PR #21701 完整报告

sgl-project/sglang

[diffusion] disaggregated diffusion

合并时间：2026-04-16 23:51

原文链接：http://prhub.com.cn/sgl-project/sglang/pull/21701

执行摘要

- 一句话：新增扩散模型解聚架构，将编码器、去噪器、解码器角色独立运行于不同 GPU 实例。
- 推荐动作：建议仔细阅读 scheduler_mixin.py 和 orchestrator.py 以理解核心调度和路由逻辑；关注 review 中讨论的设计决策，如数据类初始化和传输协议设计，以避免潜在缺陷；注意风险点，如实例索引一致性和性能优化，建议在部署前进行全面测试。

功能与动机

根据 review 评论，动机是构建一个解聚的扩散管道架构，允许编码器、去噪器和解码器角色在独立的 GPU 实例上运行（引用 gemini-code-assist[bot] 的表述：'allowing the Encoder, Denoiser, and Decoder roles to run on independent GPU instances'），从而提高系统的可扩展性和资源利用率。

实现拆解

1. 新增调度器混合：文件 scheduler_mixin.py 添加 SchedulerDisaggMixin，扩展核心调度器以支持解聚角色，处理 Req 对象的字段提取、张量序列化以及事件循环逻辑。
2. 中央协调器：文件 orchestrator.py 引入 DiffusionServer 类，作为全局管道协调器，负责容量感知的请求分派、TTA 队列管理和传输状态跟踪。
3. 传输基础设施：新增 transport 目录下的多个文件，包括 TransferTensorBuffer（基于 BuddyAllocator 的内存池）、DiffusionTransferManager（管理张量传输）、BaseTransferEngine（抽象传输引擎，支持 RDMA）等，实现张量的 P2P 传输。
4. 配置和文档：文件 disagg_args.py 扩展服务器 CLI 参数，添加 --disagg-role 等标志；文档文件如 disaggregation.md 详细说明解聚管道的使用和部署。
5. 测试配套：新增端到端测试文件，集成到 CI 中，确保功能正确性并覆盖边缘场景。

关键文件：

- python/sglang/multimodal_gen/runtime/disaggregation/scheduler_mixin.py（模块 多模态调度；类别 source；类型 core-logic；符号 _is_tensor_like, _to_json_serializable, _is_default, _extract_extra_fields）：核心调度器混合，负责解聚角色的传输和事件循环逻辑，定义了字段提取、张量序列化等关键函数。
- python/sglang/multimodal_gen/runtime/disaggregation/orchestrator.py（模块 管道协调；类别 source；类型 core-logic；符号 _EncoderTTAEntry, _TransferRequestState,

post_init, _RoleTTAEntry）：中央请求路由器，管理编码器、去噪器和解码器实例间的请求调度和传输协调。

- python/sglang/multimodal_gen/runtime/disaggregation/transport/manager.py（模块 传输层；类别 source；类型 core-logic；符号 StagedTransfer, PendingReceive, DiffusionTransferManager, init）：单角色实例的传输管理器，负责张量暂存和传输协调，连接传输引擎和内存缓冲区。

关键符号：extract_transfer_fields, SchedulerDisaggMixin, DiffusionServer.init, stage_tensors, allocate, transfer_sync, create_transfer_engine

关键源码片段

python/sglang/multimodal_gen/runtime/disaggregation/scheduler_mixin.py

核心调度器混合，负责解聚角色的传输和事件循环逻辑，定义了字段提取、张量序列化等关键函数。

```
# 字段提取：将 Req 拆分为张量（传输缓冲区）和标量（JSON）
_EXCLUDE_FIELDS = frozenset({
    "sampling_params", "generator", "modules", "metrics", "extra_step_kwargs",
    "extra", "condition_image", "vae_image", "pixel_values", "preprocessed_image",
    "image_embeds", "original_condition_image_size", "vae_image_sizes", "output",
    "audio", "audio_sample_rate", "trajectory_timesteps", "trajectory_latents",
    "trajectory_audio_latents", "timestep", "step_index", "prompt_template",
    "max_sequence_length"
})

def _is_tensor_like(value) -> bool:
    """检查值是否为类张量对象（torch.Tensor 或张量列表）。"""
    if isinstance(value, torch.Tensor):
        return True
    if isinstance(value, list) and value and isinstance(value[0], torch.Tensor):
        return True
    return False

def extract_transfer_fields(req):
    """从 Req 对象提取可传输的字段，分为张量和标量两部分。"""
    tensor_fields = {}
    scalar_fields = {}
    for field_name, field_value in dataclasses.asdict(req).items():
        if field_name in _EXCLUDE_FIELDS:
            continue # 跳过内部或不可序列化字段
        if _is_tensor_like(field_value):
            tensor_fields[field_name] = field_value # 张量字段进入传输缓冲区
        else:
            scalar_fields[field_name] = _to_json_serializable(field_value) # 标量字段转为 JSON
    return tensor_fields, scalar_fields
```

