

PR #55885 完整报告

vllm-project/vllm

[KV Offload] Add per-request `max\_load\_tokens` control`

合并时间: 2026-09-16 11:57

原文链接: <http://prhub.com.cn/vllm-project/vllm/pull/55885>

执行摘要

- 一句话: KV Offload 新增按请求的 `max_load_tokens` 加载上限
- 推荐动作: 值得精读, 尤其面向 KV offload / 调度方向的同学。建议关注三个设计决策: 一是 `__post_init__` 的解析约定 (`type(raw) is int` 排除 `bool`、非法值 `warning` + 忽略而非报错); 二是 `cap` 在 `_lookup_complete_chunks` 中的两个插入点 (先按 GPU 前缀偏移取 `min`, 再按 `group chunk` 粒度下取整), 理解为什么 0 会自然退化为“直接返回 0”; 三是测试设计——在 `hybrid KV cache helper` 上同时覆盖 `sync/async scheduling`, 并用 `expected_stored` 断言 `store` 路径不受影响。若要投产, 建议先补齐多级部署验证, 并考虑在服务入口对字符串数字等易错输入给出更显式的告警或校验。

功能与动机

卸载连接器此前只有 `max_offload_tokens` 控制“写出去多少”, 缺少对称的“读回来多少”开关。最初的实现尝试用“空的 `kv_load_tiers` 列表”表达禁用外部加载, `review` 中维护者 `orozer` 指出该思路与既有语义冲突: “`kv_load_tiers` assumes CPU is always included since all other tiers promote through it, so it's meaningless to have a non-empty list without CPU included.”, 并建议新增与 `max_offload_tokens` 对称的 `max_load_tokens`。PR body 进一步明确动机: 既能让单请求彻底禁用外部加载 (缺失 token 重算), 也能按请求限制回载量以控制带宽 / 成本, 同时不影响本地 GPU 前缀缓存复用与 KV store。

实现拆解

1. 参数解析与状态字段 (`vllm/distributed/kv_transfer/kv_connector/v1/offloading/scheduler.py`): `RequestOffloadState` 新增 `max_load_tokens: int | None = None`, `__post_init__` 从 `req.kv_transfer_params` 读取, 判定条件为 `type(raw) is int and raw >= 0` (刻意排除 `bool`, `True` 视为非法), 非法值只记 `logger.warning` 并保持 `None`。该解析与紧随其后的 `max_offload_tokens` 完全对称且互不影响, 保证既有请求零回归。
2. 完整 chunk 命中上限 (`OffloadingConnectorScheduler._lookup_complete_chunks`): 在 `max_num_new_tokens` 收紧之后, 用 `num_computed_tokens + max_load_tokens` 再取 `min`, 明确 `cap` 以“GPU 常驻前缀之外的 token 数”计量; 随后在按 `group chunk` 粒度约束处, 仅当 `cap` 生效时才追加 `round_down(max_hit_size_tokens, tokens_per_chunk)`。若对齐后 `max_hit_size_tokens - num_computed_tokens < tokens_per_chunk`, 函数返回 0——这正是 `max_load_tokens = 0` 的实现落点, 因此 0 值连 `lookup` 都不会发起。

- 3. partial tail 细粒度上限 (OffloadingConnectorScheduler._lookup) : max_boundary 同样被 local_tokens + max_load_tokens 收紧，之后才 round_down(..., tokens_per_hash) ; 若 cap 把边界压到 complete_boundary 之下则直接返回 complete_hit，不会为超出 cap 的尾部 token 发起 boundary key 查询。
- 4. 测试配套 (tests/v1/kv_connector/unit/offloading_connector/test_scheduler.py) : helper _make_partial_tail_request 新增 kv_transfer_params 形参并补上 request.skip_reading_prefix_cache = False; 新增 5 个用例覆盖 0 值跳过 partial tail 查询、正数封顶、无 partial tail 时的 chunk 下取整、非法值忽略，以及 parametrize async_scheduling 的端到端用例，用 expected_stored=(0, 1, 2) 断言 store 路径不受 cap 影响。
- 5. 文档 (docs/features/kv_offloading_usage.md) : 新增 “Per-request load control” 小节，给出JSON示例、声明字段为实验性，并补充kv_load_tiers的语义说明（CPU始终包含、空列表只表示不查询次级 tier），把 review 中的解释沉淀进文档。

