽然修复了失败时不跳过它，但它现在可能因同分片用例失败而整体 job 失败（此前 standalone 分片独立判定）。
  3. 估算时间失配影响均衡质量：standalone 文件使用 get_standalone_file_est_time() 的估算值，若估算不准（fallback 默认值），LPT 均衡的实际运行时长可能偏斜；PR body 给出的 2642s/2597s/2652s 是基于估算的。
  4. 预计算计划路径与本地路径的统一风险：_run_partition_assignment 提取后，计划驱动的 CUDA/NPU 车道也走同一执行函数，虽然 PR 声称字节级不变，但任何该函数的改动现在同时影响两条路径。
  5. 预存在的 diffusion_case_parser.py bug 未修复：scripts/ci/utils/diffusion/diffusion_case_parser.py AST 解析 STANDALONE_FILES 的源已经移到 server/gpu_cases.py，导致 CUDA 分区计划自 6 月起未调度任何 standalone 测试，verify_diffusion_coverage.py 因同源 bug 未察觉。PR 有意不修，但这是悬挂的技术债。 - 影响：影响面主要是 CI 基础设施而非运行时服务代码。直接效果：AMD 两个工作流的 2-GPU diffusion 车道从必然 abort 变为可正常执行测试（此前 #33725/#33775 后每个 AMD PR 的该矩阵都 red）；AMD 1-GPU 车道的测试组成被重组；CUDA/NPU 车道理论上行为不变。对团队的影响是 AMD 车道恢复有效信号和参数化覆盖（此前 3 standalone 文件时参数化分片为 0，参数化测试静默不跑）。该方案让分片计数与 standalone 文件数量解耦，未来新增 standalone 文件不再需要调整 workflow 矩阵。影响程度中等：涉及所有 AMD PR CI 门禁与 diffusion 测试调度逻辑。 - 风险标记：CI 门禁行为变更，存在预存在未修复 bug，估算驱动均衡可能偏斜，统一执行路径影响面扩大

关联脉络

- PR #33879 [diffusion] align AMD 2-GPU test partitions: 之前关闭的同类修复尝试：通过把 AMD 2-GPU 分区数从 3 提高到 7（每个 standalone 文件一个分区）来绕过 abort。本 PR 明确拒绝该方案并改走解耦路线，是直接的前置讨论对象。
- PR #33725 （推测）新增 2-GPU standalone 测试文件：PR body 提到 #33725 和 #33775 使 STANDALONE_FILES["2-gpu"] 增长到 7 个，直接触发了本次 abort；虽然历史 PR 列表中未收录条目，但这是根因链的一环。
- PR #33775 （推测）新增 2-GPU standalone 测试文件：同上，与 #33725 一起把 2-gpu 套件 standalone 文件推到 7 个，触发旧守卫。
- PR #24630 （推测）将 STANDALONE_FILES 移入 server/gpu_cases.py: PR body 指出 diffusion_case_parser.py 的 AST 解析仍在旧文件查找 STANDALONE_FILES，导致 CUDA 分区计划自 6 月起没有调度任何 standalone 测试，该迁移是预存在 bug 的来源。