PR #26863 完整报告

sgl-project/sglang

Fix weights_checker checksum for 0-dim tensors and multi-GPU

合并时间：2026-06-01 12:27

原文链接：http://prhub.com.cn/sgl-project/sglang/pull/26863

执行摘要

- 一句话：修复权重检查器零维张量与多 GPU 支持
- 推荐动作：建议团队成员特别是部署运维人员阅读此 PR，了解 weights_checker 的新用法和潜在风险。对于分布式调试非常有用。关注 review 中提出的 deadlock 问题，确保实际部署环境中的进程模型匹配。

功能与动机

修复零维张量在 weights_checker 中调用 tensor_hash 和 _as_uint32_words 时因 .view(torch.uint8) 导致的错误；同时支持多 GPU 场景下的 checksum 聚合和 draft 模型权重校验，便于分布式部署中的权重一致性验证。

实现拆解

1. 修复零维张量哈希：在 python/sglang/srt/managers/mm_utils.py 的 tensor_hash 函数和 python/sglang/srt/layers/multimodal.py 的 _as_uint32_words 函数中，将 .view(torch.uint8) 之前插入 .reshape(-1)，确保零维张量（shape 为 []）先展平为 1D 再重新解释为字节，避免 view 因形状不连续而报错。此改动是核心修复，直接影响 checksum 计算的正确性。
2. 添加 per_gpu_checksum：在 python/sglang/srt/utils/weight_checker.py 的 ChecksumInfo 模型中新增 per_gpu_checksum: str 字段；在 _compute_checksum 方法中，对所有参数的 checksums 按键排序后，用 SHA-256 依次更新 name 和 checksum 值，生成一个统一的每 GPU 哈希值，并存入 per_gpu_checksum。这为后续跨 GPU 聚合提供了基础。
3. 支持多 GPU 负载收集：在 python/sglang/srt/managers/scheduler_components/weight_updater.py 的 check_weights 方法中，当 tp_size > 1 且 payload 非空时，通过 torch.distributed.all_gather_object 收集所有 TP rank 的 payload（已包含 per_gpu_checksum），并将 payload 替换为列表。这样 tokenizer manager 能拿到所有 rank 的结果。
4. 支持 draft 模型合并：在 weight_updater.py 中新增 _get_draft_model_runner 辅助函数，兼容 EAGLE v1（draft_model_runner 属性）和 v2（_draft_worker.draft_runner）；新增 _merge_checksum_payloads 函数，将 draft 模型的 checksum 以 draft. 前缀合并到主 payload 中，并重新计算 per_gpu_checksum。在 check_weights 中如果 draft_worker 存在且能获取 runner，则计算 draft 的 checksum 并合并。

5. 聚合引擎级 checksum 并调整 API：在 python/sglang/srt/managers/tokenizer_control_mixin.py 的 check_weights 方法中，处理返回的 ranks 列表时，对每个 rank 的 per_gpu_checksum 进行 SHA-256 聚合，生成 per_engine_checksum。同时更新返回类型为 4 元组。在 python/sglang/srt/entrypoints/http_server.py 中，将 /weights_checker 端点从 @app.post 改为 @app.api_route 支持 GET/POST，并使 obj 参数可选以支持 GET 请求（默认 action 为 "checksum"），响应中增加了 per_engine_checksum 字段。同时更新了 CheckWeightsReqInput 的 action 默认值为 "checksum"（在 io_struct.py 中）。

关键文件：

- python/sglang/srt/managers/scheduler_components/weight_updater.py（模块 调度器；类别 source；类型 core-logic；符号 _get_draft_model_runner, _merge_checksum_payloads）：核心逻辑：新增 draft 模型支持、多 GPU all_gather 收集 checksum 负载，是本次变更的主战场。
- python/sglang/srt/utils/weight_checker.py（模块 权重检查器；类别 source；类型 dependency-wiring）：添加 per_gpu_checksum 字段和聚合计算，是 checksum 数据模型的根基。
- python/sglang/srt/managers/mm_utils.py（模块 多模态；类别 source；类型 core-logic）：修复零维张量在 tensor_hash 中的崩溃，是本次 bugfix 的核心。

