

PR #53453 完整报告

vllm-project/vllm

[KVConnector][P2P] Configurable unbound-store timeout and one-RTT rejection of a late fetch

合并时间: 2026-09-15 10:38

原文链接: <http://prhub.com.cn/vllm-project/vllm/pull/53453>

执行摘要

- 一句话: P2P 卸载层超时可配, 晚到 fetch 一 RTT 内被拒
- 推荐动作: 值得精读, 尤其是负责 KV offload / tiering / PD 分离部署的工程师。三个值得关注的设计点: 一是用带剪枝的墓碑表 `_reaped_stores` 承接跨实例失败通知, 复用既有 `finish_request` $\rightarrow$ `_finalize_outbound` 终态路径而不改协议、不加 session API, 是把「超时」变成「可立即观测的失败」的干净做法; 二是配置入口「先 `float()` 强转、再以 NaN 兜底统一报错」的模式, 对任何从 `kv_connector_extra_config` JSON 原样 splat 进来的 tier 参数都适用, 可直接照搬到其它 tier; 三是 `_REAPED_ID_RETENTION_S` 与消费者 `_LOAD_TIMEOUT_S` 的隐式耦合是唯一需要持续盯住的设计点, 建议后续要么加断言 / 测试, 要么把二者关联起来。测试方面, `recording_drain` 断言「线上从未出现 `abort_fetch`」这种验证路径而非仅验证结果的手法, 也值得其它失败路径测试借鉴。

功能与动机

Issue #53128 报告: PD Multi Tier 的生产者从「第一个 stored chunk」起算 60 s 就丢弃 park 的 KV, 而消费者的 fetch 要等剩余 prefill、生产者 HTTP 返回、代理分发、解码准入之后才发出, 实测 98 K token 的 prefill 在空闲 H200 上就吃掉 6.7 s、轻载下 13.3 s, 长 prompt 真实并发下总和突破 60 s。后果是消费者打满 `_LOAD_TIMEOUT_S = 30.0` (实测 control 组 0.30 s / 580 MB, late 组 38.31 s / 0 字节) 再回退本地重算, 而重算本身又推高负载。PR body 明确指出两点: 「The 60 s ceiling was not configurable」以及「A fetch arriving after the reap was never rejected — this is the user-visible damage」。此外 PR 还指出回收路径里的 `self._failed_req_ids.add(kid)` 只在文档里声称能让晚到 fetch 短路, 实际 fetch 路径没有任何读取者, 且 `on_request_finished` 是唯一的 discard 点而它早已执行过, 于是该集合在进程生命周期内只增不减——本次一并修掉。

实现拆解

1. 引入 per-tier 配置键并做先行类型强转: `P2PSecundaryTierManager.__init__` 新增形参 `unbound_store_timeout_s: float = _UNBOUND_STORE_TIMEOUT_S`, 形状与既有 `host/port/backends/num_threads` 对齐 (由 `SecondaryTierFactory.create_secondary_tier` 以 `**config` 原样 splat 传入, 因此不能假设类型已是 float)。先 `float()` 再用 `if not timeout_s > 0` 判范围, `TypeError/ValueError` 统一转成 `float('nan')` 兜底, 保证 "180" 这类被模板渲染成字符串的 JSON 数值能通过、而不可转换的值会得到带配置键名的 `ValueError`。结果存入 `self._unbound_store_timeout_s`。

