

PR #54746 完整报告

vllm-project/vllm

[Frontend] Share max_num_queued_reqs across API server processes

合并时间: 2026-09-15 22:50

原文链接: <http://prhub.com.cn/vllm-project/vllm/pull/54746>

执行摘要

- 一句话: 多 API server 进程共享全局准入计数
- 推荐动作: 值得精读, 尤其是负责前端多进程、过载保护与背压的工程师。建议重点关注三个设计决策: 一是用 cache line 对齐槽位 + 单写多读实现无锁跨进程计数器 (以及 64 字节常量为什么必须显式写出); 二是把“硬上限”主动降级为“近似全局上限”并用 docstring 固化契约, 这是本次 review 最有价值的结论, 也是引入任何跨进程准入机制时都要面对的取舍; 三是把计数刷新挂在 OutputProcessor 的 add/abort/finish 三个生命周期出口上, 理解这一点才能判断计数什么时候可能失真。落地前建议补两件事: 在 SchedulerConfig/CLI 帮助文本中同步“近似”措辞, 并考虑进程异常退出时的槽位回收或周期性重算。

功能与动机

PR body 的测试计划直接给出了动机: 以 `--api-server-count 4 --max_num_queued_reqs 10` 启动后, 用 hey 打 80 并发, 变更前 `vllm:num_requests_running` 稳定在 40, 说明多进程前端下这个“排队上限”被放大了 4 倍, 过载保护形同虚设; 变更后该指标收敛到 10。即: 准入控制是进程内状态, 与多 API server 前端的架构前提冲突, 需要一个跨进程可见的计数通道。PR 未关联任何 Issue, 动机完全来自这一组 before/after 指标。

实现拆解

1. 新增共享计数模块 (vllm/v1/engine/admission_control.py, +52): 定义 AdmissionCounterBuffer Protocol (只约束 `__getitem__` / `__setitem__`, 便于未来换成真正的共享内存实现) 与 SharedAdmissionStats。核心是 `CACHE_LINE_BYTES = 64`、`COUNTER_BYTES = 8`、`COUNTERS_PER_SLOT = 8` 三个常量: 每个进程独占一个 64 字节槽位, `__init__` 用 `client_addresses.pop("mp_admission_counters")` 取出共享数组并记录 `_slot_offset = client_index * COUNTERS_PER_SLOT`; `set_num_requests()` 只写自己槽位的第 0 个计数器, `get_num_requests()` 遍历所有进程槽位求和。这样读写都不需要跨进程锁, 也避免不同进程计数器落在同一条 cache line 上互相失效; `num_counters(client_count)` 供父进程计算数组长度。
2. 父进程分配并下发共享数组 (vllm/v1/utils.py 的 `APIServerProcessManager.__init__`, +11): 仅当 `num_servers > 1` 且 `getattr(args, "max_num_queued_reqs", None)` 非空时, 才用 `spawn_context.RawArray("q", SharedAdmissionStats.num_counters(num_servers))` 分配数组, 并写入每个子进程的 `client_config["mp_admission_counters"]`。使用函数

