PR #52281 完整报告

vllm-project/vllm

[ROCm] Give EngineCore cleanup grace after request abort

合并时间：2026-08-20 00:40

原文链接：http://prhub.com.cn/vllm-project/vllm/pull/52281

执行摘要

- 一句话：给 ROCm EngineCore 关闭增加 15 秒清理宽限
- 推荐动作：建议关注该 PR 的双层超时解耦设计与 gc.freeze() 应用边界。它对理解 vLLM V1 EngineCore 进程生命周期、ROCm teardown 特殊性以及 Python 解释器最终化阶段的 GC 行为都有参考价值。若团队维护 ROCm 或需要自定义进程退出宽限逻辑，值得精读。

功能与动机

AMD CI build 12007 暴露了 shutdown race：请求已完成，但 shutdown_timeout=0 让父进程在发送 SIGTERM 后不预留任何清理时间就 force-kill EngineCore，ROCm 的 executor、scheduler、distributed 与 cache 清理没有机会执行，导致 GPU 内存停留在 11.72 GB，并持续占用整个 120 秒 release wait。PR body 明确指出 zero timeout 语义只是“立即中止在途请求”，父进程仍需要独立窗口让 EngineCore 释放设备资源。

实现拆解

1. 在 vllm/v1/engine/utils.py 中新增常量 ROCM_ENGINE_PROCESS_SHUTDOWN_TIMEOUT_S = 15.0 与函数 get_engine_process_shutdown_timeout(request_timeout, process_timeout)：仅当 request_timeout 与 process_timeout 同时为 0 且 current_platform.is_rocm() 时返回 15.0，其余情况原样返回 process_timeout。这样既给 ROCm 零超时路径提供清理窗口，又不破坏正数请求排空超时预算。
2. CoreEngineProcManager.__init__保存 self._request_shutdown_timeout = vllm_config.shutdown_timeout，并在 shutdown() 中调用 get_engine_process_shutdown_timeout 解耦计算进程清理超时；当计算值与调用方传入值不一致时记录日志，再调用 shutdown(self.processes, timeout=process_timeout)。
3. 在 vllm/v1/engine/core.py 的 EngineCoreProc.run_engine_core 中增加 clean_shutdown 判定：SystemExit 退出码为 None/0、shutdown_state 为 SHUTTING_DOWN、has_work() 为 False、shutdown_timeout 为 0。满足条件且平台为 ROCm 时，在 finally 中执行完 engine_core.shutdown() 后调用 gc.freeze()，把存活对象图冻结在即将终止的专用进程内，避免解释器 finalization 阶段的慢速循环 GC 扫描。
4. vllm/entrypoints/openai/dp_supervisor.py 的 _shutdown_children 复用 get_engine_process_shutdown_timeout(self.args.shutdown_timeout, self.args.shutdown_timeout)，再叠加 CHILD_EXIT_GRACE_S，使 DP 场景下 ROCm 零路径最多等待 20 秒。

5. 测试配套：tests/v1/engine/test_startup_watch_processes.py 新增两个参数化测试，覆盖 5 种超时组合与 6 种 GC 冻结场景；tests/entrypoints/openai/test_dp_supervisor.py 新增 test_shutdown_children_uses_engine_process_timeout 验证 DP supervisor 的超时叠加逻辑。

