PR #25263 完整报告

sgl-project/sglang

ci: dynamic partition + LPT from live sglang-ci-stats model

合并时间：2026-05-14 17:35

原文链接：http://prhub.com.cn/sgl-project/sglang/pull/25263

执行摘要

- 一句话：CI 测试分区基于实时性能数据动态调整
- 推荐动作：值得精读，因为它实现了一个实用的动态 CI 分区策略，并展示了如何安全地集成外部数据源。设计选择（SHA 固定、逐步回退、基于 OLS 的性能建模）对于构建可靠的自动化系统有借鉴意义。

功能与动机

PR body 指出：'Dispatch sizing and runtime LPT bucketing read live est (per-file p90) and (coeff, bias) OLS fit from sgl-project/sglang-ci-stats's model.json instead of the static in-source est_time literals.' 目标是让分区大小根据实际的测试执行时间动态调整，减少超时和资源浪费。

实现拆解

1. 数据源抽象：在 scripts/ci/utils/compute_partitions.py 中新增 load_partition_model 函数，从外部 model.json 加载实时估计值和 OLS 系数（coeff, bias），并在文件不可读或非字典时返回 None，保证下游回退。
2. 目标计算与分片数公式：新增 load_run_timeouts 从 pr-test.yml 的工作流定义解析阶段超时，并通过 per_shard_target_seconds 计算每个分片的预算目标（0.75 * 超时，逆 LPT 4/3 最坏边界）。修改 compute_partitions 使用实时数据计算总大小：size = ceil(coeff * total / (target - bias))，并移除硬限制 MAX_PARTITION_SECONDS。
3. 运行时 LPT 集成：在 test/run_suite.py 中新增 load_live_est 函数，在测试运行前读取相同的 model.json，并将实时估计值传递给 auto_partition。
4. CI 注册表升级：在 python/sglang/test/ci/ci_register.py 中修改 auto_partition，接受可选的 live_est 字典，优先使用实时估计，缺失时回退到静态 est_time。
5. 工作流串联：在 .github/workflows/_pr-test-check-changes.yml 中添加步骤，固定 sglang-ci-stats 的 SHA 并获取模型 JSON；在 _pr-test-stage.yml 中添加相同的获取步骤，并将文件路径传递给 run_suite.py；在 pr-test.yml 中为各阶段补全 run_timeout_minutes 参数。同时调整 max_parallel 从 size//4 改为 size//3，并更新 DeepSeek V4 FP4 测试的 est_time 从 1800 降低到 900 秒。

关键文件：

- scripts/ci/utils/compute_partitions.py（模块 分区计算；类别 infra；类型 infrastructure；符号 load_run_timeouts, per_shard_target_seconds, load_partition_model, compute_partitions）：核心实现，新增 live 数据加载和动态分区计算逻辑。
- test/run_suite.py（模块 测试运行；类别 test；类型 test-coverage；符号 load_live_est）：运行时 LPT 集成，新增 load_live_est 函数。
- python/sglang/test/ci/ci_register.py（模块 CI 注册；类别 test；类型 test-coverage；符号 auto_partition, est_of）：auto_partition 函数扩展支持 live_est 参数。
- .github/workflows/_pr-test-check-changes.yml（模块 CI 工作流；类别 infra；类型 infrastructure）：添加 SHA 固定和模型获取步骤，作为数据源入口。
- .github/workflows/_pr-test-stage.yml（模块 CI 工作流；类别 infra；类型 infrastructure）：在测试阶段添加模型获取，并将文件路径传递给 run_suite.py。
- .github/workflows/pr-test.yml（模块 CI 工作流；类别 infra；类型 infrastructure）：为各阶段补全 run_timeout_minutes 必填参数。
- test/registered/dsv4/test_deepseek_v4_flash_fp4_megamoe_b200.py（模块 DeepSeek 测试；类别 test；类型 test-coverage）：调整 est_time 从 1800 到 900，反映实际执行时间更短。

关键符号：load_run_timeouts, per_shard_target_seconds, load_partition_model, compute_partitions, load_live_est, auto_partition, est_of

关键源码片段

scripts/ci/utils/compute_partitions.py

核心实现，新增 live 数据加载和动态分区计算逻辑。

```
# compute_partitions.py — 实时分区模型加载与分片数量计算

def load_partition_model(path):
    """从 sglang-ci-stats 的 `model.json` 加载分区模型，
    返回 `None` 当文件缺失或格式异常时（调用者回退到静态估计）。"""
    if not path or not os.path.exists(path):
        return None
    try:
        with open(path) as f:
            data = json.load(f)
        # 确保顶层是字典，避免后续 .get() 引发 AttributeError
        return data if isinstance(data, dict) else None
    except (OSError, json.JSONDecodeError):
        return None


def per_shard_target_seconds(suite: str, run_timeouts: dict) -> float:
    """每个分片的预算时间 = 0.75 × 阶段超时（分钟→秒）。
    0.75 是 LPT 4/3 最坏情形的倒数，确保最不平衡的分片刚好填满超时。"""
    return 0.75 * run_timeouts[suite] * 60
```

```
def compute_partitions(tests, run_timeouts, partition_model=None, full_parallel=False):
    # ... ( 前面的初始化 )
    est_table = (partition_model or {}).get("est", {})
    fit_table = (partition_model or {}).get("fit", {})
    for suite, group in suite_tests.items():
        live_est = est_table.get(suite, {})
        total = 0.0
        for t in group:
            # 优先使用实时估计，缺失时回退到静态 est_time
            total += float(live_est.get(t.relpath, t.est_time))
        fit = fit_table.get(suite, {})
        coeff = fit.get("coeff", 1.0) # 安全默认值
        bias = fit.get("bias", 0.0)
        target = per_shard_target_seconds(suite, run_timeouts)
        # 计算分片数，向上取整
        size = max(1, math.ceil(coeff * total / (target - bias)))
        # ... 其余逻辑
```