关键符号：_get_draft_model_runner, _merge_checksum_payloads, check_weights, _compute_checksum, tensor_hash, _as_uint32_words

关键源码片段

python/sglang/srt/managers/scheduler_components/weight_updater.py

核心逻辑：新增 draft 模型支持、多 GPU all_gather 收集 checksum 负载，是本次变更的主战场。

```
# 辅助函数：从 draft_worker 中提取模型 runner，兼容 EAGLE v1/v2
def _get_draft_model_runner(draft_worker):
    # EAGLEWorker (v1): draft_model_runner 属性指向 self.model_runner
    runner = getattr(draft_worker, "draft_model_runner", None)
    if runner is not None:
        return runner
    # EAGLEWorkerV2: _draft_worker.draft_runner
    inner = getattr(draft_worker, "_draft_worker", None)
    if inner is not None:
        runner = getattr(inner, "draft_runner", None)
        if runner is not None:
            return runner
    return None


# 辅助函数：将 draft 模型的 checksum 合并到主 payload 中，并重新计算 per_gpu_checksum
def _merge_checksum_payloads(target: Dict, draft: Dict) -> Dict:
    # 以 draft. 前缀合并 checksums，避免与主模型同名参数冲突
    merged_checksums = dict(target["checksums"])
```

```
    for name, chk in draft["checksums"].items():
        merged_checksums[f"draft.{name}"] = chk
    # 基于所有参数（含 draft）的有序集合计算 per_gpu_checksum
    h = hashlib.sha256()
    for name in sorted(merged_checksums):
        h.update(name.encode())
        h.update(merged_checksums[name].encode())
    target["checksums"] = merged_checksums
    target["per_gpu_checksum"] = h.hexdigest()
    return target

# 修改后的 check_weights 方法片段
def check_weights(self, recv_req: CheckWeightsReqInput):
    try:
        payload = self.tp_worker.model_runner.check_weights(action=recv_req.action)

        # 若有 draft model，计算其 checksum 并合并
        if self.draft_worker is not None:
            draft_runner = _get_draft_model_runner(self.draft_worker)
            if draft_runner is not None:
                draft_payload = draft_runner.check_weights(action=recv_req.action)
                if payload is not None and draft_payload is not None:
                    payload = _merge_checksum_payloads(payload, draft_payload)

        # 多 GPU 场景：通过 all_gather_object 收集所有 rank 的 payload
        tp_size = torch.distributed.get_world_size(group=self.tp_cpu_group)
        if tp_size > 1 and payload is not None:
            all_payloads = [None] * tp_size
            torch.distributed.all_gather_object(
                all_payloads, payload, group=self.tp_cpu_group
            )
            payload = all_payloads

        return CheckWeightsReqOutput(
            success=True, message="Success.", payload=payload
        )
    except Exception as e:
        logger.warning(f"check_weights see error: {e}")
        traceback.print_exc()
        return CheckWeightsReqOutput(success=False, message=f"{e}")
```

python/sglang/srt/utils/weight_checker.py

添加 per_gpu_checksum 字段和聚合计算，是 checksum 数据模型的根基。

```
# ChecksumInfo 数据模型（新增 per_gpu_checksum 字段）
class ChecksumInfo(_StrictBaseModel):
    checksums: Dict[str, str]
    per_gpu_checksum: str # 当前 GPU 上所有参数的聚合哈希
    parallelism_info: ParallelismInfo
```

```
# _compute_checksum 方法中的核心变更片段
def _compute_checksum(self) -> Dict:
    torch.cuda.synchronize()
    start = time.perf_counter()

    skip_compare_names = {
        name for name, param in self._model_state()
        if getattr(param, "_skip_weight_check", False)
    }

    # 计算每个参数的 checksum（经过 fp8 反量化等预处理）
    checksums = {
        name: _hash_tensor(tensor.data)
        for name, should_compare, tensor in _postprocess_tensors(
            dict(self._model_state()), skip_compare_names
        )
        if should_compare
    }

    # 新增：基于所有参数的有序名称和 checksum 计算 per_gpu_checksum
    h = hashlib.sha256()
    for name in sorted(checksums):
        h.update(name.encode())
        h.update(checksums[name].encode())
    overall = h.hexdigest()

    torch.cuda.synchronize()
    elapsed = time.perf_counter() - start
    logger.info(
        f"[WeightChecker] checksum computed for {len(checksums)} tensors in {elapsed:.3f}s"
    )

    info = ChecksumInfo(
        checksums=checksums,
        per_gpu_checksum=overall, # 存入数据模型
        parallelism_info=self._parallelism_info(),
    )
    return info.model_dump()
```