关键文件：

- vllm/v1/engine/utils.py（模块 进程管理；类别 source；类型 core-logic；符号 get_engine_process_shutdown_timeout, CoreEngineProcManager）：核心逻辑所在：新增 get_engine_process_shutdown_timeout，将请求排空超时与进程清理超时解耦，并在 CoreEngineProcManager.shutdown() 中接入 ROCm 15 秒宽限。
- vllm/v1/engine/core.py（模块 引擎核心；类别 source；类型 dependency-wiring；符号 run_engine_core, clean_shutdown）：EngineCore 子进程入口：新增 clean_shutdown 判定，并在干净退出且 ROCm 时调用 gc.freeze()，避免终止进程 finalization 阶段重复循环 GC。
- vllm/entrypoints/openai/dp_supervisor.py（模块 数据并行；类别 source；类型 dependency-wiring；符号 _shutdown_children）：DP supervisor 复用同一 EngineCore 进程超时并叠加 CHILD_EXIT_GRACE_S，保证多进程场景下 ROCm 零路径等待不超过 20 秒。
- tests/v1/engine/test_startup_watch_processes.py（模块 进程测试；类别 test；类型 test-coverage；符号 test_engine_core_process_shutdown_timeout, test_freeze_gc_after_clean_rocm_engine_core_shutdown）：新增两个参数化测试，覆盖超时解耦的 5 种组合与 gc.freeze 的 6 种边界场景，是本次变更的主要质量保障。
- tests/entrypoints/openai/test_dp_supervisor.py（模块 监督测试；类别 test；类型 test-coverage；符号 test_shutdown_children_uses_engine_process_timeout）：验证 DP supervisor 正确复用 EngineCore 进程超时并叠加 CHILD_EXIT_GRACE_S，防止回归。

关键符号：get_engine_process_shutdown_timeout, CoreEngineProcManager.shutdown, EngineCoreProc.run_engine_core, DPSupervisor._shutdown_children, test_engine_core_process_shutdown_timeout, test_freeze_gc_after_clean_rocm_engine_core_shutdown, test_shutdown_children_uses_engine_process_timeout

关键源码片段

vllm/v1/engine/utils.py

核心逻辑所在：新增 get_engine_process_shutdown_timeout，将请求排空超时与进程清理超时解耦，并在 CoreEngineProcManager.shutdown() 中接入 ROCm 15 秒宽限。

```
# vllm/v1/engine/utils.py
# 常量：ROCm 专用清理宽限窗口（秒）
ROCM_ENGINE_PROCESS_SHUTDOWN_TIMEOUT_S = 15.0


def get_engine_process_shutdown_timeout(
    request_timeout: float | None,
```

```
    process_timeout: float | None,
) -> float | None:
    # request_timeout 即 VllmConfig.shutdown_timeout，为 0 表示收到 SIGTERM 后立即
    # 中止在途请求；但父进程仍需独立窗口让 EngineCore 释放设备资源，ROCm teardown
    # 更慢，force-kill 可能残留 VRAM。
    # process_timeout 可能是外层进程管理器传下的剩余预算：除非两者同时为 0，否则
    # 保持原值——正数请求排空超时耗尽后的零预算不应获得新的宽限期。
    if request_timeout == 0 and process_timeout == 0 and current_platform.is_rocm():
        return ROCM_ENGINE_PROCESS_SHUTDOWN_TIMEOUT_S
    return process_timeout


class CoreEngineProcManager:
    # 关键集成点：shutdown() 将请求排空超时与进程清理超时解耦
    def shutdown(self, timeout: float | None = None) -> None:
        self.manager_stopped.set()
        if self._finalizer.detach() is not None:
            process_timeout = get_engine_process_shutdown_timeout(
                self._request_shutdown_timeout, timeout
            )
            if process_timeout != timeout:
                logger.info(
                    "[shutdown] EngineCore process manager: using %ss ROCm "
                    "cleanup grace after immediate request abort",
                    process_timeout,
                )
            shutdown(self.processes, timeout=process_timeout)
```

vllm/v1/engine/core.py

EngineCore 子进程入口：新增 clean_shutdown 判定，并在干净退出且 ROCm 时调用 gc.freeze()，避免终止进程 finalization 阶段重复循环 GC。

```
# vllm/v1/engine/core.py EngineCoreProc.run_engine_core 中的关键分支
        except SystemExit as e:
            logger.info_once("[shutdown] EngineCore: exiting busy loop")
            # 仅对“干净退出”启用后续优化：退出码成功、已进入 SHUTTING_DOWN、
            # EngineCore 无剩余工作、且配置的 shutdown_timeout 为 0
            clean_shutdown = (
                e.code in (None, 0)
                and engine_core is not None
                and engine_core.shutdown_state == EngineShutdownState.SHUTTING_DOWN
                and not engine_core.has_work()
                and engine_core.vllm_config.shutdown_timeout == 0
            )
            raise
        # ... 中间为 run_busy_loop、信号处理等既有逻辑（未改动）...
        finally:
            signal.signal(signal.SIGTERM, signal.SIG_DFL)
            signal.signal(signal.SIGINT, signal.SIG_DFL)
```

```
if signal_callback is not None:
    signal_callback.stop()
if engine_core is not None:
    engine_core.shutdown()
if clean_shutdown:
    from vllm.platforms import current_platform
    if current_platform.is_rocm():
        # 上面的清理已解冻并回收堆对象；冻结存活对象图可跳过解释器
        # finalization 阶段的慢速循环 GC 扫描；进程退出时统一回收
        gc.freeze()
```

