PR #48042 完整报告

vllm-project/vllm

[rl] Stateful Trainer Send: New Abstractions [1/N]

合并时间：2026-07-17 15:11

原文链接：http://prhub.com.cn/vllm-project/vllm/pull/48042

执行摘要

- 一句话：引入有状态 Trainer 权重发送新抽象
- 推荐动作：值得精读。重点关注：
  - WeightSource 的分通道设计（metadata 不触发 all-gather vs 迭代时 materialize）
  - VLLMWeightSyncClient 使用结构协议而非抽象基类，降低耦合
  - materialize_full_tensor 如何安全处理 FSDP 分片
  - WeightTransferTrainerFactory 的 lazy-load 注册模式 对于需要集成自定义训练引擎的团队，理解这些抽象有助于写出适配器。

功能与动机

原有 WeightTransferEngine 将 Trainer 视为无状态，通过静态方法 trainer_send_weights 和 per-backend 参数 args 实现传输，导致状态散落在调用者、类型系统被绕过（trainer_args: dict | Any）。旨在使 Trainer 端与 Worker 端对称，拥有状态引擎，通过 WeightSource 拉取权重，通过传输无关的客户端驱动握手。同时为更好的 M2N 集成提供注册点。

实现拆解

1. 核心抽象（base.py）新增 materialize_full_tensor 工具函数、ParamMeta 数据类、WeightSource ABC（含 metadata 和 __iter__ 两个通道）、ModuleSource 实现、TrainerInitInfo（带 is_sender 属性）、TrainerWeightTransferEngine ABC（有状态，trainer_init 工厂方法、send_weights 和可选 shutdown）、VLLMWeightSyncClient 结构协议（四个同步方法）。
2. 传输客户端（clients.py）新增 HTTPVLLMWeightSyncClient（通过 RLHF HTTP 端点通信）和 RayVLLMWeightSyncClient（向一个或多个 Ray actor 扇出请求），以及 _json_safe_update_info 辅助函数将 CUDA IPC handles 安全编码为 JSON。导入 requests/ray 延迟到调用时，降低导入依赖。
3. 工厂注册（factory.py）新增 WeightTransferTrainerFactory，与 WeightTransferEngineFactory 平行，提供 register_engine 和 trainer_init 方法，当前注册表为空（需后续 PR 注册具体后端引擎）。
4. 引擎入口兼容（async_llm.py）修改 init_weight_transfer_engine 和 update_weights，接受 dict | Request 类型参数，以兼容新 Ray 客户端传递请求对象。旧调用路径保持不变。

5. 模块导出与测试（__init__.py、test_weight_transfer.py）更新 __init__.py 导出所有新符号；在测试文件中添加 GPU-free 单元测试：RecordingClient（记录调用顺序）、_module_with（快速构建参数模块）、_DummyTrainerEngine（最小化具体实现），覆盖协议符合性、工厂调用链、metadata 准确性和 iteration 独立性。
6. 安全白名单（check_forbidden_imports.py）将 clients.py 添加到 pickle 导入白名单。

关键文件：

- vllm/distributed/weight_transfer/clients.py（模块 客户端；类别 source；类型 core-logic；符号 _json_safe_update_info, HTTPVLLMWeightSyncClient, RayVLLMWeightSyncClient, init_weight_transfer_engine）：新增文件，实现内置的 HTTP 和 Ray 两种重量同步客户端，是 trainer 端新抽象的核心传输适配层。
- vllm/distributed/weight_transfer/base.py（模块 抽象层；类别 source；类型 dependency-wiring；符号 materialize_full_tensor, ParamMeta, WeightSource, metadata）：修改文件，新增训练器端核心抽象：WeightSource、ModuleSource、TrainerInitInfo、TrainerWeightTransferEngine 和 VLLMWeightSyncClient 协议，是整个 PR 的设计核心。
- vllm/distributed/weight_transfer/factory.py（模块 工厂；类别 source；类型 dependency-wiring；符号 WeightTransferTrainerFactory, register_engine, loader, trainer_init）：新增 WeightTransferTrainerFactory，为训练器引擎提供独立的注册和初始化入口，与已有 WeightTransferEngineFactory 平行。
- tests/distributed/test_weight_transfer.py（模块 测试；类别 test；类型 test-coverage；符号 RecordingClient, _module_with, _DummyTrainerEngine, TestTrainerClients）：新增训练器抽象的 GPU-free 单元测试，包括协议符合性验证、工厂调用栈检查、metadata 准确性和 iteration 独立性。
- vllm/v1/engine/async_llm.py（模块 引擎；类别 source；类型 dependency-wiring；符号 init_weight_transfer_engine, update_weights）：修改 init_weight_transfer_engine 和 update_weights 方法，支持 dict | Request 类型以兼容新 Ray 客户端，向后兼容。

