PR #23189 完整报告

sgl-project/sglang

feat(scheduler): add adaptive queue-based prefill delayer trigger

合并时间：2026-05-09 07:54

原文链接：http://prhub.com.cn/sgl-project/sglang/pull/23189

执行摘要

- 一句话：新增队列感知预填充延迟触发器
- 推荐动作：推荐精读。该 PR 通过精细的调度优化解决了负载方差导致的吞吐 collapse 问题，设计上保持了与原有触发器的兼容性和可选的启用方式。重点关注其队列触发与 slot 触发的组合逻辑、挂钟超时兜底的设计，以及跨 rank 同步扩展的方式。评论区的讨论（尤其是超时范围争议）也值得回顾。

功能与动机

基准测试显示，当随机范围比从 1.0 变为 0.8（OSL 从固定变为区间采样）时，输出吞吐从 7830 tok/s 降至 3846 tok/s，下降约 51%。根因分析表明：OSL 方差导致请求逐个结束，调度器每次仅回填 1-2 个请求的小预填充，预填充步骤数增加 62×，p99 解码间隙从 27.6ms 暴涨至 396.9ms。现有的 slot 触发器（--prefill-delayer-token-usage-low-watermark）仅当 max_running_requests - running_batch < max_prefill_bs 时触发，而在聚合部署中 max_running_requests 远大于 max_prefill_bs，因此该条件永不满足，无法阻止碎片化。

实现拆解

1. 新增 CLI 参数：在 server_args.py 的 ServerArgs 中添加 prefill_delayer_queue_min_ratio（可选 float）和 prefill_delayer_max_delay_ms（可选 float），并注册对应的 argparse 参数。
2. 扩展全局同步通道：在 prefill_delayer.py 中，将跨 rank 的 _global_info_buffer 从 4 列扩展为 5 列，新增 waiting_queue_len 字段；修改 _gather_info 方法以同步该列。
3. 传递等待队列长度：在 scheduler.py 的 _get_new_batch_prefill_raw 中，将 len(self.waiting_queue) 传入 PrefillAdder；在 schedule_policy.py 的 PrefillAdder.__init__ 中存储 self.waiting_queue_len，并在 add_one_req 调用 prefill_delayer_single_pass 时传递该值。
4. 实现队列触发决策：在 PrefillDelayer._negotiate_should_allow_prefill_pure 中新增队列触发逻辑：计算 queue_min = min(running_req * ratio, max_prefill_bs)，若所有 rank 的 waiting_queue_len 最小值小于 queue_min 则触发延迟；同时记录延迟起始时间，当超过 max_delay_ms 时强制释放（超时兜底）。该队列触发与原有 slot 触发是 OR 关系。
5. 配套测试与文档：在 test_prefill_delayer.py 中新增 4 个单元测试用例（queue_trigger_delay、queue_trigger_immediate、queue_trigger_wall_clock_timeout、

queue_trigger_no_effect_on_slot_trigger），覆盖队列触发延迟、立即放行、超时释放以及与 slot 触发共存等场景。同步在 server_arguments.md 中添加两个新参数的说明表格。

关键文件：

- python/sglang/srt/managers/prefill_delayer.py（模块 预填充延迟器；类别 source；类型 core-logic；符号 _queue_min_ratio, _queue_trigger_enabled, _max_delay_ms, _global_info_buffer）：核心文件，实现队列触发逻辑，包括配置读取、全局缓冲区扩展、触发决策
- test/registered/scheduler/test_prefill_delayer.py（模块 测试；类别 test；类型 test-coverage；符号 NegotiateCall, NegotiateTestCase, queue_min_ratio, max_delay_ms）：新增单元测试覆盖队列触发场景，验证核心决策逻辑的正确性
- python/sglang/srt/server_args.py（模块 配置；类别 source；类型 configuration；符号 prefill_delayer_queue_min_ratio, prefill_delayer_max_delay_ms）：定义两个新参数的数据类型和 CLI 接口
- python/sglang/srt/managers/schedule_policy.py（模块 策略；类别 source；类型 data-contract；符号 waiting_queue_len）：传递等待队列长度到 PrefillDelayer 决策入口
- python/sglang/srt/managers/scheduler.py（模块 调度器；类别 source；类型 entrypoint；符号 _get_new_batch_prefill_raw）：在创建 PrefillAdder 时传入当前等待队列长度
- docs/advanced_features/server_arguments.md（模块 文档；类别 docs；类型 documentation）：文档化新参数供用户参考

