PR #33473 完整报告

sgl-project/sglang

[HiCache] Batch PP write and load completion sync

合并时间：2026-08-19 11:27

原文链接：http://prhub.com.cn/sgl-project/sglang/pull/33473

执行摘要

- 一句话：HiCache PP 合并写入与加载完成同步，一次 all_reduce 替代两次，吞吐提升 37%
- 推荐动作：值得精读，尤其关注其设计取舍：用单次向量化 all_reduce 替代两次独立同步，并用 ReduceOp.MIN 保证跨 stage 安全推进；同时明确拒绝合并 storage 队列同步以维持 PP storage 语义。测试构造技巧（object.__new__ 绕过 __init__ + MagicMock 精确断言）也值得借鉴。

功能与动机

PR body 明确说明动机：Reduce scheduler overhead for HiCache with pipeline parallelism by batching write and load completion-count synchronization into a single operation. 原实现中 writing_check() 与 loading_check() 各自隐含一次跨 stage 同步，在共享前缀、高并发 prefill-heavy 负载下，这一开销被放大；作者用 3.5K/32K/128K 输入长度的 prefill-only workload 做基准，32K 高压力场景下吞吐从约 494K 提升到 677K tokens/s（+37%）。

实现拆解

本 PR 的变更围绕调度热路径 check_hicache_events() 展开，按以下步骤拆解：

1. 变更入口：UnifiedRadixCache.check_hicache_events()（python/sglang/srt/mem_cache/unified_radix_cache.py）。该方法每个 scheduler step 被调用，轮询 HiCache 异步写 / 读事件。原实现在 pp_size != 1 分支直接调用无参的 writing_check() 与 loading_check()，二者各自在 PP 维度做一次完成计数同步，共两次同步。
2. 核心改造：合并为一次向量 all_reduce。在 pp_size != 1 分支内，先构造长度为 2 的 CPU int 张量 finish_counts；仅当 pp_rank == 0 且 cache_controller 非空时，用 _count_ready_acks() 分别统计 ack_write_queue 与 ack_load_queue 中 finish_event.query() 已就绪的 ACK 数量；随后一次 self._all_reduce(finish_counts, ReduceOp.MIN) 同时完成 CP/TP 归约与 PP 传播；最后 map(int, finish_counts.tolist()) 解包得到 write_finish_count / load_finish_count，并传给 writing_check(finish_count=...) 与 loading_check(finish_count=...)，复用 pp_size == 1 分支已有的带参接口。drain_storage_control_queues() 在 enable_storage 时保持原样，维持 storage 队列的 stage-local 同步。

3. 测试配套：test/registered/unit/mem_cache/test_hiradix_pp_sync_drain.py 新增 TestUnifiedPPSyncBatching。_make_cache() 用 object.__new__(UnifiedRadixCache) 绕过 __init__，以 SimpleNamespace + MagicMock 打桩出 check_hicache_events() 依赖的成员。leader 用例验证 _all_reduce 只调用一次且载荷为 [1, 2]，writing_check(finish_count=1)、loading_check(finish_count=2) 各调用一次；follower 用例验证其不查询本地 ACK 队列，完成数完全来自 all_reduce 填充结果。原有 TestPPSyncDrain 继续保留。
4. CI 配套（非本 PR 功能范围）：PR 过程中因 #31323 在 topk.py 引入违反 global-config 访问约束的 get_server_args() 读取导致 CI 失败，作者在分支上追加 [CI] Read top-k buffer limits from schedule config 修复并在合并 upstream main 时解决冲突；最终 diff 只统计到 2 个文件，topk.py 的修复未体现在最终变更集中。

关键文件：

- python/sglang/srt/mem_cache/unified_radix_cache.py（模块 缓存层；类别 source；类型 core-logic；符号 check_hicache_events, _count_ready_acks, writing_check, loading_check）：核心逻辑所在：check_hicache_events() 的 PP > 1 分支将两次完成计数同步合并为单次向量 all_reduce，是性能收益和语义变化的主要来源。
- test/registered/unit/mem_cache/test_hiradix_pp_sync_drain.py（模块 PP 同步；类别 test；类型 test-coverage；符号 TestUnifiedPPSyncBatching, _make_cache, test_pp_batches_write_and_load_counts_once）：新增 TestUnifiedPPSyncBatching 覆盖 leader 与 follower 两种角色的行为，验证 all_reduce 只调用一次且计数正确，是本次语义变更的主要回归保障。

