PR #7099 完整报告

verl-project/verl

[perf] feat: re-support torch profiler and optimize output naming

合并时间：2026-07-21 15:58

原文链接：http://prhub.com.cn/verl-project/verl/pull/7099

执行摘要

- 一句话：重新支持 torch profiler 并优化 trace 文件命名
- 推荐动作：值得精读，特别是 profiler 调度设计（TorchProfilerScheduleConfig）和多角色文件名生成策略（build_trace_basename）的实现。

功能与动机

重新启用 torch profiler，解决之前无法使用的问题，并引入 step 调度以控制 trace 文件大小，优化输出文件名以便于多角色多 rank 场景下的分析。

实现拆解

1. 新增 TorchProfilerScheduleConfig 配置类，封装 skip_first、wait、warmup、active、repeat 参数，并提供 enabled 属性和 to_torch_kwargs 方法；更新 TorchProfilerToolConfig，新增可选的 schedule 字段。
2. 在 verl/utils/profiler/torch_profile.py 中新增 get_dist_topology、build_trace_basename 等辅助函数，用于收集分布式拓扑信息并构建自描述的 trace 文件名；重构 get_torch_profiler 函数，支持传递 schedule 参数。
3. 在 DistProfiler 基类和 Profiler（torch profiler 实现）中新增 step() 方法，用于推进 profiler 调度；同时为其他 profiler 实现（NPUProfiler、NsightSystemsProfiler）添加空的 step() 方法以确保接口一致性。
4. 在 DistProfiler.__init__ 中添加 save_file_prefix 参数，用于在 trace 文件名中嵌入角色标识；在各个 trainer（ray_trainer、trainer_base、separation/ray_trainer 等）的 mini-batch 循环中插入 profiler.step() 调用，使 step 调度生效。
5. 更新文档 docs/perf/torch_profiling.md，说明 schedule 字段的用法；新增大量单元测试覆盖 step 行为、异常处理（discrete annotate 异常停止）等。

关键文件：

- verl/utils/profiler/torch_profile.py（模块 Profiler 核心；类别 source；类型 core-logic；符号 get_dist_topology, _sanitize_name_part, build_trace_basename, _resolve_schedule_kwargs）：核心修改：新增分布式拓扑发现、trace 文件名生成、调度支持等功能
- verl/utils/profiler/config.py（模块 配置定义；类别 source；类型 core-logic；符号 TorchProfilerScheduleConfig, post_init, enabled, to_torch_kwargs）：新增

TorchProfilerScheduleConfig 配置类，定义 step 调度参数

- tests/utils/test_torch_profile.py（模块 测试；类别 test；类型 test-coverage；符号 tearDown, test_discrete_annotate_stops_profiler_on_exception, boom, test_dist_annotate_propagates_and_runs_func_once）：大量单元测试覆盖新增功能，尤其是异常处理和 step 行为
- verl/utils/profiler/profile.py（模块 Profiler 调度器；类别 source；类型 core-logic；符号 step, step_profile）：DistProfiler 基类修改：新增 step 方法、save_file_prefix 参数
- verl/workers/engine_workers.py（模块 引擎 Worker；类别 source；类型 core-logic）：在 mini-batch 循环中插入 profiler.step() 调用，集成调度
- verl/trainer/ppo/ray_trainer.py（模块 Trainer；类别 source；类型 core-logic）：trainer 集成 profiler.step() 调用

关键符号：get_dist_topology, _sanitize_name_part, build_trace_basename, _resolve_schedule_kwargs, TorchProfilerScheduleConfig, enabled, to_torch_kwargs, DistProfiler.step, Profiler.step, NPUProfiler.step, NsightSystemsProfiler.step, step_profile

关键源码片段

verl/utils/profiler/torch_profile.py

核心修改：新增分布式拓扑发现、trace 文件名生成、调度支持等功能

```
def build_trace_basename(
    rank: int,
    role: Optional[str] = None,
    save_file_prefix: Optional[str] = None,
    topology: Optional[dict] = None,
) -> str:
    """Build a descriptive, per-process trace filename stem.

    Encodes -- when available -- the worker role (``save_file_prefix``, e.g. ``actor``),
    the profiling scope role (``role``, e.g. ``e2e``), the global rank and world size,
    and the tensor/pipeline/data/context parallel ranks, followed by pid and a
    timestamp so that files written by different processes never collide.
    """
    topology = get_dist_topology() if topology is None else topology
    current_time = datetime.now(tz=timezone.utc).astimezone()
    timestamp = current_time.strftime("%Y%m%d%H%M%S%f")[:-3]
    pid = os.getpid()

    parts: list[str] = []
    if save_file_prefix:
        parts.append(_sanitize_name_part(save_file_prefix))
    if role:
        parts.append(_sanitize_name_part(role))

    global_rank = topology.get("rank", rank)
```

```
    world_size = topology.get("world_size")
    rank_part = f"rank{global_rank}"
    if world_size:
        rank_part += f"-of-{world_size}"
    parts.append(rank_part)

    parallel_part = "-".join(f"{dim}{topology[dim]}" for dim in ("tp", "pp", "dp", "cp") if dim in
    topology)
    if parallel_part:
        parts.append(parallel_part)

    parts.append(f"pid{pid}")
    parts.append(timestamp)
    return "_".join(parts)


# 辅助函数用于清理文件名中的特殊字符
def _sanitize_name_part(text: str) -> str:
    """Make an arbitrary label safe to embed in a filename."""
    return re.sub(r"[^0-9A-Za-z.=+-]+", "-", str(text)).strip("-")


# 收集分布式拓扑信息（rank / world_size / tp / pp / dp / cp）
def get_dist_topology() -> dict:
    info: dict = {}
    try:
        import torch.distributed as dist
        if dist.is_available() and dist.is_initialized():
            info["rank"] = dist.get_rank()
            info["world_size"] = dist.get_world_size()
    except Exception:
        pass
    try:
        from megatron.core import parallel_state as mpu
        if mpu.model_parallel_is_initialized():
            info["tp"] = mpu.get_tensor_model_parallel_rank()
            info["pp"] = mpu.get_pipeline_model_parallel_rank()
            info["dp"] = mpu.get_data_parallel_rank()
            try:
                info["cp"] = mpu.get_context_parallel_rank()
            except Exception:
                pass
    except Exception:
        pass
    return info
```