2. 用墓碑表替代有缺陷的失败集合：新增 `self._reaped_stores: dict[str, float]` (`kv_request_id` → 回收时刻)。`_reap_unbound_stores` 把原来的 `self._failed_req_ids.add(kid)` 换成 `self._reaped_stores[kid] = now`，并在函数开头按新增常量 `_REAPED_ID_RETENTION_S = 60.0` 剪枝过期墓碑——注释明确它必须大于消费者侧的 `session.client._LOAD_TIMEOUT_S`，否则消费者早已自行放弃、没有人能收到拒绝。剪枝放在 `if not self._unbound_stores: return` 之前，保证无 `park` 批次时墓碑仍会过期。`_failed_req_ids` 语义随之收窄为「传输 / 会话失败」，仍供 `lookup()` 拒绝使用。
3. `_poll_once` 对晚到 `fetch` 一个 RTT 终结：在 `for kv_request_id in result.new_fetch_ids` 循环顶部插入 `_reaped_stores` 命中分支，调用既有的 `session.finish_request(kv_request_id)` 走 `finish_request` → `_finalize_outbound` 终态路径（无协议改动、无新 session API），对端同 `tick` 收到 `TransferDoneMsg(success=False)`；同时 `pop` 掉该 `id` 下可能被重新 `park` 的批次并以失败上报，随后 `continue` 而不写入 `_kv_to_session`，让后续 `submit_store` 继续走拒绝路径。
4. `submit_store` 短路已回收 `id`：新增 `if kv_request_id in self._reaped_stores:` 分支，记一条 `warning` 并直接 `JobResult(success=False)` 返回，避免「prefill 超出超时」的生产者把没人能 `fetch` 的块再 `park` 一整个超时周期、持续钉住 `primary tier` 的 CPU 槽位。这一步是在 `review` 之后以独立 `commit 28b92ec` 补上的。
5. 测试与文档配套：`tests/v1/kv_offload/tiering/p2p/test_manager.py` 的 `_make_manager` 桩补上 `_reaped_stores` 与 `_unbound_store_timeout_s`，`test_unbound_store_reaped_after_timeout` 的断言从 `_failed_req_ids` 迁到 `_reaped_stores`；新增 `test_late_fetch_after_reap_fails_immediately`，通过包装 `mgr_a._control._drain_outbound_to` 的 `recording_drain` 记录消费者所有出站消息类型，断言失败结果来自 `TransferDoneMsg`、线上从未出现 `abort_fetch`、墙钟远小于 `_LOAD_TIMEOUT_S`。`docs/features/kv_offloading_usage.md` 在 `secondary_tiers` 配置表新增 `unbound_store_timeout_s` 行（默认 60）并给出 `"unbound_store_timeout_s": 180` 的完整 `--kv-transfer-config` 示例，说明调大它会延长 `primary-tier` 槽位占用。

变更点	文件 / 符号	说明
配置键	<code>manager.py</code> → <code>__init__</code> 、 <code>_unbound_store_timeout_s</code>	先 <code>float()</code> 再校验，NaN 兜底无效值
墓碑表	<code>manager.py</code> → <code>_reaped_stores</code> 、 <code>_REAPED_ID_RETENTION_S</code>	取代 <code>_failed_req_ids</code> 记录回收，带 60 s 剪枝
回收逻辑	<code>manager.py</code> → <code>_reap_unbound_stores</code>	使用实例超时；剪枝墓碑；写墓碑而非失败集合
拒绝路径	<code>manager.py</code> → <code>_poll_once</code>	命中墓碑即 <code>finish_request</code> + 清空重 <code>park</code> 批次
生产侧短路	<code>manager.py</code> → <code>submit_store</code>	已回收 <code>id</code> 立即失败，不再 <code>park</code>

变更点	文件 / 符号	说明
测试	<code>test_manager.py</code>	桩补字段、断言迁移、新增路径级断言用例
文档	<code>docs/features/kv_offloading_usage.md</code>	配置表 + 示例 + 槽位占用权衡说明

