PR #33098 完整报告

sgl-project/sglang

Fix DSpark and DP/EP

合并时间：2026-08-04 15:35

原文链接：http://prhub.com.cn/sgl-project/sglang/pull/33098

执行摘要

- 一句话：修复 DSpark 草案批次元数据缺失导致的启动与首请求崩溃
- 推荐动作：值得精读。重点看三处：一是 _fill_dp_moe_sync_metadata 中 " 缩放计数 vs 未缩放计数 " 的语义划分（CUDA graph 准入用原始请求数、EP accounting 用非填充 token 数），这是多机制交错下容易踩坑的契约；二是 enable_num_token_non_padded 环境开关的渐进式迁移模式；三是测试用 __new__ 绕过构造器直接测方法契约的技巧。整体改动小而精准，是理解 SGLang 投机解码与 DP/EP 组合细节的好案例。

功能与动机

PR body 明确指出：DSpark 的 draft ForwardBatch 遗漏了 CUDA graph admission 和 EP token accounting 所需的 metadata，造成启动与首个请求崩溃（TypeError/original_global_num_tokens_cpu 缺失）。外部用户 junliu-mde 在 v0.5.16 基线上复现同一问题并确认补丁有效，说明这是真实且影响面明确的回归。

实现拆解

1. 变更入口：python/sglang/srt/speculative/dspark_components/dspark_draft.py 中 DraftBlockProposer._fill_dp_moe_sync_metadata，该方法在 DSpark draft 批次构造后填充 DP MoE 同步元数据。
2. 核心改动：新增三处赋值——将未缩放的 batch.global_num_tokens 原样存入 forward_batch.original_global_num_tokens_cpu（CUDA graph 准入需要原始请求数而非放大后的 token 数）；用 forward_batch.input_ids.numel() 计算非填充 token 数，在 enable_num_token_non_padded() 环境开关开启时写入 num_token_non_padded（torch.int32 GPU 张量），并始终保存 num_token_non_padded_cpu；原有 spec_scale_global_num_tokens 缩放、CPU/GPU 双写、can_run_dp_cuda_graph 透传逻辑保持不变。同时从 forward_batch_info 新增导入 enable_num_token_non_padded。
3. 测试配套：test/registered/spec/dspark/test_dspark_dp_tier.py 新增 TestDraftDpSyncMetadata，用 DraftBlockProposer.__new__ 绕过构造器、SimpleNamespace 模拟 batch，patch 环境开关，断言未缩放计数、缩放计数（每请求 6 个 token 时 [1,3,0,2]→[6,18,0,12]）、num_token_non_padded 的值与 dtype、DP CUDA graph 标志共 6 项。
4. 验证矩阵：PR 作者在 4xH200 上验证 GSM8K 96.0%；reviewer 在 4xGB300 上验证 27/27 tier CUDA graph 完成、GSM8K 96.66%、c256 吞吐 26,868 tok/s；外部用户

junliu-mde 在 2xB300 上验证无崩溃。无配置、schema 或部署配套改动。

关键文件：

- python/sglang/srt/speculative/dspark_components/dspark_draft.py（模块 投机解码；类别 source；类型 core-logic；符号 _fill_dp_moe_sync_metadata）：核心修复文件：在 DSpark draft ForwardBatch 构造后补齐 original_global_num_tokens_cpu 与非填充 token 计数，解决 CUDA graph admission 与 EP token accounting 在 DP/EP 组合下的崩溃。
- test/registered/spec/dspark/test_dspark_dp_tier.py（模块 回归测试；类别 test；类型 test-coverage；符号 TestDraftDpSyncMetadata, test_preserves_unscaled_request_counts_for_cuda_graph_admission）：新增回归测试 TestDraftDpSyncMetadata，验证未缩放计数在 CUDA graph 准入路径中得以保留、非填充计数与缩放计数正确写入，防止后续演进重新踩坑。

关键符号：_fill_dp_moe_sync_metadata, test_preserves_unscaled_request_counts_for_cuda_graph_admission