verl/utils/profiler/config.py

新增 TorchProfilerScheduleConfig 配置类，定义 step 调度参数

```
@dataclass
class TorchProfilerScheduleConfig(BaseConfig):
    """Schedule for ``torch.profiler.schedule``.

    Field names mirror the official ``torch.profiler.schedule`` API. The profiler
    cycles through ``skip_first`` -> (``wait`` -> ``warmup`` -> ``active``) x ``repeat``.
    Scheduling is only enabled when ``active > 0``; otherwise the profiler runs in
    continuous mode (collect everything between start and stop).
    """

    # Number of steps to skip at the very beginning (not counted in the cycle).
    skip_first: int = 0
    # Number of steps to idle (no collection) at the start of each cycle.
    wait: int = 0
    # Number of steps to warm up (tracing on, data discarded) each cycle.
    warmup: int = 0
    # Number of steps to actively record each cycle. <= 0 disables scheduling.
    active: int = 0
    # Number of cycles to repeat. 0 means repeat until profiling stops.
    repeat: int = 0
    name: str = "torch_schedule"

    def __post_init__(self) -> None:
        """Validate all fields are non-negative integers."""
        for field_name in ("skip_first", "wait", "warmup", "active", "repeat"):
            value = getattr(self, field_name)
            assert isinstance(value, int), f"{field_name} must be int, got {type(value)}"
            assert value >= 0, f"{field_name} must be >= 0, got {value}"

    @property
    def enabled(self) -> bool:
        """Scheduling only takes effect when at least one active step is requested."""
        return self.active > 0

    def to_torch_kwargs(self) -> dict:
        """Return kwargs for ``torch.profiler.schedule``."""
        return {
            "skip_first": self.skip_first,
            "wait": self.wait,
            "warmup": self.warmup,
            "active": self.active,
            "repeat": self.repeat,
        }
```

tests/utils/test_torch_profile.py

大量单元测试覆盖新增功能，尤其是异常处理和 step 行为

```
@patch("verl.utils.profiler.torch_profile.get_torch_profiler")
def test_discrete_annotate_stops_profiler_on_exception(self, mock_get_profiler):
```

```
    # A stage raising inside a discrete-mode annotate must still stop the
    # (process-global) torch profiler; otherwise it leaks, the next stage's
    # start() fails with "Profiler is already enabled" and the process aborts.
    mock_prof_instance = MagicMock()
    mock_get_profiler.return_value = mock_prof_instance

    tool_config = TorchProfilerToolConfig(contents=["cpu"], discrete=True)
    config = ProfilerConfig(save_path="/tmp/test", enable=True, tool_config=tool_config)
    profiler = Profiler(rank=0, config=config, tool_config=tool_config)

    calls = {"n": 0}

    @profiler.annotate(role="boom")
    def boom():
        calls["n"] += 1
        raise RuntimeError("stage failed on purpose")

    with self.assertRaises(RuntimeError):
        boom()

    # Profiler must be started and, crucially, stopped despite the exception,
    # and the stage body must run exactly once (no re-execution).
    mock_prof_instance.start.assert_called_once()
    mock_prof_instance.stop.assert_called_once()
    self.assertEqual(calls["n"], 1)
```

评论区精华

- 关于 transformers 版本限制的讨论：wuxibin89 认为不应限制 transformers<5.6.0，ETOgaosion 提议改为 <5.12，最终未确定具体版本但方向一致。
- schedule 设计问题：mengchengTang 指出 schedule 在 discrete=False 时第一个 step 覆盖不干净，且 ref 角色缺少 step 调用；ETOgaosion 承认是设计 bug 并承诺修复。
- step API 语义：tardis-key 询问 mstx 中的 step 与 torch profiler 的 step 语义是否一致，mengchengTang 和 ETOgaosion 最终认为 step 是 mini-batch 级别的公共接口。
- transformers 版本限制 (question): 暂时放宽限制，但最终版本未确定，需要进一步验证兼容性。
- schedule 与 discrete 模式交互 (design): 确认需要修复，scheduled collection 应仅与 discrete 模式配合使用，ref 也需要插入 step。
- step API 在不同 profiler 中的语义 (design): step 被确认为公共接口，各 profiler 实现保持空实现或兼容实现。

风险与影响

- 风险：涉及多个 trainer 核心循环的修改，可能影响训练流程的稳定性；step() 方法在多个 profiler 实现中需要保持一致，否则可能引发 AttributeError（如 NPUProfiler 之前没有 step 导致崩溃，已在测试中回归）；transformers 版本限制的修改可能影响依赖兼容性。

- 影响：用户可以通过配置 schedule 更精细地控制 profiler，减少 trace 文件体积；多角色（actor/critic/ref）的 trace 文件通过自描述文件名可区分，方便分析；团队需要维护新增的 schedule 配置和 step 接口，以及多 profiler 实现的一致性。
- 风险标记：核心训练循环修改，多 profiler 接口兼容，依赖版本限制

关联脉络

- 暂无明显关联 PR