内延迟导入规避模块级循环依赖，并用 `getattr` 兼容测试里的 `SimpleNamespace`。单进程或未开启该选项的部署不会产生额外内存和分支。

3. 前端读路径接线 (`vllm/v1/engine/async_llm.py`, +19/-5) : `AsyncLLM.__init__` 在构造 `EngineCoreClient` 之前判断 `client_addresses is not None and "mp_admission_counters" in client_addresses`, 满足时创建 `self.admission_stats` 并作为 `admission_stats=` 注入 `OutputProcessor`; `check_admission()` 的 `max_num_queued_reqs` 分支由 `get_num_unfinished_requests()` 改为优先读 `self.admission_stats.get_num_requests()`, 为 `None` 时回退本地计数, `raise QueueOverflowError()` 的逻辑与日志字段随之改名。同一方法内把文档字符串从 "hard cap" 改为 "approximate global cap", 把“读快照 + 判定 + 写入”非原子这一事实固化为公开契约。注意 `SharedAdmissionStats.__init__` 的 `pop` 会从同一个 `dict` 中摘掉 `key`, 而该 `dict` 随后仍要传给 `EngineCoreClient.make_async_mp_client()`, 因此构造顺序敏感。
4. 引擎写路径挂点 (`vllm/v1/engine/output_processor.py`, +13/-1) : 新增 `admission_stats` 构造参数 (用 `TYPE_CHECKING` 导入 `SharedAdmissionStats` 避免运行期循环依赖) 与私有方法 `_update_admission_stats()`, 后者在 `admission_stats is not None` 时把 `get_num_unfinished_requests()` (即 `len(request_states)`) 写入本进程槽位。调用点有三处: `add_request()` 末尾 (新请求登记之后)、`abort_requests()` 返回前 (覆盖用户取消与级联取消子请求)、`_finish_request()` 末尾 (覆盖正常完成与流式输入结束的分支)。也就是说计数刷新绑定在请求生命的三个出口上, 而不是周期性扫描。
5. 测试配套 (`tests/v1/engine/test_admission_control.py` +39、`tests/entrypoints/launchers/api_server/test_api_server_process_manager.py` +39) : 前者新增 `_make_shared_stats()` (用 `multiprocessing.RawArray("q", SharedAdmissionStats.num_counters(2))` 造两个进程视图) 与两个 `@pytest.mark.skip_global_cleanup` 用例, 分别验证聚合可见性 (`2+3=5`、回写 1 后为 4) 和 `AsyncLLM.check_admission()` 会因共享快照触发 `QueueOverflowError` (同时断言对方槽位未被写脏); 后者新增模块级 `update_admission_stats_worker` (`spawn` 子进程写 `client_index + 1`, 用 `Barrier` 同步后回读) 与 `test_api_server_processes_share_admission_stats`, 断言 3 个进程都读到聚合值 6, 证明 `RawArray` 在 `spawn` 语义下真的跨进程可见。两处都用 `args=SimpleNamespace(max_num_queued_reqs=10)` 走 `APIServerProcessManager` 的真实分配分支。测试未包含并发竞态回归与真实 HTTP 压测。

关键文件:

- `vllm/v1/engine/admission_control.py` (模块 准入控制; 类别 `source`; 类型 `core-logic`; 符号 `AdmissionCounterBuffer`, `getitem`, `setitem`, `SharedAdmissionStats`) : 本 PR 的核心新增模块: 用 `multiprocessing.RawArray` + `cache line` 对齐槽位实现无锁跨进程准入计数器, 是后续所有跨进程准入 / 统计共享的基础抽象。
- `vllm/v1/engine/async_llm.py` (模块 引擎客户端; 类别 `source`; 类型 `core-logic`; 符号 `AsyncLLM.init`, `check_admission`, `admission_stats`) : 前端读路径: 初始化共享统计并注入 `OutputProcessor`, `check_admission()` 从本地计数改为读全局快照, 同时把语义文档从 `hard cap` 改为 `approximate global cap`。
- `vllm/v1/engine/output_processor.py` (模块 输出处理; 类别 `source`; 类型 `core-logic`; 符号 `OutputProcessor.init`, `add_request`, `abort_requests`, `_finish_request`) : 引擎写路径

: 新增 admission_stats 参数与 _update_admission_stats(), 把本进程未完成请求数在请求生命周期的三个出口同步到共享槽位。

- vllm/v1/Utils.py (模块 进程管理; 类别 source; 类型 dependency-wiring; 符号 APIServerProcessManager.init) : 父进程侧: APIServerProcessManager 在满足条件时分配 RawArray 并通过 client_config 下发给每个 API server 子进程, 是共享内存的唯一创建点。
- tests/v1/engine/test_admission_control.py (模块 准入控制; 类别 test; 类型 test-coverage; 符号 _make_shared_stats, test_shared_admission_stats_aggregate_api_servers, test_shared_request_snapshot_enforces_global_limit) : 新增共享计数聚合与全局限制的单进程单元测试, 覆盖 SharedAdmissionStats 的读写语义和 check_admission 走共享快照的判定分支。
- tests/entrypoints/launchers/api_server/test_api_server_process_manager.py (模块 进程管理; 类别 test; 类型 test-coverage; 符号 update_admission_stats_worker, test_api_server_processes_share_admission_stats) : 真实 spawn 多进程集成测试, 证明 APIServerProcessManager 下发的 RawArray 能被各子进程共享并聚合出全局计数。

