PR #7438 完整报告

PaddlePaddle/FastDeploy

[BugFix] Fix real token exceeding max_batched_tokens limit

合并时间：2026-04-17 16:18

原文链接：http://prhub.com.cn/PaddlePaddle/FastDeploy/pull/7438

执行摘要

- 一句话：修复投机解码场景下调度器 token 预算计算错误，避免显存 OOM。
- 推荐动作：该 PR 值得精读，重点关注调度器预算计算的设计权衡：为何选择预减而非逐请求扣减？临时下限 512 的选取依据是什么？建议结合 review 讨论思考更优方案。

功能与动机

根据 PR body 描述，当前实际推理的 step token 数不断超过 max_batched_tokens，导致 Paddle 持续分配新显存直到 OOM。作者附带了截图显示问题现象，核心动机是修复投机解码场景下的调度器预算计算错误，防止显存溢出。

实现拆解

1. 调整 token 预算计算逻辑：在 fastdeploy/engine/sched/resource_manager_v1.py 的 schedule() 方法中，修改 token_budget 的计算方式。原逻辑直接使用 max_num_batched_tokens，现改为根据投机解码配置预留预算：token_budget = max_num_batched_tokens - max_num_seqs * tokens_per_seq，其中 tokens_per_seq 在投机解码开启时为 num_speculative_tokens + 1，否则为 1。
2. 添加临时下限保护：为避免 token_budget 变为负数，添加一行代码：token_budget = max(token_budget, min(max_num_batched_tokens, 512))，作为临时解决方案防止负预算。
3. 配套改动：本次变更仅涉及调度器核心逻辑文件，没有明显的测试、配置或部署配套改动。review 评论指出需要补充单元测试覆盖投机解码场景，但 PR 中未包含。

关键文件：

- fastdeploy/engine/sched/resource_manager_v1.py（模块 调度器；类别 source；类型 core-logic；符号 schedule）：这是 PR 唯一修改的文件，包含调度器核心预算计算逻辑，修复了投机解码场景下的 OOM 问题。

关键符号：schedule

关键源码片段

fastdeploy/engine/sched/resource_manager_v1.py

这是 PR 唯一修改的文件，包含调度器核心预算计算逻辑，修复了投机解码场景下的 OOM 问题。

```
def schedule(self, requests: List[Request]) -> Tuple[List[Request], List[Request], List[Tuple[str, str]]]:
    # ... 其他代码 ...
    scheduled_reqs: list[Request] = []
    preempted_reqs: list[Request] = []
    error_reqs: list[tuple[str, str]] = []

    # 计算每个序列在投机解码下可能消耗的token数（包括推测token和真实token）
    tokens_per_seq = (
        (self.config.speculative_config.num_speculative_tokens + 1)
        if self.config.speculative_config is not None
        else 1
    )
    # 从总预算中预留出所有最大序列数在decoding阶段可能消耗的token
    token_budget = (
        self.config.scheduler_config.max_num_batched_tokens
        - self.config.scheduler_config.max_num_seqs * tokens_per_seq
    )
    # 临时解决方案：避免token_budget变为负数，设置下限为512或max_num_batched_
    tokens中的较小值
    token_budget = max(token_budget, min(self.config.scheduler_config.max_num_batched_
    tokens, 512))

    need_abort_requests = [] # users trigger abortion
    # 首先调度RUNNING请求
    # ... 后续调度逻辑 ...
```

评论区精华

review 中围绕实现正确性和设计权衡进行了深度讨论：

- Python 三元表达式优先级 bug：PaddlePaddle-bot 指出原代码因 Python 条件表达式优先级问题，导致非投机解码场景下 token_budget 被错误设为 1，严重退化调度吞吐。结论是改用显式 if/else 语句避免歧义。
- 预算计算过于保守：Copilot 和 PaddlePaddle-bot 都提到预减 max_num_seqs * tokens_per_seq 假设所有序列同时处于 decoding 状态，过度压缩 prefill 预算，建议改为逐请求精确扣减。但最终实现保留了预减方案。
- 魔法数字和拼写错误：评论指出硬编码的 512 缺乏依据，且注释中“temperatory”拼写错误，应改为“temporary”。作者在最终代码中未修改拼写。
- 测试覆盖不足：多个评论建议补充单元测试验证投机解码开 / 关时的预算计算，但 PR 未包含测试改动。
    - Python 三元表达式优先级导致非投机解码场景预算错误 (correctness): 建议改用显式 if/else 语句消除歧义，最终实现已修复。

- 预算计算方式过于保守与魔法数字问题 (design): 作者保留了预减方案和魔法数字，作为临时解决方案。
- 测试覆盖不足 (testing): PR 未包含测试改动，建议后续补充。

风险与影响

- 风险：1. 回归风险：修改涉及调度器核心预算计算，若逻辑错误可能导致非投机解码场景吞吐下降（原 bug）或投机解码场景仍超限。当前实现通过临时下限保护缓解，但魔法数字 512 可能在高负载场景下不足。 2. 性能风险：预减方式可能过度保守，减少 prefill 预算，影响系统吞吐量。 3. 兼容性风险：无 API 或配置变更，兼容性良好。 4. 测试风险：缺乏针对投机解码场景的单元测试，未来变更易引入回归。
- 影响：1. 用户影响：修复了投机解码场景下的 OOM 问题，提升系统稳定性；但非投机解码场景预算计算已修复，避免吞吐退化。 2. 系统影响：调度器能更准确控制 token 预算，防止显存超限，但可能因保守预算降低资源利用率。 3. 团队影响：揭示了调度器与投机解码集成时的预算管理漏洞，为后续优化提供参考。
- 风险标记：核心路径变更，缺少测试覆盖，魔法数字

关联脉络

- PR #7237 [Optimization] Auto set num_max_dispatch_tokens_per_rank: 同样涉及调度器参数优化，关注投机解码状态下的配置调整，与本 PR 的预算计算相关。
- PR #7407 [BugFix][Scheduler]Fix FD_DISABLE_CHUNKED_PREFILL max_num_batched_tokens limit: 同为调度器 bugfix，修复 max_num_batched_tokens 相关限制问题，技术领域重叠。
- PR #7180 [XPU] Unify Spec and non-spec branch.(#6947): 涉及投机解码（Speculative Decoding）功能，与本 PR 修复的投机解码预算计算场景直接相关。