PR #7162 完整报告

verl-project/verl

[perf] fix: Prevent creating new threads/event loop for each `tqbridge` call

合并时间：2026-07-30 12:02

原文链接：http://prhub.com.cn/verl-project/verl/pull/7162

执行摘要

- 一句话：阻止每次 tqbridge 调用创建新线程和事件循环
- 推荐动作：建议所有使用 TransferQueue 的开发者精读此 PR，特别是 transferqueue_utils.py 中单例事件循环的设计和 atexit 注册清理的模式。配置抽取也展示了如何通过 Hydra defaults 实现模块化配置，值得参考。

功能与动机

PR #7157 修复了 TransferQueue 临时事件循环的 FD 泄漏问题，但其每次调用都新建线程和事件循环的机制本身仍有性能开销。本 PR 旨在进一步优化，通过复用单例事件循环来提升性能，并完善资源生命周期管理。

实现拆解

1. 引入全局共享事件循环和线程：在 verl/utils/transferqueue_utils.py 中添加模块级变量 _ASYNC_BRIDGE_LOOP、_ASYNC_BRIDGE_THREAD 和 _ASYNC_BRIDGE_LOCK，用于持有单例事件循环和线程。新增 _run_event_loop_forever 作为后台线程目标，负责设置事件循环并运行；_get_async_bridge_loop 使用双重检查锁定模式惰性初始化全局循环；_shutdown_async_bridge_runtime 负责优雅停止循环和线程，并通过 atexit 注册确保进程退出时清理。
2. 改造 _run_async_in_temp_loop：移除原先每次创建临时事件循环和线程的逻辑，改为调用 _get_async_bridge_loop 获取共享循环，然后通过 asyncio.run_coroutine_threadsafe 提交协程并等待结果。简化后的函数不再需要在 finally 块中停止循环。
3. 重构 TransferQueue 配置：创建新文件 verl/trainer/config/transfer_queue/transfer_queue.yaml 承载 TransferQueue 的独立配置，并从 ppo_trainer.yaml 中移除内联的 transfer_queue 段落，改为通过 defaults 列表中的 - transfer_queue@transfer_queue: transfer_queue 引用。同时更新了所有生成的配置文件（_generated_*_trainer.yaml）以反映这一变更。
4. 添加测试覆盖：在 tests/utils/test_transferqueue_utils_on_cpu.py 中新增 test_async_bridge_loop_reused_between_calls，验证多次调用共享同一个线程和事件循环；同时调整现有测试 test_temp_event_loop_releases_file_descriptors，在测试前后调用 _shutdown_async_bridge_runtime 以隔离状态。

5. 配套调整：更新 docs/data/transfer_queue.md 文档以反映配置变化；调整 tests/special_sanity/test_config_docs.py 确保配置文档完整性。

关键文件：

- verl/utils/transferqueue_utils.py（模块 工具库；类别 source；类型 dependency-wiring；符号 _run_event_loop_forever, _shutdown_async_bridge_runtime, _get_async_bridge_loop, run_coroutine）：核心实现文件，引入了全局共享事件循环和线程，重构了异步桥接运行时生命周期管理。
- tests/utils/test_transferqueue_utils_on_cpu.py（模块 单元测试；类别 test；类型 test-coverage；符号 test_async_bridge_loop_reused_between_calls）：新增测试确保事件循环和线程重用，验证没有泄漏。
- verl/trainer/config/transfer_queue/transfer_queue.yaml（模块 配置；类别 config；类型 configuration）：新创建的独立配置文件，承载 TransferQueue 的所有配置项。
- verl/trainer/config/ppo_trainer.yaml（模块 配置；类别 config；类型 configuration）：配置文件结构调整，将内联的 transfer_queue 配置移除，改为通过 defaults 引用独立文件。

关键符号：_run_event_loop_forever, _shutdown_async_bridge_runtime, _get_async_bridge_loop, _run_async_in_temp_loop