关键符号: SharedAdmissionStats.init, SharedAdmissionStats.num_counters, SharedAdmissionStats.set_num_requests, SharedAdmissionStats.get_num_requests, OutputProcessor._update_admission_stats, OutputProcessor.add_request, OutputProcessor.abort_requests, OutputProcessor._finish_request, AsyncLLM.check_admission, APIServerProcessManager.init, update_admission_stats_worker

关键源码片段

vllm/v1/engine/admission_control.py

本 PR 的核心新增模块: 用 multiprocessing.RawArray + cache line 对齐槽位实现无锁跨进程准入计数器, 是后续所有跨进程准入 / 统计共享的基础抽象。

```
# SPDX-License-Identifier: Apache-2.0
# SPDX-FileCopyrightText: Copyright contributors to the vLLM project
from typing import Any, Protocol, cast
```

```
class AdmissionCounterBuffer(Protocol):
```

```
    """共享计数缓冲区的最小协议: 只需要能按下标读写。
```

```
    抽成 Protocol 是为了让上层不依赖 multiprocessing.RawArray 的具体类型,
    未来可替换为其它共享内存实现。
```

```
    """
```

```
    def __getitem__(self, index: int) -> int: ...
```

```
    def __setitem__(self, index: int, value: int) -> None: ...
```

```
class SharedAdmissionStats:
```

```
    """Lock-free admission stats shared by API server processes.
```

每个 API server 进程只写自己独占的槽位，读取方聚合所有槽位，因此既不需要跨进程锁，也不会在请求热路径上产生 cache line 争用。

"""

```
# 每个进程的计数器相隔 64 字节，避免 64 字节 cache line 上的 false sharing:
# 否则一个进程的写入会反复让另一个进程计数器所在的 cache line 失效。
CACHE_LINE_BYTES = 64
COUNTER_BYTES = 8
COUNTERS_PER_SLOT = CACHE_LINE_BYTES // COUNTER_BYTES

def __init__(
    self,
    client_addresses: dict[str, Any],
    client_count: int = 1,
    client_index: int = 0,
):
    # 取出父进程通过 client_config 下发的共享数组 (multiprocessing.RawArray("q", ...)) ;
    # pop 同时把该 key 从 client_addresses 摘掉，避免它继续传给 EngineCoreClient。
    self._counters = cast(
        AdmissionCounterBuffer,
        client_addresses.pop("mp_admission_counters"),
    )
    self._client_count = client_count
    # 本进程在共享数组中的起始下标：每个进程独占一个 cache-line 大小的槽位
    self._slot_offset = client_index * self.COUNTERS_PER_SLOT

    @classmethod
    def num_counters(cls, client_count: int) -> int:
        # 供父进程分配共享数组时使用：进程数 × 每槽计数器个数
        return client_count * cls.COUNTERS_PER_SLOT

    def set_num_requests(self, count: int) -> None:
        # 只写自己槽位的第 0 个计数器，天然无锁
        self._counters[self._slot_offset] = count

    def get_num_requests(self) -> int:
        # 聚合所有进程的槽位，得到全局未完成请求数的快照（非原子）
        return sum(
            self._counters[client_index * self.COUNTERS_PER_SLOT]
            for client_index in range(self._client_count)
        )
```