关键符号：PrefillDelayer.init, PrefillDelayer._negotiate_should_allow_prefill_pure, PrefillDelayer._gather_info, PrefillAdder.init, Scheduler._get_new_batch_prefill_raw

关键源码片段

python/sglang/srt/managers/prefill_delayer.py

核心文件，实现队列触发逻辑，包括配置读取、全局缓冲区扩展、触发决策

```
# 片段 1：初始化部分的队列触发配置与全局缓冲区扩展
class PrefillDelayer:
    def __init__(
        self,
        dp_size: int,
        attn_tp_size: int,
        cpu_group,
        server_args,
        max_delay_passes: int,
        token_usage_low_watermark: Optional[float],
        metrics_collector: Optional["SchedulerMetricsCollector"] = None,
        device: Optional[torch.device] = "cpu",
    ):
        self._max_delay_passes = max_delay_passes
        self._token_usage_low_watermark = token_usage_low_watermark

        # 队列触发配置：从 server_args 读取，仅当 queue_min_ratio 非 None 时启用
```

```
        self._queue_min_ratio = server_args.prefill_delayer_queue_min_ratio
        self._max_delay_ms = server_args.prefill_delayer_max_delay_ms
        if self._max_delay_ms is None:
            self._max_delay_ms = 5000.0 # 默认 5s 挂钟超时兜底
        self._queue_trigger_enabled = self._queue_min_ratio is not None

        # 全局信息缓冲区从 4 列扩展为 5 列，新增 waiting_queue_len
        dp_size_dim = dp_size if self.enable_dp_attention else 1
        self._global_info_buffer = torch.empty(
            (dp_size_dim, attn_tp_size, 5),
            dtype=torch.int64,
            device=device,
        )
        # 其余初始化省略 ...

# 片段 2：队列触发决策逻辑（位于 _negotiate_should_allow_prefill_pure 中）
# 全局信息同步后，从 tp0_info 提取各字段
global_waiting_queue_len = tp0_info[:, 4]

# 判断队列触发条件
queue_trigger = False
timeout_triggered = False
if self._queue_trigger_enabled:
    # 计算最小等待队列阈值：min(running_req * ratio, max_prefill_bs)
    queue_min = min(
        max_running_requests * self._queue_min_ratio,
        global_max_prefill_bs.min().item()
    )
    # 当所有 rank 的等待队列长度均小于阈值时触发延迟
    queue_trigger = global_waiting_queue_len.min().item() < queue_min
    if queue_trigger and prev_state is not None:
        elapsed_ms = (time.perf_counter() - prev_state.start_time) * 1000
        if elapsed_ms >= self._max_delay_ms:
            # 挂钟超时，强制放行
            timeout_triggered = True

# slot 触发条件（原有逻辑）
slot_condition = (
    global_max_prefill_bs.min().item() > 0
    and global_running_batch.min().item() + global_max_prefill_bs.min().item() > max_running_
    requests
)

# 综合决策：队列触发或 slot 触发任一为 True 且未超时且无节水印强制放行，则延迟
should_delay = (
    (slot_condition or queue_trigger)
    and not timeout_triggered
    and not global_exists_token_watermark_force_allow
)
```