python/sglang/srt/managers/mm_utils.py

修复零维张量在 tensor_hash 中的崩溃，是本次 bugfix 的核心。

```
def tensor_hash(tensor_list) -> int:
    # ... 前处理 ...
    for t in tensors:
        t = t.detach().contiguous()
        # 使用 reshape(-1) 确保任何维度的张量都能展平为 1D，然后重新解释为 uint8 字节
        hasher.update(memoryview(t.reshape(-1).view(torch.uint8).numpy()))
    hash_bytes = hasher.digest()[:8]
```

```
    return int.from_bytes(hash_bytes, byteorder="big", signed=False)
```

评论区精华

- 多 GPU all_gather_object 潜在死锁：gemini-code-assist[bot] 指出 SchedulerWeightUpdaterManager 仅运行在调度器进程（通常只有 rank 0），而其他 TP worker 不执行 check_weights，因此调用 all_gather_object 会导致死锁，因为只有 rank 0 参与收集，其他 rank 永远不会调用对应的方法。不过 PR 已合并，可能此问题在实际部署中未触发，或者有隐含的通信模式未被考虑。
- None payload 与 checksum 顺序非确定：同样由 gemini-code-assist[bot] 指出，tokenizer_control_mixin.py 中假设所有 payload 非空，但某 rank 失败时 payload 可能为 None，导致 TypeError；同时 per_engine_checksum 的计算依赖于 ranks 列表顺序，但 all_gather_object 的顺序是确定的（按 rank 顺序），但若 ranks 中混入 None，则保持顺序的重要性降低。这两个问题未在后续讨论中得到回应。
  - 多 GPU all_gather_object 潜在死锁 (correctness): 未在讨论中得到回应，PR 已合并。可能实际部署中所有 TP rank 都运行 scheduler 进程，或死锁已被其他机制避免。
  - None payload 与 checksum 顺序非确定 (correctness): 未在讨论中得到回应，PR 已合并。建议后续添加空值检查和防御性编程。

风险与影响

- 风险：
  - 死锁风险：多 GPU 下的 all_gather_object 可能因调度器与 worker 进程不一致导致死锁，需要确认所有进程是否都参与该通信。如果只有 rank 0 训练 scheduler，其他 rank 未调用 check_weights 则可能死锁。但目前 PR 已合并且 CI 可能通过，暗示实际场景中所有 TP rank 都运行 scheduler？需要进一步排查。
  - None payload 崩溃：当某个 rank 的计算失败时，其 payload 为 None，在聚合时直接访问 rank["per_gpu_checksum"] 会引发 TypeError。应添加空值检查。
  - 性能影响：tensor_hash 中增加了 reshape(-1)，对零维张量无影响，不影响大部分场景性能。all_gather_object 仅在调用 weights_checker 时执行，频率极低，无显著性能负担。
  - 兼容性：/weights_checker 端点改为 GET/POST 后，旧的 POST-only 客户端仍可正常工作（但返回新增的 per_engine_checksum 字段，客户端需忽略未知字段）。
- 影响：
  - 用户 / 运维：现在可以通过 GET 请求快速获取 weights checksum，无需构造 POST body；返回的 checksum 信息更丰富，便于跨机器对比权重一致性。
  - 系统：增加了分布式 all_gather 通信，但仅在显式调用时发生，不影响常规推理。
  - 多模态模块：tensor_hash 和 _as_uint32_words 被修复，可能影响其他依赖这些函数的功能（如多模态编解码中的哈希操作），但修复是向前兼容的。
  - 风险标记：核心路径变更，潜在死锁，None 值处理，缺少异常保护

关联脉络

- 暂无明显关联 PR