PR #6507 完整报告

verl-project/verl

[ckpt] feat: pass global steps to checkpoint engines

合并时间：2026-05-29 14:10

原文链接：http://prhub.com.cn/verl-project/verl/pull/6507

执行摘要

- 一句话：为 checkpoint 引擎添加 global steps 参数
- 推荐动作：建议阅读本 PR 以学习如何以向后兼容方式扩展关键抽象接口。对于使用自定义 checkpoint 后端的团队，应尽快为 send/receive 方法添加 global_steps: int | None = None 参数。同时推荐在后续 PR 中修复 ColocatedCheckpointEngine 的异步一致性问题，并补充 GPU 集成测试。

功能与动机

Custom checkpoint backends may need the update step as a stable model-version identifier. In particular, external/versioned weight-sync backends need the same version on the trainer publish side and rollout receive side rather than relying on local counters or environment variables.

实现拆解

1. 在抽象基类 verl/checkpoint_engine/base.py 的 CheckpointEngine 中扩展 send_weights 和 receive_weights 方法签名，添加可选的 global_steps: int | None = None 参数。
2. 更新所有具体引擎实现（nccl_checkpoint_engine.py, nixl_checkpoint_engine.py, mooncake_checkpoint_engine.py, hccl_checkpoint_engine.py, kimi_checkpoint_engine.py, 以及本文件中的 ColocatedCheckpointEngine）使其签名接受新参数，实现中忽略该参数以保持行为不变。
3. 在 verl/workers/engine_workers.py 中修改 ActorRolloutRefWorker.update_weights() 使其将 global_steps 传递给 send_weights；修改 CheckpointEngineWorker.update_weights() 使其将 global_steps 传递给 receive_weights 和 server adapter 的 update_weights。
4. 新增 tests/checkpoint_engine/test_global_steps_on_cpu.py，定义 _FakeTrainerEngine、_FakeCheckpointEngine 和 _FakeServerAdapter 用具类，编写两个测试用例分别验证 trainer→checkpoint engine 和 checkpoint engine→rollout 的 global_steps 传播。
5. 修改 tests/checkpoint_engine/test_utils.py 中的测试辅助函数以接受并转发 global_steps 参数。
6. 验证：所有改动通过 py_compile 和 CPU 单元测试。

关键文件：

- verl/checkpoint_engine/base.py（模块 检查点引擎；类别 source；类型 core-logic；符号 send_weights, receive_weights, update_weights）：抽象基类定义，核心接口变更的起点。修改了 CheckpointEngine.send_weights 和 receive_weights 抽象方法签名，并更新了 ColocatedCheckpointEngine 及 CheckpointEngineWorker 中的调用。
- verl/workers/engine_workers.py（模块 引擎工作器；类别 source；类型 core-logic；符号 ActorRolloutRefWorker.update_weights, CheckpointEngineWorker.update_weights）：调用侧关键改动，将 trainer 和 checkpoint worker 的 global_steps 实际传递到 checkpoint engine。
- tests/checkpoint_engine/test_global_steps_on_cpu.py（模块 传播测试；类别 test；类型 test-coverage；符号 _FakeTrainerEngine, init, get_per_tensor_param, _FakeCheckpointEngine）：新增 CPU 单元测试，验证 global_steps 从 trainer 传播到 checkpoint engine 以及从 checkpoint engine 传播到 rollout server adapter。
- verl/checkpoint_engine/nccl_checkpoint_engine.py（模块 检查点引擎；类别 source；类型 core-logic；符号 send_weights, receive_weights）：内置 NCCL 引擎，展示签名变更的模式。
- verl/checkpoint_engine/nixl_checkpoint_engine.py（模块 检查点引擎；类别 source；类型 core-logic；符号 send_weights, receive_weights）：内置 NIXL 引擎，同步签名。
- verl/checkpoint_engine/mooncake_checkpoint_engine.py（模块 检查点引擎；类别 source；类型 core-logic；符号 send_weights, receive_weights）：内置 Mooncake 引擎，同步签名。
- verl/checkpoint_engine/hccl_checkpoint_engine.py（模块 检查点引擎；类别 source；类型 core-logic；符号 send_weights, receive_weights）：内置 HCCL 引擎，同步签名。
- verl/checkpoint_engine/kimi_checkpoint_engine.py（模块 检查点引擎；类别 source；类型 core-logic；符号 send_weights, receive_weights）：内置 Kimi 引擎，同步签名。
- tests/checkpoint_engine/test_utils.py（模块 测试工具；类别 test；类型 test-coverage）：测试工具更新，确保测试框架兼容新参数。

关键符号：CheckpointEngine.send_weights, CheckpointEngine.receive_weights, ColocatedCheckpointEngine.send_weights, ColocatedCheckpointEngine.receive_weights, CheckpointEngineWorker.update_weights, ActorRolloutRefWorker.update_weights

关键源码片段

verl/checkpoint_engine/base.py

抽象基类定义，核心接口变更的起点。修改了 CheckpointEngine.send_weights 和 receive_weights 抽象方法签名，并更新了 ColocatedCheckpointEngine 及 CheckpointEngineWorker 中的调用。

```
# verl/checkpoint_engine/base.py

@abstractmethod
async def send_weights(
```

```
    self,
    weights: Generator[tuple[str, torch.Tensor], None, None],
    global_steps: int | None = None, # 训练步数，自定义后端可用于版本标识
):
    raise NotImplementedError

@abstractmethod
async def receive_weights(
    self,
    global_steps: int | None = None,
) -> Generator[tuple[str, torch.Tensor], None, None]:
    raise NotImplementedError

# ColocatedCheckpointEngine 中的实现（注意：同步 def，潜在异步不匹配风险）
class ColocatedCheckpointEngine(CheckpointEngine):
    def send_weights(
        self,
        weights: Generator[tuple[str, torch.Tensor], None, None],
        global_steps: int | None = None, # 参数被接收但不使用
    ):
        self.weights = weights

    def receive_weights(
        self,
        global_steps: int | None = None, # 忽略
    ):
        yield from self.weights
        self.weights = None

# CheckpointEngineWorker 中消费侧传递
@register(dispatch_mode=Dispatch.ONE_TO_ALL, blocking=False)
async def update_weights(self, global_steps: int = None):
    weights = self.checkpoint_engine.receive_weights(global_steps=global_steps)
    await self.server_adapter.update_weights(weights, global_steps=global_steps)
```

