PR #44612 完整报告

vllm-project/vllm

[ASR] Optimize CPU preproc to get 2.5x RTFx via multi-threading

合并时间：2026-06-12 12:05

原文链接：http://prhub.com.cn/vllm-project/vllm/pull/44612

执行摘要

- 一句话：多线程预处理使 ASR 长音频 RTFx 提升 2.5 倍
- 推荐动作：值得精读。关键设计决策包括：独立于 Renderer 的线程池（对比测试显示独立池性能更好）、使用 Semaphore 限制并发以便取消、环境变量默认值由实验驱动。如有类似 CPU 预处理阻塞事件循环的场景，可参考此模式。

功能与动机

ASR 的 CPU 预处理（音频加载、RMS 分块）是同步代码，阻塞了 FastAPI 服务器的主事件循环，导致串行执行而非批量处理。高并发下还会引起 health/ 端点超时，难以识别故障节点并导致过度自动扩缩容。参考 PR body："CPU preprocessing like audio loading and RMS chunking is a synchronous code which blocks the main eventloop of FastAPI server which leads to serial execution instead of batched processing."

实现拆解

1. 定义环境变量（vllm/envs.py）：新增 VLLM_MAX_AUDIO_PREPROCESS_WORKERS，默认值通过 max(1, min(os.cpu_count(), 2)) 计算，实验证明 2 线程综合最优。
2. 新增带信号量的异步包装器（vllm/utils/async_utils.py）：新增 make_async_with_semaphore 函数，它接受一个同步函数和一个 ThreadPoolExecutor，返回一个 async 函数。内部使用 asyncio.Semaphore 限制并发数（等于线程池大小），使得请求取消时可以尽早拒绝排队任务。
3. 改造 ASR 服务端（vllm/entrypoints/speech_to_text/base/serving.py）：在 OpenAISpeechToText.__init__ 中根据环境变量创建 ThreadPoolExecutor（线程名前缀 stt-preprocess），并用 make_async_with_semaphore 包装原有的 _decode_and_chunk_speech 方法，赋值给 self._decode_and_chunk_speech_async。新增 shutdown 方法安全关闭线程池。
4. 确保生命周期清理（vllm/entrypoints/serve/utils/server_utils.py）：在 FastAPI 的 lifespan 函数的 finally 块中遍历 app.state 中的 openai_serving_transcription 和 openai_serving_translation 对象，若存在 shutdown 方法则调用，保证线程池在服务器关闭时被正确清理。
5. 测试配套：依赖 PR #44587 引入的长音频正确性测试 test_long_audio_wer_correctness，该测试会门控本变更的代码路径。

关键文件：

- vllm/entrypoints/speech_to_text/base/serving.py（模块 服务层；类别 source；类型 dependency-wiring；符号 shutdown, _decode_and_chunk_speech）：ASR 服务入口，实现预处理 offload 的核心逻辑：创建线程池、用 make_async_with_semaphore 包装同步方法、新增 shutdown 清理。
- vllm/utils/async_utils.py（模块 异步工具；类别 source；类型 core-logic；符号 make_async_with_semaphore, _async_wrapper）：新增 make_async_with_semaphore 工具函数，提供带信号量的异步包装，是线程池并发控制的关键基础设施。
- vllm/entrypoints/serve/utils/server_utils.py（模块 服务工具；类别 source；类型 core-logic）：在 FastAPI 生命周期退出时调用 ASR 服务的 shutdown，确保线程池被正确清理。
- vllm/envs.py（模块 环境配置；类别 source；类型 core-logic）：定义 VLLM_MAX_AUDIO_PREPROCESS_WORKERS 环境变量，控制 ASR 预处理线程数，默认值经实验选定为 2。

关键符号：make_async_with_semaphore, shutdown, _decode_and_chunk_speech

关键源码片段

vllm/entrypoints/speech_to_text/base/serving.py

ASR 服务入口，实现预处理 offload 的核心逻辑：创建线程池、用 make_async_with_semaphore 包装同步方法、新增 shutdown 清理。

```
# vllm/entrypoints/speech_to_text/base/serving.py ( 关键片段 )

class OpenAISpeechToText(OpenAIServing):
    def __init__(self, ...):
        # ... 其他初始化 ...
        # 设置预处理资源：独立线程池，避免阻塞主事件循环
        num_workers = envs.VLLM_MAX_AUDIO_PREPROCESS_WORKERS
        self._preprocess_executor = ThreadPoolExecutor(
            max_workers=num_workers,
            thread_name_prefix="stt-preprocess",
        )
        # 使用信号量包装器以限制并发任务数，便于请求取消
        self._decode_and_chunk_speech_async = make_async_with_semaphore(
            self._decode_and_chunk_speech,
            executor=self._preprocess_executor,
        )

    def shutdown(self) -> None:
        # 关闭线程池，不等待进行中的任务（调用者应已取消所有待处理请求）
        self._preprocess_executor.shutdown(wait=False)

    def _decode_and_chunk_speech(
        self,
```

```
        audio_data: bytes,
    ) -> tuple[list[np.ndarray], float]:
        # 解码音频字节流。容器格式（MP4, M4A, WebM）自动回退到 ffmpeg。
        # 注意：此处重采样到模型采样率以提高效率，这是分块的前提。
        try:
            with io.BytesIO(audio_data) as buf:
                y, sr = load_audio(
                    buf,
                    sr=self.asr_config.sample_rate,
                    max_duration_s=self.max_audio_decode_duration_s,
                )
        except Exception as exc:
            raise ValueError("Invalid or unsupported audio file.") from exc

        duration = get_audio_duration(y=y, sr=sr)
        # 判断是否需要分块：配置允许且音频长度超过阈值
        do_split = (
            self.asr_config.allow_audio_chunking
            and self.asr_config.max_audio_clip_s is not None
            and duration > self.asr_config.max_audio_clip_s
        )
        if not do_split:
            chunks = [y]
        else:
            # 使用 RMS 能量分割音频
            chunks = split_audio(
                audio_data=y,
                sample_rate=int(sr),
                max_clip_duration_s=self.asr_config.max_audio_clip_s,
                # ... 其他参数（如最小能量窗口大小）...
            )
        return chunks, duration
```