评论区精华

该 PR 没有实质性的 review 评论（review comments 为 0），作者通过多次 /ci run 触发 Buildkite CI 验证，DarkLight1337 直接批准。核心设计权衡记录在 PR body 中：正数请求排空超时耗尽的零预算不应获得新的宽限期，因为 EngineCore 依赖该 deadline 强制排空请求；GC 冻结只允许出现在终止的子进程路径，失败、未完成工作、正数超时与非 ROCm 路径均不冻结 GC 图。

- shutdown_timeout=0 下 ROCm teardown 竞态 (design): 引入 get_engine_process_shutdown_timeout，仅在 ROCm 且请求超时与外层预算同时为 0 时给予 15 秒清理宽限。
- 正数 timeout 预算耗尽不享宽限 (correctness): 保留原语义，仅在 0/0 的 ROCm 路径注入 15 秒宽限。
- gc.freeze 的适用边界 (performance): 在干净退出且 ROCm 时冻结存活对象图，进程退出即回收，避免 finalization 阶段慢速循环 GC。
- DP supervisor 是否沿用同一宽限 (design): 复用 get_engine_process_shutdown_timeout 并叠加 CHILD_EXIT_GRACE_S。

风险与影响

- 风险：
  1. 行为变更面很窄：仅 ROCm + shutdown_timeout=0 + 外层 process_timeout=0 时生效，非 ROCm 路径完全不变，正数与 None 预算也保持不变。
  2. 进程退出延迟：ROCm 零路径下 EngineCore 最长多存活 15 秒，DP 场景最长 20 秒；对追求极速释放进程资源的场景可能引入可感知延迟，但换取的是避免 120 秒显存残留。
  3. 新增风险点：clean_shutdown 判定在 except SystemExit 分支内调用 has_work()，若 has_work() 抛出异常，会覆盖原 SystemExit(0) 并改变进程退出形态，可能被父进程误判为异常退出。
  4. gc.freeze() 是进程终止专用优化，不禁止引用计数销毁，循环引用仅存在于该专用进程内，进程退出即回收，外部无内存风险。
  5. DP supervisor 新增 assert process_timeout is not None，若 shutdown_timeout 配置为 None，错误形态由原 TypeError 变为 AssertionError，行为略有变化。 - 影响：用户影响：ROCm 用户配置 shutdown_timeout=0 时，EngineCore 进程最长延迟 15

秒退出，但能保证 ROCm teardown 完成并释放 GPU 显存，避免 11.72 GB 显存残留持续 120 秒；非 ROCm 用户无感知。系统影响：EngineCore 进程生命周期管理中的请求排空与进程清理两阶段语义被显式分离，后续其他平台也可复用该模式。团队影响：涉及 v1 引擎核心进程管理与 DP supervisor 两条路径，测试覆盖较完整，便于后续演进。- 风险标记：ROCm 专属路径，进程退出延迟最多 15 秒，SystemExit handler 内新增 has_work() 调用，gc.freeze 为进程终止专用优化，DP supervisor 断言超时非 None

关联脉络

- PR #50809 [Bugfix][V1] Sync mamba_block_size via EngineCoreReadyResponse: 同属 v1 EngineCore 进程生命周期相关改动，涉及 vllm/v1/engine/core.py 与引擎握手 / 关闭路径。
- PR #52041 [Core] Skip broadcasting mm tensor data to workers for prefix-cache-covered items: 同为 v1 引擎核心调度与输出路径的性能 / 生命周期优化，关注 EngineCore 与 worker 之间的资源处理。