python/sglang/multimodal_gen/runtime/disaggregation/orchestrator.py

中央请求路由器，管理编码器、去噪器和解码器实例间的请求调度和传输协调。

```
class DiffusionServer:
    """全局管道协调器，支持 N:M:K 解聚扩散管道，具备容量感知分派和 TTA 队列管理。"""

    def __init__(
        self,
        frontend_endpoint: str,
        encoder_work_endpoints: list[str],
        denoiser_work_endpoints: list[str],
        decoder_work_endpoints: list[str],
        encoder_result_endpoint: str,
        denoiser_result_endpoint: str,
        decoder_result_endpoint: str,
        dispatch_policy_name: str = "round_robin",
        timeout_s: float = 600.0,
        encoder_capacity: int = 4,
        denoiser_capacity: int = 2,
        decoder_capacity: int = 4,
        p2p_mode: bool = True,
    ):
        self._frontend_endpoint = frontend_endpoint
        self._encoder_work_endpoints = encoder_work_endpoints
        self._denoiser_work_endpoints = denoiser_work_endpoints
        self._decoder_work_endpoints = decoder_work_endpoints
        self._encoder_result_endpoint = encoder_result_endpoint
        self._denoiser_result_endpoint = denoiser_result_endpoint
        self._decoder_result_endpoint = decoder_result_endpoint

        self._num_encoders = len(encoder_work_endpoints)
        self._num_denoisers = len(denoiser_work_endpoints)
        self._num_decoders = len(decoder_work_endpoints)
        self._timeout_s = timeout_s

        self._tracker = RequestTracker() # 请求状态跟踪器
        self._dispatcher = PoolDispatcher(
            num_encoders=self._num_encoders,
            num_denoisers=self._num_denoisers,
            num_decoders=self._num_decoders,
            policy_name=dispatch_policy_name, # 分派策略如轮询或最大空闲槽优先
        )

        self._context = zmq.Context(io_threads=2)
        self._running = False
        self._ready = threading.Event() # 用于同步启动
        self._thread: threading.Thread | None = None

        self._pending: dict[str, bytes] = {} # request_id -> client ZMQ identity
        self._lock = threading.Lock()
```

```
# 每个实例的 FreeBufferSlots，用于容量管理
self._encoder_free_slots = [encoder_capacity] * self._num_encoders
self._denoiser_free_slots = [denoiser_capacity] * self._num_denoisers
self._decoder_free_slots = [decoder_capacity] * self._num_decoders

# 每个角色类型的 TTA（Task-Transfer-Ack）队列
self._encoder_tta: deque[_EncoderTTAEntry] = deque()
self._denoiser_tta: deque[_RoleTTAEntry] = deque()
self._decoder_tta: deque[_RoleTTAEntry] = deque()

self._transfer_mode = p2p_mode # 是否启用点对点传输
self._transfer_state: dict[str, _TransferRequestState] = {} # 传输状态跟踪
# 省略后续实例注册和套接字初始化逻辑...
```

评论区精华

- 平均延迟计算错误：gemini-code-assist[bot] 指出 metrics.py 中平均延迟计算错误，分母包含了失败请求但分子只计入成功请求，导致低估（正确性风险）。
- 传输协议死代码：protocol.py 中 result_frames 字段被移除，导致 diffusion_server.py 中的相关分支成为死代码（设计缺陷）。
- 数据类初始化改进：多处建议使用 field(default_factory=dict) 替代 __post_init__ 来初始化可变默认值，以提升代码风格和可维护性。
- 实例索引一致性问题：mickqian 指出 _denoiser_peers 和 _denoiser_pushes 顺序不一致可能导致控制消息发送错误，需要确保索引对齐（正确性风险）。
- 文档改进：mickqian 建议将文档移动到 docs/diffusion 目录并补充内容如 TimestepPreparationStage。
- 性能优化：建议对 allocator.py 中的函数使用 lru_cache 缓存，提升分配效率。
  - 平均延迟计算错误 (correctness): 建议分母只使用成功请求数（self._completed）以正确计算平均延迟。
  - 传输协议死代码 (design): 需要修复协议以保留 result_frames 或移除死代码分支，确保功能完整性。
  - 实例索引一致性问题 (correctness): 需要确保实例索引对齐，例如通过统一排序或映射机制来匹配控制平面和数据平面。

风险与影响

- 风险：- 正确性风险：metrics.py 中的平均延迟计算错误可能影响监控准确性；传输协议死代码可能导致结果返回路径失效；实例索引不一致可能引发数据错乱或传输失败。
- 性能风险：传输缓冲区分配（BuddyAllocator）在高压下可能产生碎片或竞争条件；RDMA 传输引擎依赖特定硬件（如 Mooncake），若不可用可能回退到低效路径。
- 兼容性风险：新增 CLI 参数（如 --disagg-role）需要用户适配部署配置；依赖 Python 3.10+ 的特性可能限制环境。
- 测试覆盖风险：尽管有端到端测试，但复杂角色交互和传输场景可能仍需更多集成测试以确保稳定性。

- 影响：- 用户影响：用户可通过 --disagg-role 标志启用解聚模式，灵活部署扩散管道到异构 GPU 环境，但需学习新配置和架构概念。
- 系统影响：核心调度逻辑扩展以支持解聚角色，新增传输层增加系统复杂性，但提升资源利用率和横向扩展能力；传输缓冲区管理可能影响内存使用模式。
- 团队影响：为多模态扩散模块引入新架构范式，相关开发者需熟悉解聚概念、传输协议和调试工具，可能增加维护成本。
- 风险标记：延迟计算错误，协议死代码，索引一致性风险，RDMA 依赖

关联脉络

- PR #22490 [EPD][VLM] Support Kimi VL EPD: 同属多模态解聚功能线，扩展 EPD 分解管道以支持 Kimi VL 模型，可能共享类似架构或传输逻辑。
- PR #22920 Remove compatibility restriction between Pipeline Parallelism and Mixed Chunked Prefill: 涉及调度器兼容性调整，可能影响解聚模式下的流水线并行配置，需注意协同效应。