vllm/utils/async_utils.py

新增 make_async_with_semaphore 工具函数，提供带信号量的异步包装，是线程池并发控制的关键基础设施。

```
# vllm/utils/async_utils.py ( 新增函数 )

def make_async_with_semaphore(
    func: Callable[P, T],
    executor: ThreadPoolExecutor,
) -> Callable[P, Awaitable[T]]:
    """
    将阻塞函数包装为异步函数，在 executor 线程中运行。
    使用信号量（semaphore）限制并发数，使得取消尚未开始的任务成为可能。
    """
    # 信号量容量等于线程池最大工作数，防止过多任务排队
    semaphore = asyncio.Semaphore(executor._max_workers)
```

```
    async def _async_wrapper(*args: P.args, **kwargs: P.kwargs) -> T:
        loop = asyncio.get_event_loop()
        p_func = partial(func, *args, **kwargs)
        # 获取信号量后再提交到线程池，若信号量不可用则任务等待，便于外层取消
        async with semaphore:
            return await loop.run_in_executor(executor, p_func)

    return _async_wrapper
```

评论区精华

> NickLucche: "I think this approach can work, although it would be nice to align methodology with how the rest of the items are preprocessed with Renderer @DarkLight1337" ekagra-ranjan: "max thread = 2 was found optimal ... 2 threads RTFx: 1890, Benchmark duration 78.45. 1 thread: 1335.05, 4 thread: 1739.89, 8 thread: 1272.91, 16 thread: 1292.97" DarkLight1337: "We can simplify this via make_async" ekagra-ranjan: "I have now added a new util function called make_async_with_semaphore() because we also need the semaphore to bound tasks for easier cancellation." NickLucche: "I think we should just init this once clearly, these getters don't add much. Let's just safely assume these are initialized after boot up rather than at runtime" ekagra-ranjan: 同意简化，去掉了防御性 getter 和 Optional 类型。

- 独立线程池 vs 复用 Renderer 池 (performance): 保持独立 STT 预处理线程池，不复用 Renderer 池。
- 线程池大小选择 (performance): 默认值设为 max(1, min(cpu_count, 2))，用户可通过环境变量覆盖。
- 简化预处理资源初始化 (design): 去掉了 _get_stt_preprocess_max_workers 函数和 Optional 类型，直接在 __init__中使用环境变量值并创建 executor。
- 使用 semaphore 包装的异步函数 (design): 新增 make_async_with_semaphore 作为通用工具，并在 ASR 服务中使用。
- 移除预处理线程数启动日志 (style): 日志被移除。
- 避免 shutdown 时的 getattr (design): 在 lifespan 中直接罗列 'openai_serving_transcription' 和 'openai_serving_translation' 并调用 shutdown。

风险与影响

- 风险:
  1. 短音频性能回退：对于短音频（>80% 样本 <10s），RTFx 有约 1.5% 的下降，但用户可通过设置 VLLM_MAX_AUDIO_PREPROCESS_WORKERS=1 规避。
  2. 线程安全：_decode_and_chunk_speech 需保持线程安全，当前它只使用本地变量和 self.asr_config 等只读配置，风险较低，但任何未来修改需注意线程安全性。
  3. 默认线程数可能不通用：在 1xH100 上测试最优为 2，但其他硬件（如 CPU 核数少或 I/O 密集场景）可能需要调整。默认值 min(cpu_count, 2) 提供了合理上限。

4. 线程池泄漏：已通过 shutdown 在服务端和生命周期中处理，不会泄漏。 - 影响：用户影响：长音频转录吞吐大幅提升（2.5x），health 端点不再高并发超时，自动扩缩容更稳定。短音频用户可通过环境变量找回原有性能。系统影响：增加最多 2 个后台线程，CPU 负载轻微上升。团队影响：引入新环境变量 VLLM_MAX_AUDIO_PREPROCESS_WORKERS，需在部署文档中说明。代码结构清晰，易于维护。

- 风险标记：短音频性能回退 1.5%，默认线程数 2 可能需调优，多线程安全需持续关注

关联脉络

- PR #44587 [ASR] Add Long Audio benchmark and correctness test: 该 PR 引入了长音频基准测试和正确性测试，本 PR 依赖其测试来门控改动后的代码路径，且 benchmark 数据基于该数据集。