test/registered/scheduler/test_prefill_delayer.py

新增单元测试覆盖队列触发场景，验证核心决策逻辑的正确性

```
# 片段 3：测试数据结构和队列触发测试用例
@dataclass
class NegotiateCall:
    prefillable: List[bool]
    token_usage: List[float]
    # 可选的调度器状态，None 表示不传递，兼容旧行为
    running_batch: Optional[List[int]] = None
    max_prefill_bs: Optional[List[int]] = None
    waiting_queue_len: Optional[List[int]] = None
    max_running_requests: Optional[int] = None
    sleep_before_s: float = 0.0 # 用于测试挂钟超时

@dataclass
class NegotiateTestCase:
    name: str
    max_delay_passes: int
    token_usage_low_watermark: Optional[float]
    calls: List[NegotiateCall]
    expected_allow: bool
    expected_reason: str
    # 队列触发配置，None 表示使用纯 slot 逻辑
    queue_min_ratio: Optional[float] = None
    max_delay_ms: Optional[float] = None

# 测试用例：等待队列不足时触发延迟
_NEGOTIATE_TEST_CASES.append(
    NegotiateTestCase(
        name="queue_trigger_delay",
        max_delay_passes=100,
        token_usage_low_watermark=0.8,
        queue_min_ratio=0.5,
        max_delay_ms=5000,
        calls=[
            NegotiateCall(
                prefillable=[True, True, True, True],
                token_usage=[0.9, 0.9, 0.9, 0.9],
                running_batch=[100, 100, 100, 100],
                max_prefill_bs=[80, 80, 80, 80],
                waiting_queue_len=[10, 10, 10, 10], # 远小于 queue_min=50
                max_running_requests=1024,
            ),
            # 第二个调用（skip_first_delayer 消耗了第一个延迟，现在应实际延迟）
            NegotiateCall(
                prefillable=[True, True, True, True],
                token_usage=[0.9, 0.9, 0.9, 0.9],
                running_batch=[100, 100, 100, 100],
```

```
                max_prefill_bs=[80, 80, 80, 80],
                waiting_queue_len=[10, 10, 10, 10],
                max_running_requests=1024,
            ),
        ],
        expected_allow=False,
        expected_reason="delay",
    )
)
```

评论区精华

- Fridge003 评论：询问能否用已有参数 --prefill-delayer-forward-passes-buckets 等实现队列触发。YAMY1234 回复：这些参数仅用于 Prometheus 直方图桶配置，不控制触发逻辑，因此需新增参数。
- Fridge003 评论：询问 slot_trigger 的含义。YAMY1234 将其重命名为 slot_condition 以提高可读性。
- gemini-code-assist[bot] 评论：指出超时机制同时影响 slot 触发和队列触发，可能导致 slot 限制仍存在时误释放。YAMY1234 在后续提交中修复，将 timeout 仅应用于队列触发组件。
- Fridge003 要求：添加测试和更新文档。YAMY1234 已补充。最终 Fridge003 批准合并。
  - 能否复用已有参数实现队列触发？ (design): 确认需要新增 --prefill-delayer-queue-min-ratio 和 --prefill-delayer-max-delay-ms。
  - slot_trigger 命名与角色 (style): 重命名为 slot_condition，提高可读性。
  - 超时兜底的作用范围 (correctness): 超时仅用于队列触发，slot 触发不受超时影响。
  - 测试覆盖要求 (testing): 测试已添加并通过 CI。

风险与影响

- 风险：
  - 回归风险：新参数默认未启用，现有行为不变；启用后队列触发逻辑与 slot 触发逻辑是 OR 关系，任何一方为 True 都会延迟预填充，因此不会意外跳过延迟，但可能增加延迟概率。
- TTFT 恶化：队列触发可能无限期等待更多请求，但 max_delay_ms 提供挂钟超时兜底（默认 5s），避免 TTFT 无限增长。
- 超时范围问题：早期版本中超时同时释放 slot 触发和队列触发，经 review 指出后已修复，超时仅作用于队列触发，slot 触发不受超时影响。
- 跨 rank 通信开销：全局缓冲区从 4 列增至 5 列，每次决策多同步一个 int64 值，开销可忽略。
- 兼容性：仅影响 enable_prefill_delayer=True 且 DP attention 启用的场景；非 DP 模式无变化。
- 影响：

  - 用户：提供新的性能优化选项，特别适合输出长度有波动的在线服务场景。用户需设置 --prefill-delayer-queue-min-ratio 来启用，典型值 0.1-0.5。
- 系统：在实验环境中，OSL 方差场景下输出吞吐提升 1.16x-1.85x（conc 16-128），且 GSM8K 准确率无退化。p99 解码间隙从 396.9ms 降至可接受水平。
- 团队：需维护两个新参数及对应测试，但改动集中在 prefill_delayer.py，模块内聚性较好。
- 风险标记：核心路径变更（调度器），新增 server 参数，跨 rank 通信扩展，超时范围问题（已修复）

关联脉络

- 暂无明显关联 PR