关键文件：

- `vllm/v1/kv_offload/tiering/p2p/manager.py`（模块 P2P 卸载层；类别 source；类型 core-logic；符号 `_UNBOUND_STORE_TIMEOUT_S`, `_REAPED_ID_RETENTION_S`, `P2PSecundaryTierManager.init`, `P2PSecundaryTierManager.submit_store`）：全部核心逻辑所在：新增 per-tier 配置键 `unbound_store_timeout_s` 及其强转校验、新增墓碑表 `_reaped_stores` 与 `_REAPED_ID_RETENTION_S`、重写 `_reap_unbound_stores` 的记账方式、在 `_poll_once` 中加入一 RTT 拒绝分支、在 `submit_store` 中加入已回收 id 的短路。同时移除了 `_failed_req_ids` 在回收路径上的无界增长。
- `tests/v1/kv_offload/tiering/p2p/test_manager.py`（模块 P2P 卸载测试；类别 test；类型 test-coverage；符号 `test_late_fetch_after_reap_fails_immediately`, `recording_drain`, `_make_manager`, `test_unbound_store_reaped_after_timeout`）：新增路径级测试 `test_late_fetch_after_reap_fails_immediately`，用 `recording_drain` 包装出站通道记录消费者发出的每条消息类型，断言晚到 `fetch` 走的是 `TransferDoneMsg` 而非超时后 `abort`；同时把 `_make_manager` 桩补齐 `_reaped_stores` / `_unbound_store_timeout_s`，并把 `test_unbound_store_reaped_after_timeout` 的断言从 `_failed_req_ids` 迁到 `_reaped_stores`。
- `docs/features/kv_offloading_usage.md`（模块 配置文档；类别 docs；类型 documentation）：面向用户的配置参考：在 `secondary_tiers` 配置表新增 `unbound_store_timeout_s` 行（默认 60），补充 `--kv-transfer-config` 完整示例（"`unbound_store_timeout_s`": 180）与一段说明，讲清「调大窗口会延长 primary-tier CPU 槽位占用」以及「过期后晚到 `fetch` 一个往返内被拒、消费者立即回退本地 `prefill`」的行为。

关键符号：`P2PSecundaryTierManager.init`, `P2PSecundaryTierManager.submit_store`, `P2PSecundaryTierManager._reap_unbound_stores`, `P2PSecundaryTierManager._poll_once`, `P2PSecundaryTierManager.on_request_finished`, `test_late_fetch_after_reap_fails_immediately`, `recording_drain`, `_make_manager`