行为	配置	效果
禁用外部加载	max_load_tokens: 0	不发起 lookup，缺失 token 重算；GPU 前缀命中与 store 不受影响
封顶加载	max_load_tokens: N > 0	仅加载 GPU 前缀之外 N 个 token，并按 chunk / tokens_per_hash 下取整
不限量	省略字段	与改动前一致
非法值	"8"、8.5、-1、True	warning 后按“不限量”处理

关键文件：

- vllm/distributed/kv_transfer/kv_connector/v1/offloading/scheduler.py（模块 卸载调度；类别 source；类型 core-logic；符号 RequestOffloadState, RequestOffloadState.post_init, OffloadingConnectorScheduler._lookup_complete_chunks, OffloadingConnectorScheduler._lookup）：核心实现文件：新增 max_load_tokens 状态字段与解析逻辑，并在完整 chunk 查询与 partial tail 查询两处收敛该 cap，是本次行为的唯一决策点。
- tests/v1/kv_connector/unit/offloading_connector/test_scheduler.py（模块 卸载测试；类别 test；类型 test-coverage；符号 _make_partial_tail_request, test_max_load_tokens_zero_skips_partial_tail_lookup, test_max_load_tokens_caps_tokens_beyond_gpu_prefix, test_max_load_tokens_rounds_down_without_partial_tail）：测试配套文件：5 个新用例把 cap 的语义（0 值、封顶、下取整、非法值、store 不受影响）固化成可执行规范，并改造 partial tail helper 支持 kv_transfer_params。
- docs/features/kv_offloading_usage.md（模块 使用文档；类别 docs；类型 documentation）：文档更新文件：新增 “Per-request load control” 小节，固化 max_load_tokens 的取值语义与实验性声明，并澄清 kv_load_tiers 中 CPU 常驻的含义。

关键符号: RequestOffloadState.post_init, OffloadingConnectorScheduler._lookup_complete_chunks, OffloadingConnectorScheduler._lookup, _make_partial_tail_request, test_max_load_tokens_limits_external_load

关键源码片段

[vllm/distributed/kv_transfer/kv_connector/v1/offloading/scheduler.py](#)

核心实现文件: 新增 max_load_tokens 状态字段与解析逻辑, 并在完整 chunk 查询与 partial tail 查询两处收敛该 cap, 是本次行为的唯一决策点。

```
@dataclass(slots=True)
class RequestOffloadState:
    config: SchedulerOffloadConfig
    req: Request
    req_context: ReqContext
    offloading_context: RequestOffloadingContext
    group_states: tuple[RequestGroupState, ...] = field(init=False)
    # 本请求允许从外部加载的 token 上限; None 表示不设上限
    max_load_tokens: int | None = None
    # 本请求允许卸载的 token 上限; None 表示不设上限 (与 max_load_tokens 对称且互不影响)
    max_offload_tokens: int | None = None
    # GPU 前缀缓存命中数: cap 只作用于这部分之后的 token
    num_locally_computed_tokens: int = 0

    def __post_init__(self) -> None:
        self.group_states = tuple(
            RequestGroupState() for _ in self.config.kv_group_configs
        )
        params = self.req.kv_transfer_params

        # NOTE: 实验性字段, 后续可能调整; 0 表示禁止任何外部 KV 加载
        raw = params.get("max_load_tokens") if params else None
        # 刻意用 type(raw) is int 而非 isinstance: bool 是 int 的子类,
        # 若用 isinstance 则 True 会被当成 1 而不是非法值
        if type(raw) is int and raw >= 0:
            self.max_load_tokens = raw
            logger.debug(
                "Request %s: max_load_tokens set to %d", self.req.request_id, raw
            )
        elif raw is not None:
            # 非法值 (字符串、浮点、负数、bool) 只告警并忽略, 退化为“不限量”,
            # 这样既保持向后兼容, 也不会因为一个坏参数打断请求
            logger.warning(
                "max_load_tokens must be a non-negative int, got %r; ignoring", raw
            )

    def _lookup_complete_chunks(
        self,
        req_status: RequestOffloadState,
```

```

    max_num_new_tokens: int | None = None,
) -> int | None:
    """统计 num_locally_computed_tokens 之后还有多少 token 可以从外部加载。"""
    num_computed_tokens = req_status.num_locally_computed_tokens
    max_hit_size_tokens: int = req_status.req.num_tokens
    if max_num_new_tokens is not None:
        max_hit_size_tokens = min(
            max_hit_size_tokens, num_computed_tokens + max_num_new_tokens
        )
    if req_status.max_load_tokens is not None:
        # cap 以“GPU 常驻前缀之后的 token 数”计量,
        # 因此是 num_computed_tokens + max_load_tokens, 而非绝对 token 位置
        max_hit_size_tokens = min(
            max_hit_size_tokens, num_computed_tokens + req_status.max_load_tokens
        )
    if self._sliding_window_groups:
        # sliding window 必须重算最后一个 prompt token 以获取 logprobs, 故再减 1
        max_hit_size_tokens -= 1
    if self._mamba_align_size is not None:
        max_hit_size_tokens = round_down(
            max_hit_size_tokens, self._mamba_align_size
        )

    num_hit_tokens: int = 0
    while lookup_groups:
        for group_idx in groups_iter:
            group_config: GroupOffloadConfig = self.config.kv_group_configs[group_idx]
            group_state: RequestGroupState = req_status.group_states[group_idx]
            tokens_per_chunk = group_config.tokens_per_chunk

            # 先按该 group 的 chunk 容量约束整体上限
            max_hit_size_tokens = min(
                max_hit_size_tokens, len(group_state.offload_keys) * tokens_per_chunk
            )
            if req_status.max_load_tokens is not None:
                # 仅当 cap 生效时才对 chunk 边界下取整:
                # 例如 cap = 20、chunk = 16 时实际只加载 16, 其余走重算
                max_hit_size_tokens = round_down(
                    max_hit_size_tokens, tokens_per_chunk
                )
            if max_hit_size_tokens - num_computed_tokens < tokens_per_chunk:
                # 对齐后不足一个 chunk 就彻底放弃加载;
                # max_load_tokens = 0 时正是从这里直接返回 0 (连 lookup 都不发起)
                return 0

    def _lookup(
        self,
        req_status: RequestOffloadState,
        max_num_new_tokens: int | None = None,