vllm/v1/engine/output_processor.py

引擎写路径：新增 admission_stats 参数与 _update_admission_stats()，把本进程未完成请求数在请求生命周期的三个出口同步到共享槽位。

```
if TYPE_CHECKING:
    # 仅类型检查期导入，避免运行期循环依赖
    from vllm.v1.engine.admission_control import SharedAdmissionStats
```

```

class OutputProcessor:
    """Process EngineCoreOutputs into RequestOutputs."""

    def __init__(
        self,
        tokenizer: TokenizerLike | None,
        *,
        log_stats: bool,
        stream_interval: int = 1,
        tracing_enabled: bool = False,
        # 单进程部署时为 None, 所有刷新逻辑自动退化为 no-op
        admission_stats: "SharedAdmissionStats | None" = None,
    ):
        ...
        self.admission_stats = admission_stats

    def abort_requests(self, request_ids: Iterable[str], internal: bool) -> list[str]:
        ...
        for request_id in internal_req_ids:
            req_state = self.request_states.pop(request_id, None)
            ...
        # 中止 (含级联取消子请求) 会减少本进程未完成请求, 需要立刻同步给其它 API server
        self._update_admission_stats()
        return request_ids_to_abort

    def add_request(
        self,
        request: EngineCoreRequest,
        prompt: str | None,
        parent_req: ParentRequest | None = None,
        request_index: int = 0,
        queue: RequestOutputCollector | None = None,
    ) -> None:
        ...
        # Track the external_req_id -> [internal_req_id, ...] mapping
        self.external_req_ids[req_state.external_req_id].append(request_id)
        # 请求已登记进 request_states, 刷新共享计数以便其它进程立刻看到
        self._update_admission_stats()

    def _finish_request(self, req_state: RequestState) -> None:
        ...
        parent_req = req_state.parent_req
        if parent_req and not parent_req.child_requests:
            self.parent_requests.pop(parent_req.request_id, None)
        # 正常完成与流式输入结束都会走到这里, 是计数回收的最后一个出口
        self._update_admission_stats()

    def _update_admission_stats(self) -> None:
        """把本进程当前未完成请求数写入自己的共享槽位."""

```

```
if self.admission_stats is not None:
    self.admission_stats.set_num_requests(self.get_num_unfinished_requests())
```

评论区精华

最核心的交锋是 sylvesterkaczmarek 提出的竞态: `check_admission()` 读取聚合计数, 但各进程只在 `OutputProcessor.add_request()` 之后才更新自己的槽位, 两个 server 可以同时观察到 `current < max` 并同时放行, 突发流量会超出 `max_num_queued_reqs`; 他追问“如果要做硬全局上限就该原子 reserve/release 并配同时准入的回归测试, 如果是有意的近似就必须写进用户可见文档”。NickLucche 的回应是取舍结论: “the concern is introducing overhead on admission, so I think it's ok to trade atomicity here (with clear docs). I think we can bound overshooting to num-api-servers though..?”, 即热路径上不引入跨进程原子预留, 用文档承认近似并接受最多约 num-servers 级别的超额。sylvesterkaczmarek 复看后确认 “docstring now explicitly calls max_num_queued_reqs an approximate global cap, which resolves the contract ambiguity I raised”。另一条设计讨论来自 NickLucche: 建议把 `client_addresses.pop("mp_admission_counters", None)` 的判断放到调用方, “when mp_admission_counters is not present, dont create the class and the rest is a noop”, 让类型保持单一; 作者曾回复 “I feel the original approach might actually be cleaner”, 但最终合入版本仍采用调用方判断加类内必填的形式。此外 NickLucche 称赞按 cache line 间隔计数器 “pretty cool”, 要求补注释以便外部实现者理解, 并给了 `_get_global` 用切片聚合的 nit, 前者已落到最终代码注释里, 后者未采纳。

- 跨进程准入的非原子性: 硬上限还是近似上限 (design): 保留无锁近似实现; AsyncLLM.check_admission 的 docstring 由 "hard cap" 改为 "approximate global cap", reviewer 复看后确认契约歧义已解决并批准。原子预留与并发准入回归测试未实现。
- SharedAdmissionStats 的 None 语义与构造位置 (design): 最终合入版本采纳了 reviewer 方案: SharedAdmissionStats.__init__ 的 client_addresses 变为必填并直接 pop, AsyncLLM 用 client_addresses is not None and "mp_admission_counters" in client_addresses 决定是否构造, OutputProcessor 保持 SharedAdmissionStats | None。
- cache line 间隔布局的注释与聚合写法 (style): 注释建议已落地: 最终代码保留 CACHE_LINE_BYTES / COUNTER_BYTES / COUNTERS_PER_SLOT 常量, 并在类 docstring 与常量处写明避免 cache line false sharing; 切片式聚合未采纳, get_num_requests 仍用显式步长索引求和。
- n = 1 时 noop 分支的整理 (question): 作为非阻塞建议被搁置: 作者未给出对应改动说明, 最终版本的准入判定逻辑与刷新钩子看不出针对 n = 1 的独立分支, 评审者随后直接 APPROVED。