关键源码片段

`vllm/v1/kv_offload/tiering/p2p/manager.py`

全部核心逻辑所在：新增 per-tier 配置键 `unbound_store_timeout_s` 及其强转校验、新增墓碑表 `_reaped_stores` 与 `_REAPED_ID_RETENTION_S`、重写 `_reap_unbound_stores` 的记账方式、在 `_poll_once` 中加入一 RTT 拒绝分支、在 `submit_store` 中加入已回收 id 的短路。同时移除了 `_failed_req_ids` 在回收路径上的无界增长。

```
# vllm/v1/kv_offload/tiering/p2p/manager.py
```

```
# 未绑定 store 的默认回收时限：生产者 park 的 KV 块在这段时间内没有等到
# FetchMsg 把它绑定到 session，就会被回收，以免继续占住 primary tier 的 CPU 槽位。
# 该值现在可由 tier 配置键 unbound_store_timeout_s 逐层覆盖。
_UNBOUND_STORE_TIMEOUT_S = 60.0
```

```
# 被回收的 kv_request_id 的“墓碑”保留时长：晚到的 FetchMsg 靠它被一次性拒绝。
# 必须大于消费者侧的 session.client._LOAD_TIMEOUT_S (30 s) ——超过之后消费者
# 已经自行放弃，再保留墓碑也没有人能收到这个拒绝。
_REAPED_ID_RETENTION_S = 60.0
```

```
class P2PSecondaryTierManager(SecondaryTierManager):
```

```
    def __init__(
        self,
        offloading_spec: OffloadingSpec,
        primary_kv_view: memoryview,
        tier_type: str = "p2p",
        host: str | None = None,
        port: int | None = None,
        backends: list[str] | None = None,
        num_threads: int = 4,
        # 新增 per-tier 配置键，形状与 host/port/backends/num_threads 一致；
        # SecondaryTierFactory.create_secondary_tier 以 **config 原样 splat 传入，
        # 所以不能假设类型已经是 float——部署模板常把 JSON 数值渲染成字符串 "180"。
        unbound_store_timeout_s: float = _UNBOUND_STORE_TIMEOUT_S,
        **kwargs: Any,
    ) -> None:
        super().__init__(offloading_spec, primary_kv_view, tier_type)
```

```
        # 先 float() 再判范围：否则 "180" 会死在与 0 的比较上抛 TypeError,
        # 而完全不可转换的值也拿不到带配置键名的报错。NaN 作为“转不动”的哨兵，
        # 因为任何 NaN 比较都为 False，会自然落到下面的 ValueError。
```

```
        try:
            timeout_s = float(unbound_store_timeout_s)
        except (TypeError, ValueError):
            timeout_s = float("nan")
        if not timeout_s > 0:
            raise ValueError(
                f"unbound_store_timeout_s must be a positive number, got "
                f"{unbound_store_timeout_s!r}"
            )
        self._unbound_store_timeout_s = timeout_s
```

```
        self._unbound_stores: dict[str, list[_UnboundStoreBatch]] = {}
        # kv_request_id → 其 park 批次被回收的时刻（墓碑表）。命中这里意味着该 id 下
        # 已经没有任何可服务的块，于是：
        # 1. _poll_once 收到它的 FetchMsg 时当场结束这一轮，不再 park 需求；
        # 2. submit_store 直接失败，不再把没人能取的块重新 park 一个完整超时。
        # 条目由 _reap_unbound_stores 在 _REAPED_ID_RETENTION_S 之后剪掉。
```

```

self._reaped_stores: dict[str, float] = {}

def _reap_unbound_stores(self) -> None:
    """回收从未被任何 peer fetch 过的 submit_store 批次。

    遍历 _unbound_stores，找出最早批次已超过 _unbound_store_timeout_s 的条目：
    丢弃该 kv_request_id、把每个批次以失败上报，并把 id 记入 _reaped_stores，
    这样晚到的 FetchMsg 在一个 RTT 内被拒绝、晚到的 submit_store 也会直接失败。
    同时顺带过期消费者不可能还在等待的墓碑条目——这正是该函数必须放在
    “无待回收批次”提前返回之前的原因。
    """
    now = time.monotonic()
    if self._reaped_stores:
        retention_deadline = now - _REAPED_ID_RETENTION_S
        for kid in [
            kid
            for kid, reaped_at in self._reaped_stores.items()
            if reaped_at <= retention_deadline
        ]:
            del self._reaped_stores[kid]
    if not self._unbound_stores:
        return
    deadline = now - self._unbound_store_timeout_s
    expired: list[str] = []
    for kid, batches in self._unbound_stores.items():
        # 批次按到达顺序追加，队首即最老的一批，回收时钟从首 chunk 起算。
        if batches[0].submitted_at <= deadline:
            expired.append(kid)
    if not expired:
        return
    for kid in expired:
        batches = self._unbound_stores.pop(kid)
        # 关键语义变更：不再写 _failed_req_ids（fetch 路径没有读取者，且
        # on_request_finished 早已跑过，唯一 discard 点不会再命中，集合会
        # 随进程生命周期无界增长），改为写带时间的墓碑。
        self._reaped_stores[kid] = now
        for batch in batches:
            self._finished_jobs.append(
                JobResult(job_id=batch.job_id, success=False)
            )
        logger.warning(
            "P2P %s: unbound store kv_request_id=%s timed out after %s "
            "without a fetch — failing %d job(s)",
            self._local_id,
            kid,
            self._unbound_store_timeout_s,
            len(batches),
        )

def submit_store(self, job_metadata: TransferJob) -> None:

```

```

job_id = job_metadata.job_id
# ... 省略 dest / kv_request_id 解析与既有失败分支 ...
# 该 id 的早期批次已被回收，消费者要么已经放弃、要么会在 fetch 路径上被
# 拒绝；此时继续 park 只会让没人能取的块再占住 primary-tier 槽位一个完整
# 超时周期，所以立刻失败——无论哪种情况，解码端都会回退到本地重算。
if kv_request_id in self._reaped_stores:
    logger.warning(
        "P2P %s: submit_store for reaped kv_request_id=%s job_id=%d "
        "— failing without parking",
        self._local_id,
        kv_request_id,
        job_id,
    )
    self._finished_jobs.append(JobResult(job_id=job_id, success=False))
    return
# Fast path: a session has already received FetchMsg for this id,
# so we can route the batch straight into its ServerRole.
session = self._kv_to_session.get(kv_request_id)

def _poll_once(self) -> None:
    # ... 省略 transport / session 的既有轮询 ...
    for kv_request_id in result.new_fetch_ids:
        if kv_request_id in self._reaped_stores:
            # 这个 id 的块已经被回收，将来也不会有任何 submit_store 能满足
            # on_fetch 刚记录下的需求。就在本 tick 结束这一轮：对端会在同一次
            # poll 内收到 TransferDoneMsg(success=False) 并立即回退本地 prefill,
            # 而不是 park 到自己的 load timeout 才放弃。
            # id 故意不绑定到 session，这样后续 submit_store 继续走拒绝路径；
            # 回收与墓碑剪枝之间被重新 park 的批次在这里一并失败，避免它们
            # 占住槽位再等一个完整超时。
            session.finish_request(kv_request_id)
            for batch in self._unbound_stores.pop(kv_request_id, ()):
                self._finished_jobs.append(
                    JobResult(job_id=batch.job_id, success=False)
                )
            continue
        self._kv_to_session[kv_request_id] = session

```

tests/v1/kv_offload/tiering/p2p/test_manager.py

新增路径级测试 `test_late_fetch_after_reap_fails_immediately`，用 `recording_drain` 包装出站通道记录消费者发出的每条消息类型，断言晚到 fetch 走的是 `TransferDoneMsg` 而非超时后 `abort`；同时把 `_make_manager` 桩补齐 `_reaped_stores` / `_unbound_store_timeout_s`，并把 `test_unbound_store_reaped_after_timeout` 的断言从 `_failed_req_ids` 迁到 `_reaped_stores`。

```

def test_late_fetch_after_reap_fails_immediately(self):
    """针对已被回收的 kv_request_id，fetch 在一个往返内失败。

```

生产者一旦越过 `unbound_store_timeout_s` 就丢弃 park 的块；此后才到达的

fetch 必须在下一个 tick 通过 TransferDoneMsg(success=False) 得知结果,
而不是先卡满 _LOAD_TIMEOUT_S 再走 abort 路径。

```
"""
```

```
from vllm.v1.kv_offload.tiering.p2p.session.client import _LOAD_TIMEOUT_S
```

```
mgr_a, mgr_b = _build_paired_managers()
```

```
kv_id = "req-late"
```

```
# 记录消费者发出的每一条消息类型, 后面的断言用它证明从未需要 abort。
```

```
sent_from_a: list[str] = []
```

```
drain = mgr_a._control._drain_outbound_to
```

```
def recording_drain(peer_local_id: str):
```

```
    out = drain(peer_local_id)
```

```
    sent_from_a.extend(msg.get("type") for _, msg in out)
```

```
    return out
```

```
mgr_a._control._drain_outbound_to = recording_drain # type: ignore[method-assign]
```

```
# 生产者先 park, 再在任何 fetch 到达之前把块回收掉。
```

```
mgr_b.submit_store(
```

```
    _job_metadata(
```

```
        job_id=200,
```

```
        keys=[b"b-block"],
```

```
        chunk_ids=[0],
```

```
        kv_params={"remote_decoder": {"kv_request_id": kv_id}},
```

```
    )
```

```
)
```

```
mgr_b._unbound_store_timeout_s = 0.0
```

```
reap_results = list(mgr_b.get_finished_jobs())
```

```
assert JobResult(job_id=200, success=False) in reap_results
```

```
assert kv_id in mgr_b._reaped_stores
```

```
# 消费者这时才来取已经不存在的块。
```

```
consumer_params = {
```

```
    "remote_prefiller": {
```

```
        "kv_request_id": kv_id,
```

```
        "remote_host": "B",
```

```
        "remote_port": 2,
```

```
    },
```

```
}
```

```
mgr_a.on_new_request(_req_context(consumer_params))
```

```
mgr_a.submit_load(
```

```
    _job_metadata(
```

```
        job_id=101,
```

```
        keys=[b"b-block"],
```

```
        chunk_ids=[0],
```

```
        kv_params=consumer_params,
```

```
    )
```

```
)

started = time.monotonic()
all_a: list[JobResult] = []
for _ in range(8):
    all_a.extend(list(mgr_a.get_finished_jobs()))
    list(mgr_b.get_finished_jobs())
elapsed = time.monotonic() - started

# 断言的是路径而非仅结果：失败来自 TransferDoneMsg，线上从未出现
# abort_fetch，且墙钟远小于消费者的 _LOAD_TIMEOUT_S。
assert JobResult(job_id=101, success=False) in all_a, all_a
assert kv_id in mgr_a._failed_req_ids
assert "abort_fetch" not in sent_from_a, sent_from_a
assert elapsed < _LOAD_TIMEOUT_S
```