verl/workers/engine_workers.py

调用侧关键改动，将 trainer 和 checkpoint worker 的 global_steps 实际传递到 checkpoint engine。

```
# verl/workers/engine_workers.py

class ActorRolloutRefWorker:
    # 原签名已包含 global_steps（hydra 注入），现在转发给 checkpoint engine
    async def update_weights(self, global_steps: int, mode: str = "auto"):
        # ...
        # 将 global_steps 传递给 send_weights
        await self.checkpoint_engine.send_weights(
            self.actor.engine.get_per_tensor_param(),
            global_steps=global_steps
```

```
        )

    class CheckpointEngineWorker:
        # 原签名已包含 global_steps，现在转发给 receive_weights 和 server adapter
        async def update_weights(self, global_steps: int):
            weights = self.checkpoint_engine.receive_weights(global_steps=global_steps)
            await self.server_adapter.update_weights(weights, global_steps=global_steps)
```

tests/checkpoint_engine/test_global_steps_on_cpu.py

新增 CPU 单元测试，验证 global_steps 从 trainer 传播到 checkpoint engine 以及从 checkpoint engine 传播到 rollout server adapter。

```
    # tests/checkpoint_engine/test_global_steps_on_cpu.py

    from types import SimpleNamespace
    from verl.checkpoint_engine.base import CheckpointEngineWorker
    from verl.workers.engine_workers import ActorRolloutRefWorker

    class _FakeTrainerEngine:
        def __init__(self):
            self.weights = [("w", object())]
        def get_per_tensor_param(self):
            return iter(self.weights), None

    class _FakeCheckpointEngine:
        def __init__(self):
            self.sent_global_steps = None
            self.received_global_steps = None
            self.sent_weights = None
        async def send_weights(self, weights, global_steps=None):
            self.sent_global_steps = global_steps
            self.sent_weights = list(weights)
        def receive_weights(self, global_steps=None):
            self.received_global_steps = global_steps
            async def _weights():
                yield "w", object()
            return _weights()

    class _FakeServerAdapter:
        def __init__(self):
            self.global_steps = None
            self.weights = None
        async def update_weights(self, weights, global_steps=None):
            self.global_steps = global_steps
            self.weights = [item async for item in weights]

    def test_actor_worker_passes_global_steps_to_checkpoint_engine_send():
        checkpoint_engine = _FakeCheckpointEngine()
        # 使用 __new__ 绕过完整初始化
```

```
worker = ActorRolloutRefWorker.__new__(ActorRolloutRefWorker)
worker.config = SimpleNamespace(
    rollout=SimpleNamespace(
        checkpoint_engine=SimpleNamespace(backend="modelexpress"),
    ),
)
worker.actor = SimpleNamespace(engine=_FakeTrainerEngine())
worker.checkpoint_engine = checkpoint_engine

asyncio.run(
    ActorRolloutRefWorker.update_weights.__wrapped__(
        worker, global_steps=17, mode="auto"
    )
)

assert checkpoint_engine.sent_global_steps == 17
assert checkpoint_engine.sent_weights == worker.actor.engine.weights
```

评论区精华

gemini-code-assist[bot] 指出 ColocatedCheckpointEngine.send_weights 被定义为同步方法 (def) 而非异步 (async def)，而抽象基类中该方法标记为 async def。在多态调用时（如 test_utils.py 中 await 调用），会导致 TypeError: object NoneType can't be used in 'await' expression。该评论在 PR 中未获得回应或修正，但 PR 被批准合并。这留下了潜在的正确性问题。

- ColocatedCheckpointEngine.send_weights 缺少 async 关键字 (correctness): 问题未被解决或回应，PR 已合并。后续可能需要修复。

风险与影响

- 风险：
  - 异步接口不一致：ColocatedCheckpointEngine.send_weights 是同步方法，若在 async 上下文中被 await 调用会引发 TypeError。当前所有内置调用场景可能未触发此问题（colocated 引擎在同步路径下使用），但自定义后端如果直接使用抽象接口则可能暴露。
  - 测试覆盖不足：仅提供 CPU 单元测试，缺少 GPU 集成测试验证 train-rollout 完整路径中的 global_steps 传播。
  - 自定义后端适配压力：第三方实现 CheckpointEngine 抽象类的后端需要添加新参数（虽然默认为 None，但仍需修改方法签名以通过类型检查）。
  - 无性能风险：参数可选且内置引擎忽略，运行时开销忽略不计。
- 影响：
  - 对用户：自定义 checkpoint 后端开发者现在可以在 send_weights/receive_weights 中获取训练步数（global_steps），用于版本控制或日志。内置引擎用户无感知。
  - 对系统：无行为变化，因为参数可选且未被内置引擎使用。
  - 对团队：需要通知自定义后端维护者更新实现以兼容新签名，但提供了一段兼容窗口。

- 风险标记：异步接口不一致（ColocatedCheckpointEngine），缺少 GPU 集成测试，自定义后端需适配新参数

关联脉络

- 暂无明显关联 PR