风险与影响

- 风险: 1) 准入上限变为近似值 (已确认的设计取舍): check_admission() 的读 - 判 - 写跨进程不原子, N 个 API server 在同一个瞬间都可能通过判定, 最坏超额约为并发准入的进程数; --max_num_queued_reqs 从“硬上限”降级为“近似全局上限”, 与旧的每进程语义相比行为变化明显 (同一负载下 running 由 40 变 10), 依赖旧隐含 4 倍配额的容量规划会在升级后看到更多 503。变更只改了 AsyncLLM 的 docstring,

SchedulerConfig.max_num_queued_reqs 这个用户可见字段的说明与 CLI 帮助文本不在本次文件列表中，存在文档不同步的风险。 2) 计数漂移不可自愈：槽位只在 add_request / abort_requests / _finish_request 三处刷新，任何绕过 OutputProcessor 的请求状态变更路径、或 API server 进程在持有请求时崩溃退出，都会让该槽位长期停留在旧值，且没有周期性重算或进程退出清理机制，结果是全局计数虚高并持续返回 503。 3) 构造顺序与共享 dict 副作用：SharedAdmissionStats.__init__ 用 pop 从 client_addresses 中摘除 mp_admission_counters，而该 dict 之后还要传给 EngineCoreClient.make_async_mp_client(); 若未来重排 AsyncLLM.__init__ 的步骤或复用该 dict，会出现 key 缺失 (pop 无默认值会 KeyError) 或提前清理的隐性耦合。 4) 不对称覆盖：max_num_queued_tokens 分支仍使用本进程的 get_num_queued_tokens()，多进程下该 TTFT QoS 上限依然是按进程生效，本 PR 只解决了一半准入语义。 5) 测试盲区：新增用例验证了聚合与单点判定，但没有并发同时准入的回归用例 (reviewer 建议未落实)，也没有进程崩溃后计数回收的用例。 6) 性能面基本无回退：读路径是 $O(\text{client_count})$ 的 Python 求和，写路径 $O(1)$ ，只在 num_servers > 1 且设置了该选项时启用，代价可忽略。

- 影响：用户侧：使用 --api-server-count > 1 与 --max_num_queued_reqs 的部署会立刻获得真正的过载保护，代价是同等负载下被拒 (HTTP 503) 的比例上升，压测 / 容量规划需要按“全局上限”重新评估；单进程部署与未设置该选项的部署行为、内存、分支完全不变，向后兼容。系统侧：前端多进程 (spawn) 新增一条通过 client_config 传递的共享内存通道 (multiprocessing.RawArray)，与既有的 ZMQ 地址、tensor_queue、stats_update_address 属于同一套进程间配置注入模式，不引入新锁与新线程；由于粒度是 cache line 对齐，也避免了多进程写同一数组时的 false sharing。团队侧：AdmissionCounterBuffer Protocol 与 SharedAdmissionStats 为后续把 max_num_queued_tokens、跨进程调度 / 指标状态做成共享计数器提供了可复用模板，“近似而非精确”的契约写法也为后续放宽语义提供了先例。
- 风险标记：准入语义由硬上限改为近似上限，读 - 判 - 写非原子，可超额准入，缺少并发准入回归测试，进程崩溃后共享槽位无回收，max_num_queued_tokens 未同步共享

关联脉络

- PR #56249 [Bugfix][Frontend] Handle aborted requests in beam search: 同样落在前端请求生命周期与中止路径：本 PR 的共享计数正是挂在 OutputProcessor.abort_requests 与 _finish_request 上，两者都关心请求被中止后计数与资源回收是否一致。
- PR #53124 [Bugfix] Load reasoning parser plugins before headless engine config: 同属 API server/headless 前端启动期的装配顺序问题：该 PR 修的是插件导入必须早于引擎配置，本 PR 则是 SharedAdmissionStats 必须在 EngineCoreClient 之前从 client_addresses 取出 (pop 有副作用)。
- PR #56990 [Rust Frontend] Add iteration token histogram: 同一脉络：前端 / 引擎侧统计在多进程、多引擎场景下的聚合与上报，本 PR 把“每进程独立统计”从指标层面延伸到准入控制层面，都涉及跨进程可见性的设计取舍。