评论区精华

讨论集中三条，均由 reviewer orozery 提出（第一条是转述 Claude 的分析），作者 liranschour 三条回复分别为 Done / Fixed / No need, bnellnm 与 orozery 最终均 APPROVED。

1. 「回收后仍继续 park」的漏洞：orozery 引用「`submit_store` never consults `_reaped_stores`, so a producer whose prefill outruns the timeout keeps parking new batches under the reaped id. Fix: also drain/fail `self._unbound_stores.pop(kv_request_id, ())` in the reject branch, and short-circuit `submit_store` on a reaped id.」作者回「Done」，以独立 commit [28b92ec](#) 落地了两处改动，并同步把 `_poll_once` 注释里「keeps taking the unbound path」改成「reject path」。
2. 配置值类型：orozery 指出「Probably worth converting to `float` first (with a try)」，作者回「Fixed」。最终 commit [ee79456](#) 的说明解释得最清楚：值是从用户 JSON 原样 splat 进来的，quoted "180" 会先死在与 0 的比较上抛 `TypeError`，而不可转换的值连键名都报不出来，所以改成先强转、再统一报错。
3. 两个超时是否应互相约束：orozery 建议「consider also rejecting values below `_STORE_TIMEOUT_S`」，作者明确拒绝：「No need. These are 2 unrelated timeouts.」——即未绑定回收时限与单 job 的 store 超时被视为正交，不加入交叉校验。

另有一条流程噪音：claude[bot] 因该 PR 来自 fork 而自动 review 被禁用，需要维护者手动 [@claude review](#)。

- 回收分支需清空重新 park 的批次 (correctness): 作者回复「Done」，最终实现中该分支除 `session.finish_request(kv_request_id)` 外，还会 pop 并以 `JobResult(success=False)` 上报这些批次，随后 continue 保持 id 未绑定。
- `submit_store` 需短路已回收的 `kv_request_id` (correctness): 作者回复「Done」，以独立 commit [28b92ec](#) 加入短路分支：命中 `_reaped_stores` 时记 warning 并直接返回失败结果，`_poll_once` 注释里的「unbound path」也相应改为「reject path」。

- 配置值应先 `float()` 强转再校验 (correctness): 作者回复「Fixed」, 最终 commit `ee79456` 先 `float()`、以 `float('nan')` 兜底不可转换值, 再由 `if not timeout_s > 0` 统一抛带键名的 `ValueError`。
- 是否应拒绝小于 `_STORE_TIMEOUT_S` 的配置值 (design): 作者明确拒绝: 「No need. These are 2 unrelated timeouts.」两个超时被视为正交, 未加入交叉校验; 注释中只保留了「墓碑保留期必须大于消费者 load timeout」这一条依赖说明。