关键符号：materialize_full_tensor, _json_safe_update_info, WeightSource.metadata, ModuleSource.iter, TrainerWeightTransferEngine.trainer_init, WeightTransferTrainerFactory.trainer_init, VLLMWeightSyncClient.init_weight_transfer_engine

关键源码片段

vllm/distributed/weight_transfer/clients.py

新增文件，实现内置的 HTTP 和 Ray 两种重量同步客户端，是 trainer 端新抽象的核心传输适配层。

```
# SPDX-License-Identifier: Apache-2.0
# SPDX-FileCopyrightText: Copyright contributors to the vLLM project
"""Built-in `VLLMWeightSyncClient` implementations.

These adapt the inference engine's weight-sync control plane to concrete
transports. A `TrainerWeightTransferEngine` takes one of these (or any object
```

```
with the same four methods — the protocol is structural) and drives the full
handshake through it.

Imports of `ray` / `requests` are deferred to call time so this module is
importable without those packages installed.
"""

from typing import TYPE_CHECKING, Any

from vllm.distributed.weight_transfer.base import (
    WeightTransferInitRequest,
    WeightTransferUpdateRequest,
)

if TYPE_CHECKING:
    from ray.actor import ActorHandle


def _json_safe_update_info(update_info: dict[str, Any]) -> dict[str, Any]:
    """Make an update_info dict JSON-serializable for HTTP transport.

    CUDA IPC handles (`ipc_handles`) are tuples of non-JSON-native objects, so
    over HTTP they are pickled+base64-encoded into `ipc_handles_pickled` (which
    the worker auto-deserializes when `VLLM_ALLOW_INSECURE_SERIALIZATION=1`).
    Other backends (NCCL) carry only JSON-native metadata and pass through
    unchanged. Mirrors the old IPC `_do_send` HTTP branch.
    """
    ipc_handles = update_info.get("ipc_handles")
    if ipc_handles is None:
        return update_info

    import pickle

    import pybase64 as base64

    out = {k: v for k, v in update_info.items() if k != "ipc_handles"}
    out["ipc_handles_pickled"] = base64.b64encode(pickle.dumps(ipc_handles)).decode(
        "utf-8"
    )
    return out


class HTTPVLLMWeightSyncClient:
    """Talks to a vLLM server over the RLHF HTTP routes.

    Mirrors `vllm/entrypoints/serve/dev/rlhf/api_router.py`:
    `/init_weight_transfer_engine`, `/start_weight_update`, `/update_weights`,
    `/finish_weight_update`.
    """
```

```
    def __init__(self, base_url: str, timeout: float = 300) -> None:
        self.base_url = base_url.rstrip("/")
        self.timeout = timeout

    def _post(self, path: str, json: dict[str, Any] | None = None) -> None:
        import requests

        response = requests.post(
            f"{self.base_url}/{path}", json=json, timeout=self.timeout
        )
        response.raise_for_status()

    def init_weight_transfer_engine(self, init_info: dict[str, Any]) -> None:
        self._post("init_weight_transfer_engine", {"init_info": init_info})

    def start_weight_update(self) -> None:
        self._post("start_weight_update")

    def update_weights(self, update_info: dict[str, Any]) -> None:
        self._post(
            "update_weights", {"update_info": _json_safe_update_info(update_info)}
        )

    def finish_weight_update(self) -> None:
        self._post("finish_weight_update")
```