```

```

) -> int | None:
    complete_hit = self._lookup_complete_chunks(req_status, max_num_new_tokens)
    req_status.partial_tail_boundary = None
    if complete_hit is None or not self.config.supports_partial_tail:
        return complete_hit

    local_tokens = req_status.num_locally_computed_tokens
    complete_boundary = local_tokens + complete_hit
    tokens_per_hash = self.config.tokens_per_hash
    block_end = complete_boundary + self._partial_tail_block_size
    max_boundary = min(req_status.req.num_prompt_tokens - 1, block_end - 1)
    if max_num_new_tokens is not None:
        max_boundary = min(max_boundary, local_tokens + max_num_new_tokens)
    if req_status.max_load_tokens is not None:
        # partial tail 的细粒度 boundary 查询同样不能越过 cap,
        # 否则用户设的 0 值会因为尾部命中而失效
        max_boundary = min(max_boundary, local_tokens + req_status.max_load_tokens)
    max_boundary = round_down(max_boundary, tokens_per_hash)
    if max_boundary <= complete_boundary:
        return complete_hit # cap 把边界压回完整 chunk 边界, 尾部不再缩小
    # ... 从 max_boundary 向 complete_boundary 逐 tokens_per_hash 探测 boundary key

```

评论区精华

核心交锋是设计方向：orozery 否定了最初“用空 `kv_load_tiers` 禁用加载”的方案，理由是 tier 语义上必然包含 CPU（其他 tier 都要经 CPU 提升），因此建议引入与 `max_offload_tokens` 对称的新参数 `max_load_tokens`；作者据此把首个 commit [028fd2e](#)（Empty `kv_load_tiers` disable all external loads）重构为 commit [463fb87](#)（Use `max_load_tokens` for KV load control），并在文档里补上 tier 语义澄清。第二个线程是测试风格：orozery 对 [@pytest.mark.skip_global_cleanup](#) 追问“Why do we need this? (and in the other tests as well)”，作者承认它只是本地慢机器上的提速手段、并非正确性所需，属于遗留，随后以 commit [aab1ad6](#)（tests: Remove unnecessary fixture）移除。整个讨论最终以 orozery APPROVED 收敛，未留下未决问题；自动化 review 因来自 fork 被禁用（claude[bot] 仅提示“automated review is disabled”），因此讨论主要来自单一维护者，视角较集中。