关键源码片段

python/sglang/srt/speculative/dspark_components/dspark_draft.py

核心修复文件：在 DSpark draft ForwardBatch 构造后补齐 original_global_num_tokens_cpu 与非填充 token 计数，解决 CUDA graph admission 与 EP token accounting 在 DP/EP 组合下的崩溃。

```
def _fill_dp_moe_sync_metadata(
    self, forward_batch: ForwardBatch, batch: ScheduleBatch
) -> None:
    """填充 DP MoE 同步所需的 draft ForwardBatch 元数据。

    DSpark 的 draft 批次在 DP/EP 组合下必须携带三类信息：
    1. original_global_num_tokens_cpu：未缩放的真实请求级 token 计数，
       CUDA graph 准入依赖它判断各 DP rank 的请求分布；
    2. num_token_non_padded / num_token_non_padded_cpu：非填充 token 数，
       EP token accounting（MoE 专家并行）按它计算负载；
    3. 缩放后的 global_num_tokens_cpu / gpu 系列：按 draft block 规格
       （每请求 gamma 个 token）放大后的计数，供注意力与采样使用。
    缺失这些字段会导致启动期或首个请求时崩溃。
    """
    if not self._dp_moe_sync or batch.global_num_tokens is None:
        return

    # 先按 draft block 规格放大计数（例如每请求 6 个 draft token）
    gnt, gnt_logprob = spec_scale_global_num_tokens(
        self._draft_block_spec_info,
        batch.global_num_tokens,
        batch.global_num_tokens_for_logprob,
    )
```

```
    device = self.draft_model_runner.device

    # 保留未缩放计数：CUDA graph 准入需要原始请求数，
    # 而不是放大后的 token 数（后者会让图键跨 rank 错位）
    forward_batch.original_global_num_tokens_cpu = batch.global_num_tokens

    # 非填充 token 数由实际 input_ids 长度决定，
    # 受 enable_num_token_non_padded 环境开关控制（渐进式迁移）
    num_tokens = forward_batch.input_ids.numel()
    if enable_num_token_non_padded():
        forward_batch.num_token_non_padded = torch.tensor(
            num_tokens, dtype=torch.int32, device=device
        )
    forward_batch.num_token_non_padded_cpu = num_tokens

    # 缩放后的计数同时写 CPU 与 GPU 两个版本，供不同消费路径使用
    forward_batch.global_num_tokens_cpu = gnt
    forward_batch.global_num_tokens_for_logprob_cpu = gnt_logprob
    forward_batch.global_num_tokens_gpu = torch.tensor(gnt, dtype=torch.int64).to(
        device, non_blocking=True
    )
    forward_batch.global_num_tokens_for_logprob_gpu = torch.tensor(
        gnt_logprob, dtype=torch.int64
    ).to(device, non_blocking=True)
    forward_batch.can_run_dp_cuda_graph = batch.can_run_dp_cuda_graph
```

test/registered/spec/dspark/test_dspark_dp_tier.py

新增回归测试 TestDraftDpSyncMetadata，验证未缩放计数在 CUDA graph 准入路径中得以保留、非填充计数与缩放计数正确写入，防止后续演进重新踩坑。

```
class TestDraftDpSyncMetadata(CustomTestCase):
    def test_preserves_unscaled_request_counts_for_cuda_graph_admission(self):
        # 用 __new__ 绕过 DraftBlockProposer 的构造器，
        # 只测试 _fill_dp_moe_sync_metadata 的元数据契约
        proposer = DraftBlockProposer.__new__(DraftBlockProposer)
        proposer._dp_moe_sync = True
        proposer._draft_block_spec_info = SimpleNamespace(
            num_tokens_per_req=6, # 每个 draft block 6 个 token
            num_tokens_for_logprob_per_req=1,
        )
        proposer.draft_model_runner = SimpleNamespace(device="cpu")

        forward_batch = SimpleNamespace(input_ids=torch.arange(6))
        batch = SimpleNamespace(
            global_num_tokens=[1, 3, 0, 2], # 各 DP rank 请求数不等
            global_num_tokens_for_logprob=[1, 3, 0, 2],
            can_run_dp_cuda_graph=True,
        )
```

```
with patch(
    "sglang.srt.speculative.dspark_components.dspark_draft.enable_num_token_non_
    padded",
    return_value=True,
):
    proposer._fill_dp_moe_sync_metadata(forward_batch, batch)

# 未缩放计数必须原样保留（CUDA graph 准入依据）
self.assertEqual(forward_batch.original_global_num_tokens_cpu, [1, 3, 0, 2])
# 缩放后每请求 6 个 draft token
self.assertEqual(forward_batch.global_num_tokens_cpu, [6, 18, 0, 12])
# 非填充 token 数 = input_ids 实际长度
self.assertEqual(forward_batch.num_token_non_padded.item(), 6)
self.assertEqual(forward_batch.num_token_non_padded.dtype, torch.int32)
self.assertEqual(forward_batch.num_token_non_padded_cpu, 6)
self.assertTrue(forward_batch.can_run_dp_cuda_graph)
```