vllm/distributed/weight_transfer/base.py

修改文件，新增训练器端核心抽象：WeightSource、ModuleSource、TrainerInitInfo、TrainerWeightTransferEngine 和 VLLMWeightSyncClient 协议，是整个 PR 的设计核心。

```
from abc import ABC, abstractmethod
from collections.abc import Iterator
from dataclasses import dataclass
from typing import Any

import torch

# ---- 新增加的训练器端抽象 ----

def materialize_full_tensor(tensor: torch.Tensor) -> torch.Tensor:
    """Return a full, locally-materialized tensor ready to send.

    FSDP shards (DTensors) expose `full_tensor()`, a collective all-gather;
    regular tensors do not and are returned unchanged. Trainer engines call
    this at send time so the (potentially expensive) gather happens exactly
    once — reading `.shape`/`.dtype` for metadata does not trigger it.
    """
    full_tensor = getattr(tensor, "full_tensor", None)
    return full_tensor() if callable(full_tensor) else tensor
```

```
@dataclass(frozen=True)
class ParamMeta:
    """Name / wire dtype / full (HF) shape for one output parameter."""
    name: str
    dtype: torch.dtype
    shape: tuple[int, ...]


class WeightSource(ABC):
    """A re-iterable source of the trainer's weights, handed to a trainer engine.

    Two channels:
    * `metadata()` — `(name, wire dtype, full shape)` for every parameter,
      *without* transferring. Cheap when shapes are known locally (FSDP
      `DTensor` global shape); may be expensive on first call for backends that
      must materialize to learn shapes (e.g. a Megatron-Bridge export), in which
      case it should cache.
    * iteration — yields fully-materialized `(name, tensor)` pairs, one at a
      time. Materializing is typically a collective (FSDP `full_tensor()`, a
      Megatron export), so every trainer rank must iterate the same source in the
      same order in lockstep, or ranks deadlock. Under pipeline parallelism a
      rank may not own a parameter at all — iterating still drives the collective
      and the yielded tensor is only meaningful on the sender.

    `iter(source)` must yield a *fresh* pass each round. Backends with custom
    producer logic (Megatron export, RDT plans, MoE re-fusing) subclass this.
    """

    @abstractmethod
    def metadata(self) -> list[ParamMeta]:
        raise NotImplementedError

    @abstractmethod
    def __iter__(self) -> Iterator[tuple[str, torch.Tensor]]:
        raise NotImplementedError


class ModuleSource(WeightSource):
    """`WeightSource` over `module.named_parameters()` — the common case.

    Handles both plain dense modules and FSDP-sharded ones with no special
    casing: iteration all-gathers each `DTensor` via `full_tensor()` (a
    collective) and passes regular tensors through. `metadata()` reads the
    *global* `.shape` / `.dtype`, so it never triggers a gather.
    """

    def __init__(self, module: torch.nn.Module) -> None:
        self._module = module
```

```
    def metadata(self) -> list[ParamMeta]:
        return [
            ParamMeta(name, p.dtype, tuple(p.shape))
            for name, p in self._module.named_parameters()
        ]

    def __iter__(self) -> Iterator[tuple[str, torch.Tensor]]:
        for name, param in self._module.named_parameters():
            yield name, materialize_full_tensor(param)
```

评论区精华

- async_llm.py 参数类型放宽：SumanthRH 问为什么将 WeightTransferInitRequest 改为 WeightTransferInitRequest | dict。作者解释这是为了兼容新 Ray 客户端，但后来通过创建 typed payload 避免了 dict。状态：已解决。
- is_sender 单一真源问题：aoshen02 指出 VLLMWeightSyncClient.init_weight_transfer_engine 中的 is_sender 应从 TrainerInitInfo 获取，避免重复。作者认为当前设计可接受，留待后续优化。状态：已确认识别。
- M2N 集成支持：kwen2501 询问新抽象是否帮助 M2N 集成，并指出 ParamMeta 和 TrainerInitInfo 缺乏 mesh 拓扑信息。作者确认有帮助，但拓扑信息需后续添加。状态：待跟进。
  - async_llm.py 参数类型放宽 (question): 作者解释是为了兼容新 Ray 客户端，但后来通过创建 typed payload 避免了 dict。
  - is_sender 单一真源问题 (design): 作者认为当前设计可接受，留待后续优化。
  - M2N 集成与拓扑信息缺失 (design): 作者确认有帮助，但拓扑信息需后续添加。

风险与影响

- 风险：本 PR 为纯添加，不修改现有路径，技术风险较低。但存在以下风险：
  1. 新抽象设计需经后续后端迁移验证，可能暴露出接口不完善需返工。
  2. _json_safe_update_info 将 IPC handles pickle 编码，仍需 VLLM_ALLOW_INSECURE_SERIALIZATION 环境变量保护，存在安全性依赖。
  3. WeightTransferTrainerFactory 注册表当前为空，如果后续 PR 未及时注册，使用时会立即报错。
  4. RayVLLMWeightSyncClient 在 __init__ 中调用了 import ray，但导入延迟到方法调用，无问题。
  5. async_llm.py 的类型变更向后兼容，但旧代码路径中 isinstance 检查被移除，如果调用方传递了不符合预期的类型可能触发异常。总体来说风险可控。 - 影响：对用户：无直接功能影响，所有新抽象尚未被使用。对系统：代码量增加 644 行，新增两个客户端类和工厂，但仅在导入时加载。对团队：这是系列 PR 的基础，后续需要为 IPC、NCCL 等后端实现具体引擎并注册。新抽象提供了清晰的扩展点（WeightTransferTrainerFactory.register_engine），降低后续集成成本。影响范围：中，局限于分布式权重传输模块。 - 风险标记：安全序列化风险，新抽象未经端到端验证，后续迁移依赖

关联脉络

- PR #47357 [rl] Stateful Trainer Send (original large PR): 本 PR 是大 PR 拆分后的第一部分，原始 PR 被拆分为三个 PR。