- 用 `max_load_tokens` 取代“空 `kv_load_tiers`”的禁用方案 (design): 采纳 `max_load_tokens` 新参数方案；“加载量上限”与“tier 选择”被明确为两件正交的事，并在文档中补充说明。
- 测试中的 `@pytest.mark.skip_global_cleanup` 是否必要 (testing): 标记与相关 fixture 被移除（commit [aab1ad6](#) “tests: Remove unnecessary fixture”），测试保持与既有风格一致，不引入额外执行假设。
- 非法 `max_load_tokens` 的处理策略是告警忽略而非报错 (correctness): 与 `max_offload_tokens` 保持一致：参数非法不中断请求，仅日志告警，由调用方自行核对。
- 把 `kv_load_tiers` 的 CPU 常驻语义沉淀进文档 (documentation): 文档明确了 `max_load_tokens` 与 `kv_load_tiers`、`max_offload_tokens`、`skip_reading_prefix_cache` 各自的职责边界。

风险与影响

- 风险：1) 核心调度路径变更：_lookup_complete_chunks 与 _lookup 决定“命中多少、加载多少”，虽然所有新分支都被 if req_status.max_load_tokens is not None 包裹（未设置时与旧逻辑等价），但一旦启用，影响会传导到 partial_tail_boundary、load job 的目标 block 计算与后续重算量。2) 下取整放大重算：PR body 与文档都承认 cap 会“rounded down to a supported KV-cache boundary”，但用户易按 token 数精确预期。对 hybrid KV cache (full attention + Mamba/SWA) 与 Eagle/MTP 场景，最终粒度取决于最细的 tokens_per_chunk；SWA 路径还会先做 -1（最后 token 需重算）和 _mamba_align_size 对齐，实际加载量可能明显小于设定值（测试中 20 被对齐成 16）。3) 非法值静默忽略：type(raw) is int 会拒绝 JSON 中常见的字符串数字 "8"，仅记录 warning；线上大规模部署时用户可能误以为上限已生效。这是为与 max_offload_tokens 保持一致的有意选择，但排查成本需纳入考虑。4) 验证缺口：PR body 中“Run the focused tests on the final revision”与“Validate the final API in a live multi-tier deployment”两项未勾选，Test Result 为 Pending；测试未覆盖次级 tier (secondary/SN) 与 kv_load_tiers 组合的真实行为。5) 语义正交性易误解：cap 只限 load 不限 store，max_load_tokens: 0 的请求仍会把新算出的块写回 CPU，需要依赖文档澄清。
- 影响：用户侧：默认行为完全不变（省略字段即不限量），可作为准入 / 成本控制手段——例如高优请求允许回载、低优请求强制重算以避免占用卸载带宽；但需理解“按 chunk 下取整”与“只限 load”两点语义。系统侧：改动位于 v1 KV offload scheduler 的查询路径，采用 per-request 状态而非全局配置，不引入新依赖，kv_load_tiers、skip_reading_prefix_cache、store 路径语义均不变，回归面可控。团队侧：以 max_offload_tokens 为模板确立了“实验性 per-request 参数 → 解析失败仅告警忽略 → 在统一上限收敛点生效”的扩展模式，后续 per-request 卸载策略可直接复用该范式。
- 风险标记：核心调度路径变更，实验性 API 未冻结，缺少多级部署实测，上限下取整可能放大重算，非法值静默忽略

关联脉络

- PR #56709 [Bugfix][KV Offload] Restore MTP-retained sliding-window history: 修改同一组文件 (offloading/scheduler.py 与 offloading_connector/test_scheduler.py)，同样围绕 SWA/MTP 的加载边界计算；本 PR 的 cap 恰好插在 max_hit_size_tokens 收敛链路的相邻位置，两者叠加时需关注边界计算顺序。
- PR #54014 [Bugfix][KV Offload] Check cgroup memory before SHM allocation: 同属 KV Offload 子系统 (CPU/SHM 资源与卸载区域)，是该功能线近期密集补齐“资源与边界控制”的一环，与本 PR 的“按请求控制加载量”形成互补。
- PR #50494 [KVConnector][NIXL] Support attention-HMA layouts in pipeline-parallel push prefill: 体现 kv-connector 侧 tier/ 层路由的演进方向；本 PR 的文档更新正是在澄清 tier 选择与加载量上限是正交的两件事，为后续更细粒度的 per-request 卸载策略打基础。