test/run_suite.py

运行时 LPT 集成，新增 load_live_est 函数。

```
# run_suite.py — 运行时加载实时估计值

def load_live_est(
    partition_model_file: Optional[str], suite: str, repo_root: str
) -> Optional[Dict[str, float]]:
    """从 `model.json est[suite]` 构建 `CIRegistry.filename -> est seconds` 映射;
    任何缺失均返回 `None`，由调用者决定回退到静态 `est_time`。"""
    if not partition_model_file or not os.path.exists(partition_model_file):
        return None
    try:
        with open(partition_model_file) as f:
            partition_model = json.load(f)
    except (OSError, json.JSONDecodeError):
        return None
    if not isinstance(partition_model, dict):
        return None
    suite_est = partition_model.get("est", {}).get(suite)
    # 确保 suite_est 是字典，防止 JSON 结构中值为列表或其他类型
    if not isinstance(suite_est, dict) or not suite_est:
        return None
    return {
        os.path.join(repo_root, relpath): float(elapsed)
        for relpath, elapsed in suite_est.items()
    }
```

python/sglang/test/ci/ci_register.py

auto_partition 函数扩展支持 live_est 参数。

```
# ci_register.py — LPT 分区函数支持实时估计覆盖

def auto_partition(
    files: List[CIRegistry],
    rank: int,
    size: int,
    live_est: Optional[dict] = None,
) -> List[CIRegistry]:
    """使用贪心算法（LPT 启发式）将文件分区，返回指定 rank 的分片。
    `live_est` 可选，提供 `filename -> est seconds` 覆盖；缺失文件回退到静态 `est_time`。"""
    if not files or size <= 0:
        return []

    def est_of(f: CIRegistry) -> float:
        if live_est is not None and f.filename in live_est:
            return live_est[f.filename]
        return f.est_time

    # 按估计时间降序排序，文件名作为次级键以确保确定性
    sorted_files = sorted(files, key=lambda f: (-est_of(f), f.filename))

    partitions: List[List[CIRegistry]] = [[] for _ in range(size)]
    partition_sums = [0.0] * size

    for file in sorted_files:
        min_sum_idx = min(range(size), key=partition_sums.__getitem__)
        partitions[min_sum_idx].append(file)
        partition_sums[min_sum_idx] += est_of(file)

    if rank < size:
        return partitions[rank]
    return []
```

评论区精华

在 review 中，gemini-code-assist[bot] 提出了三条关于数据安全性的评论：

- 在 load_partition_model 中未对 JSON 顶级类型做检查，若为数组会导致 AttributeError，建议添加 isinstance(data, dict) 保护。
- 在 compute_partitions 中访问 fit["coeff"] 和 fit["bias"] 可能引发 KeyError，建议使用 fit.get("coeff", 1.0) 等默认值。
- 在 load_live_est 中未验证 suite_est 是否为字典，建议添加 isinstance(suite_est, dict) 检查。 所有评论均被作者采纳，从最终代码看，相应的保护已加入。
- 外部数据解析缺少类型检查 (correctness): 作者采纳建议，在 load_partition_model 中添加 return data if isinstance(data, dict) else None，在 load_live_est 中添加 if not isinstance(suite_est, dict) or not suite_est: return None。

- 字典访问使用 .get() 避免 KeyError (correctness): 作者修改代码，使用 fit.get("coeff", 1.0) 和 fit.get("bias", 0.0) 替代直接下标访问。

风险与影响

- 风险：
  - 外部依赖风险：分区数据依赖 sgl-project/sglang-ci-stats 的 model.json，若该仓库不可达或返回非预期格式，则回退到静态估计，可能导致分片不均但不会中断 CI。SHA 固定降低了跨分片不一致的风险，但固定版本过旧可能使用过时数据。
  - 参数计算风险：每个分片目标时间假设 LPT 最坏情况比例为 4/3，若实际分布偏差较大，部分分片仍可能接近超时。size 公式中的 target - bias 若为负数会引发异常（但作者添加了防御性检查）。
  - 配置一致性：run_timeout_minutes 参数从工作流中提取，若工作流重命名或路径变化，load_run_timeouts 可能返回空字典并触发 RuntimeError。
  - 影响：仅影响 CI 基础设施，不触及产品代码。所有 CI 运行将使用动态分区，预期提升并行效率，减少测试等待和重试。对开发者表现为更快的 CI 反馈。工作流配置需要保持与脚本的同步（如 run_timeout_minutes 必填）。
  - 风险标记：依赖外部数据源，SHA 固定可能过时，回退降级可能导致性能偏差，新参数未充分测试

关联脉络

- PR #25255 ci: read est_time from sglang-ci-stats instead of scraping CI logs: 该 PR 将 est_time 数据源集中到 sglang-ci-stats 的 model.json，为本 PR 的实时数据消费奠定了基础。
- PR #25238 [CI] Bundle check-changes outputs + caller inputs into 2 JSON inputs: 该 PR 简化了 CI 工作流的输入结构，使本 PR 更容易传递新增的 partition_model_sha 等参数。