关键源码片段

verl/utils/transferqueue_utils.py

核心实现文件，引入了全局共享事件循环和线程，重构了异步桥接运行时生命周期管理。

```
import asyncio
import threading
import atexit

# 全局共享异步桥接循环、线程和锁
_ASYNC_BRIDGE_LOOP: asyncio.AbstractEventLoop | None = None
_ASYNC_BRIDGE_THREAD: threading.Thread | None = None
_ASYNC_BRIDGE_LOCK = threading.Lock()


def _run_event_loop_forever(loop: asyncio.AbstractEventLoop, ready: threading.Event) -> None:
    """后台线程目标：设置事件循环并永久运行"""
    asyncio.set_event_loop(loop)
    ready.set()
    loop.run_forever()


def _shutdown_async_bridge_runtime(
    loop: asyncio.AbstractEventLoop | None = None,
    thread: threading.Thread | None = None,
) -> None:
    """关闭共享异步桥接运行时，可指定或使用全局状态"""
    global _ASYNC_BRIDGE_LOOP, _ASYNC_BRIDGE_THREAD
```

```
    if loop is None and thread is None:
        # 未指定时，清理全局状态并清空引用
        with _ASYNC_BRIDGE_LOCK:
            loop = _ASYNC_BRIDGE_LOOP
            thread = _ASYNC_BRIDGE_THREAD
            _ASYNC_BRIDGE_LOOP = None
            _ASYNC_BRIDGE_THREAD = None
    if thread is not None and thread.is_alive():
        try:
            loop.call_soon_threadsafe(loop.stop)
        except RuntimeError:
            # 进程退出竞争时直接忽略
            pass
        thread.join()
    if loop is not None and not loop.is_closed():
        loop.close()


def _get_async_bridge_loop() -> asyncio.AbstractEventLoop:
    """惰性初始化并返回共享事件循环（双重检查锁定）"""
    global _ASYNC_BRIDGE_LOOP, _ASYNC_BRIDGE_THREAD
    with _ASYNC_BRIDGE_LOCK:
        loop = _ASYNC_BRIDGE_LOOP
        if loop is not None:
            return loop
        # 首次调用：创建新循环和线程
        ready = threading.Event()
        loop = asyncio.new_event_loop()
        thread = threading.Thread(
            target=_run_event_loop_forever,
            args=(loop, ready),
            name="batchmeta tensordict converter",
            daemon=True,
        )
        thread.start()
        if not ready.wait(timeout=5):
            _shutdown_async_bridge_runtime(loop, thread)
            raise RuntimeError("Failed to start TransferQueue async bridge event loop.")
        _ASYNC_BRIDGE_LOOP = loop
        _ASYNC_BRIDGE_THREAD = thread
        return loop


atexit.register(_shutdown_async_bridge_runtime)


def _run_async_in_temp_loop(async_func, *args, **kwargs) -> Any:
    """在共享后台循环中执行异步函数（旧接口兼容）"""
    loop = _get_async_bridge_loop()
```

```
    coroutine = async_func(*args, **kwargs)
    try:
        future = asyncio.run_coroutine_threadsafe(coroutine, loop)
    except RuntimeError:
        coroutine.close()
        raise
    return future.result()
```

tests/utils/test_transferqueue_utils_on_cpu.py

新增测试确保事件循环和线程重用，验证没有泄漏。

```
from verl.utils import transferqueue_utils as tqu

async def _noop():
    return None

def test_async_bridge_loop_reused_between_calls():
    tqu._shutdown_async_bridge_runtime() # 确保初始状态干净
    try:
        tqu._run_async_in_temp_loop(_noop)
        first_thread = tqu._ASYNC_BRIDGE_THREAD
        first_loop = tqu._ASYNC_BRIDGE_LOOP

        assert first_thread is not None
        assert first_loop is not None
        assert first_thread.is_alive()
        assert not first_loop.is_closed()

        # 第二次调用应复用相同的线程和循环
        tqu._run_async_in_temp_loop(_noop)
        assert tqu._ASYNC_BRIDGE_THREAD is first_thread
        assert tqu._ASYNC_BRIDGE_LOOP is first_loop
    finally:
        tqu._shutdown_async_bridge_runtime()
```

评论区精华

主要讨论包括：

- 配置抽取：审核者 wuxibin89 建议将 TransferQueue 配置移到单独 YAML 文件，以保持主配置文件清晰，该建议已被采纳。
- 重试逻辑：审核者问为什么在 _get_async_bridge_loop 中有 2 次重试，作者解释是为了应对线程 / 循环意外退出的场景，但随后同意可以移除，最终提交简化为无重试的版本。
- 每次调用 stop/start 问题：审核者质疑为什么每次调用都停止和启动循环，作者澄清这已经是旧行为，新实现只启动一次，不会重复停止 / 启动。
  - TransferQueue 配置抽取 (design): 已采纳，创建独立文件并通过 Hydra defaults 引用。
  - 异步桥接循环重试逻辑 (design): 重试逻辑被移除，最终实现仅一次初始化。

- 每次调用 stop/start 循环 (performance): 已纠正理解，新实现不会重复停止 / 启动。

风险与影响

- 风险:
  1. 全局状态竞态：引入全局共享事件循环和线程后，如果多个线程同时调用 _run_async_in_temp_loop，虽然有锁保护初始化，但协程调度仍可能竞争。当前设计假定调用是线程安全的（通过 asyncio.run_coroutine_threadsafe），但协程内部共享状态可能需额外注意。
  2. atexit 清理时机：atexit 注册的清理函数在进程退出时执行，但如果程序因信号强制终止，可能不会执行。此外，在 fork 后的子进程中，全局状态可能残留父进程的循环，需确保子进程重新初始化。
  3. 配置兼容性：TransferQueue 配置从内联移至单独文件，现有用户如果直接修改了 ppo_trainer.yaml 中的 transfer_queue 字段，升级后需迁移到新文件。不过 Hydra 组合机制会对默认值合并，如果用户自定义了 ppo_trainer.yaml 但未修改 defaults 列表，仍会加载新文件的默认配置，可能有意外行为。 - 影响：对用户：功能透明，所有使用 tqbridge 装饰的调用自动受益于共享事件循环，性能改善并降低 FD 泄漏风险。对系统：减少线程创建 / 销毁开销，降低系统资源消耗。对团队：配置结构更清晰，便于维护和扩展。影响范围限定在使用 TransferQueue 模块的训练流程中。 - 风险标记：全局共享事件循环竞态，atexit 清理顺序，配置迁移兼容性

关联脉络

- PR #7157 [worker] fix: close temporary TransferQueue event loop: 本 PR 是基于 #7157 修复后的进一步优化，解决临时事件循环 FD 泄漏并提升性能。