风险与影响

- 风险:
 1. 墓碑保留期与消费者 load timeout 的隐式耦合: `_REAPED_ID_RETENTION_S = 60.0` 只靠注释保证「必须大于」`session.client._LOAD_TIMEOUT_S = 30.0`, 两者是不同模块里的独立常量。若未来消费者侧超时被调大 (或网络延迟让 `fetch` 晚于 `reap + 60 s` 到达), 墓碑已被剪枝, `_poll_once` 会退回旧行为——绑定 `id`、`park` 一个永远无人满足的需求, 消费者再次打满自己的超时。新增的 `test_late_fetch_after_reap_fails_immediately` 只覆盖「reap 后 8 次 poll 内到达」, 没有覆盖墓碑过期后的边界。
 1. 生产者侧失败面与日志量扩大: `submit_store` 对已回收 `id` 每个 job 记一条 warning, 并产出 `JobResult(success=False)`。长 `prefill` + 高并发下, 同一被回收 `id` 会持续触发该分支, warning 日志与失败计数显著增多, 依赖这些 `JobResult` 做统计或告警的上游需要重新校准预期。
 2. 配置校验仍偏宽松: `float()` 接受 `bool` (`float(True) == 1.0` 且 `> 0` 成立), 会让 "`unbound_store_timeout_s: true` 静默变成 1 s 的极端短超时; 也没有对极大的值设上限, 而文档已说明窗口越长 `primary-tier` CPU 槽位被钉得越久。
 3. 回收计时起点未变: 仍按 `batches[0].submitted_at` (首个 `park` 批次) 起算, PR body 明确将其列为有意不改——实测 `idle` 下 `first-chunk` 与 `completion` 只差约 6 s、负载下约 13 s。这意味着真实预算依然是「首 chunk → fetch < 超时」, 长 `prefill` 部署必须主动调大配置, 默认 60 s 在某些场景仍可能偏紧。
 4. 测试桩维护成本: `_make_manager()` 通过 `__new__` 绕过 `__init__`, 新增的 `_reaped_stores` / `_unbound_store_timeout_s` 依赖手工补齐; 后续再往 `__init__` 加实例属性而忘记同步桩, 会以 `AttributeError` 的形式暴露在无关用例上。- 影响: 用户侧: 使用 `OffloadingConnector + TieringOffloadingSpec` (CPU `primary` + `p2p secondary`) 的 PD 分离部署, 在晚到 `fetch` 场景下不再每次都多付约 30 s 停顿加上整个 `prompt` 的重算, 失败在同一个 tick 内可见; 同时获得 `unbound_store_timeout_s` 这一可调旋钮, 长 `prefill` 部署可以从 60 s 上调 (文档给出 180 的示例)。系统侧: 失败路径语义更清晰 (墓碑拒绝 vs 打满超时后 `abort`), 生产者的 `_failed_req_ids` 不再随进程生命周期无界增长; 未绑定批次的语义从「静默 `park` 到超时」变为「超时即墓碑、后续 `store` 直接失败」。团队侧: 新增一个需要理解相对关系的运维参数 (与 `_STORE_TIMEOUT_S`、消费者 `load timeout` 并列), 文档已补充但调优经验仍需积累; 对 KV `offload/tiering` 代码所有者而言, 这次改动把「跨实例失败通知」的实现范式确立为可复用模式。影响范围局限于 `p2p` 次级层, 不影响协议、不影响生成结果 (PR 声明未做模型评测, 因为只改失败路径时序与配置默认值)。- 风险标记: 失败路径语义变更, 跨组件超时耦合无测试覆盖, 配置校验偏宽松, 生产者侧失败日志量上升

关联脉络

- PR #53235 [KVConnector][P2P] 仅通过 `kwargs.pop` 增加 `unbound_store_timeout_s`（材料未提供确切标题）：PR body 明确点名该 PR 与本 PR 不重复：它只用未文档化的 `kwargs.pop` 加了超时旋钮，没有修晚到 `fetch` 的 30 s 消费者停顿，而这才是实际回归；body 还指出它对变更的描述不准确（「keep it until it is safely bound/consumed」实际只是把回收参数化）。
- PR #52912 暖生产者级联 KV 块（标题未在材料中给出）：Issue #53128 引用了该 PR，但本 PR body 说明它处理的是不同区域（让暖生产者级联其块），并不解决「peer 尚未索取的块能存活多久」这一问题。
- PR #56486 [Bugfix][KV Offload] Ignore pending chunks in invalid sliding windows: 同属仓库近期的 KV offload 修复线，两者都在处理「过期的 pending 状态继续阻塞就绪路径」这一类语义问题：一个在 offloading connector 的调度器，一个在 tiering 的 p2p 次级层，可作为同一演进脉络对照阅读。