评论区精华

核心讨论围绕修复有效性与职责边界：

- JustinTong0323（approver）在 v0.5.16 精确基线上 cherry-pick 三个修复 commit 复现并验证："the first request previously crashed because original_global_num_tokens_cpu was missing"，修复后 "The target and draft CUDA graphs both completed all 27/27 tiers"，GSM8K 96.66%，c256 吞吐 +54.67% vs TP4+DSpark，确认该补丁让 DSpark+DPA 成为最强吞吐配置。
- junliu-mde（外部用户）两次确认在 2x B300 上补丁有效："Both target and draft CUDA graph capture completed with backend=full"，并发请求 HTTP 200，"no TypeError, traceback, or CUDA error"。
- Phoenix3334 在合并后补充了职责边界说明：#34410 处理 DSpark PD-decode draft-input handoff（SpecInput 与 overlap relay 状态），"#33098 fixed DSpark DP-attention draft ForwardBatch metadata, while #34410 covers the separate disaggregation handoff invariant"，两者互补而非重复。
- DSpark+DP 启动崩溃的根因确认与修复验证 (correctness): 修复有效，且让 DSpark+DPA 成为最强吞吐配置；批准合并。
- 外部用户 B300 环境复现确认 (other): 修复在 B300 环境同样有效，跨平台适用。
- 与 #34410 的职责边界划分 (design): 两个修复覆盖不同路径，需配套合入以补齐 DSpark 完整兼容面。

风险与影响

- 风险：
  1. 行为依赖环境开关：num_token_non_padded 张量只在 enable_num_token_non_padded() 返回真时写入，若默认关闭，依赖该字段的 EP accounting 路径可能仍走旧行为；本 PR 未调整开关默认值。

2. 关注路径在核心draft循环：改动位于_fill_dp_moe_sync_metadata，虽然仅新增字段、不改变既有控制流，但该函数在每次 DSpark draft forward 前执行，回归影响会被放大。
3. 测试覆盖局限：新增测试为纯 mock 单测，未覆盖多 rank 真实 DP 通信、CUDA graph capture 或 EP 分派，真实拓扑行为主要依赖人工验证。
4. 范围边界：PR 只修复 DP attention 路径；DSpark 与 PD disaggregation 的 handoff 问题（#34410）不在本 PR 覆盖内，两个修复需配套合入。 - 影响：用户侧：DSpark + DP attention + DP LM head（含 EP）组合的用户从启动 / 首请求崩溃中恢复，该组合在 GB300 上成为最强吞吐配置（c256 26,868 tok/s，较 target-only DPA +33.55%）。系统侧：draft ForwardBatch 元数据契约补齐后，CUDA graph 准入与 EP token accounting 消费路径获得稳定输入，后续其他调度特性组合可复用同一契约。团队侧：提供了一个 " 组合特性崩溃源于中间层元数据缺失 " 的可复制调试与测试范式，mock 构造方式（__new__ + SimpleNamespace）成本低、易扩展。 - 风险标记：组合特性兼容性路径，新增元数据字段，测试为 mock 覆盖，依赖环境开关

关联脉络

- PR #33865 Fix DSpark + DeepSeek V4 prefill CP compatibility: 同一功能线：修复 DSpark 与并行 / 解码策略（CP、DP）组合时的兼容性问题，涉及 deepseek_v4.py、speculative_hook.py 等相邻路径。
- PR #29070 [DSV4] perf: Enable alt stream during BCG prefill: DeepSeek V4 性能与稳定性演进线上的前置工作，同属 DSV4 + DSpark 相关优化脉络。
- PR #34410 Follow-up PR pinning DSpark PD-decode draft-input handoff: review 评论中明确的 follow-up 集成回归 PR，处理 DSpark PD-decode 阶段的 draft-input handoff，与 #33098 互补。