关键符号：check_hicache_events, _count_ready_acks, writing_check, loading_check

关键源码片段

python/sglang/srt/mem_cache/unified_radix_cache.py

核心逻辑所在：check_hicache_events() 的 PP > 1 分支将两次完成计数同步合并为单次向量 all_reduce，是性能收益和语义变化的主要来源。

```
def check_hicache_events(self) -> None:
    """Called per scheduler step to poll async HiCache events."""
    # 先回收上一轮 PP 同步的异步发送，避免重复计数。
    self._drain_async_work()

    if self.pp_size != 1:
        # PP > 1 时，write/load 完成计数合并成一次向量 all_reduce。
        # 之前这里分别调用 writing_check() 与 loading_check()，
        # 各自会在 PP 维度做一次同步；现在改为：
        # 1) 仅 leader（pp_rank == 0）统计本地 ACK 队列中已就绪的数量；
        # 2) 一次 all_reduce 同时完成 CP/TP 归约与 PP 传播；
        # 3) 各 rank 用 ReduceOp.MIN 得到跨 stage 安全的最小完成数。
        finish_counts = torch.zeros(2, dtype=torch.int, device="cpu")
        if self.pp_rank == 0 and self.cache_controller is not None:
            finish_counts[0] = self._count_ready_acks(
```

```
                self.cache_controller.ack_write_queue
            )
            finish_counts[1] = self._count_ready_acks(
                self.cache_controller.ack_load_queue
            )
        self._all_reduce(finish_counts, torch.distributed.ReduceOp.MIN)
        write_finish_count, load_finish_count = map(int, finish_counts.tolist())
        # 复用非 PP 路径的接口：传入明确的完成数，避免内部再做 PP 同步。
        self.writing_check(finish_count=write_finish_count)
        self.loading_check(finish_count=load_finish_count)
        # storage 控制队列保留 stage-local 同步：
        # hybrid sidecar pool 可能因 PP stage 不同而异，不能跨 stage 合并。
        if self.enable_storage:
            self.drain_storage_control_queues()
    else:
        # pp_size == 1：原逻辑，_sync_hicache_ready_counts 一次返回两组计数。
        (
            write_finish_count,
            load_finish_count,
            storage_queue_sizes,
            extra_pool_names,
        ) = self._sync_hicache_ready_counts()
        self.writing_check(finish_count=write_finish_count)
        self.loading_check(finish_count=load_finish_count)
        # ... 后续 storage 队列处理保持不变
```

test/registered/unit/mem_cache/test_hiradix_pp_sync_drain.py

新增 TestUnifiedPPSyncBatching 覆盖 leader 与 follower 两种角色的行为，验证 all_reduce 只调用一次且计数正确，是本次语义变更的主要回归保障。

```
class TestUnifiedPPSyncBatching(unittest.TestCase):
    def _make_cache(self, pp_rank, write_ready, load_ready):
        # 用 object.__new__ 绕过 __init__，只装配 check_hicache_events 需要的成员。
        cache = object.__new__(UnifiedRadixCache)
        cache.tree_core = SimpleNamespace(enable_storage=False)
        cache.pp_rank = pp_rank
        cache.pp_size = 2
        cache.enable_storage_metrics = False
        cache.storage_metrics_collector = None
        # 用 MagicMock 隔离外部依赖，精确断言调用次数与参数。
        cache._drain_async_work = MagicMock()
        cache._all_reduce = MagicMock()
        cache.writing_check = MagicMock()
        cache.loading_check = MagicMock()
        cache.drain_storage_control_queues = MagicMock()
        # 每个 ACK 的 finish_event.query 返回 ready 标志，模拟已完成的异步写 / 读。
        cache.cache_controller = SimpleNamespace(
            ack_write_queue=[
                SimpleNamespace(
```

```
                finish_event=SimpleNamespace(query=MagicMock(return_value=ready))
            )
            for ready in write_ready
        ],
        ack_load_queue=[
            SimpleNamespace(
                finish_event=SimpleNamespace(query=MagicMock(return_value=ready))
            )
            for ready in load_ready
        ],
    )
    return cache

def test_pp_batches_write_and_load_counts_once(self):
    # leader：write 队列 [True, False] 就绪 1 个；load 队列 [True, True] 就绪 2 个。
    leader = self._make_cache(0, [True, False], [True, True])
    leader.check_hicache_events()

    # 关键断言：write/load 只做一次向量 all_reduce，不再各自同步。
    leader._all_reduce.assert_called_once()
    self.assertEqual(leader._all_reduce.call_args.args[0].tolist(), [1, 2])
    leader.writing_check.assert_called_once_with(finish_count=1)
    leader.loading_check.assert_called_once_with(finish_count=2)

    # follower：不查询本地队列，完成数完全来自 all_reduce 结果。
    follower = self._make_cache(1, [True], [True])
    follower._all_reduce.side_effect = lambda counts, _: counts.fill_(1)
    follower.check_hicache_events()

    for queue in (
        follower.cache_controller.ack_write_queue,
        follower.cache_controller.ack_load_queue,
    ):
        queue[0].finish_event.query.assert_not_called()
    follower._all_reduce.assert_called_once()
    follower.writing_check.assert_called_once_with(finish_count=1)
    follower.loading_check.assert_called_once_with(finish_count=1)
```

评论区精华

核心讨论围绕 " 能否进一步合并同步 " 展开：

> stepinto（reviewer）：I think we could merge this with the else branch (line 1955..1979), which could further more reduce the number of synchronizations.
>
> luoroger37（作者）：Lines 1955–1979 also include the pp_size == 1 storage-queue path. For pp_size > 1, storage queues are synchronized stage-locally because hybrid sidecar pools may differ across PP stages. Merging that whole branch would therefore change the PP storage semantics. I'll keep those lines unchanged and

> limit this PR to batching write/load completions for pp_size > 1.
>
> stepinto：Fine. Just leave it as it. Thank you. cc @hzh0425

作者明确拒绝了合并 else 分支的建议，理由是 storage 队列在 PP 下是 stage-local 语义，合并会改变 HiCache 的存储同步行为，最终 reviewer 接受该取舍并批准。另外，issue 评论中 MichoChan 询问 benchmark 启动参数，可见记录中未见作者回复。

- 是否合并 pp_size==1 分支的 storage 同步以进一步减少同步次数 (design): 不合并；本 PR 仅批处理 write/load completion，storage 队列同步保持原样。
- benchmark 启动参数 (question): 未在可见材料中找到回复，benchmark 可复现性存疑。

风险与影响

- 风险：
  - 语义变化点：原来每个 PP rank 各自检查本地 ACK 队列后推进；现在只有 leader 统计，follower 完全依赖 all_reduce 结果。若 leader 的 cache_controller 为 None（代码加了保护但仍会退化为全 0），或 leader 的队列计数为 0，所有 rank 都会以 0 推进，与旧路径行为存在差异，但保守方向安全。
  - ReduceOp.MIN 的正确性依赖：MIN 保证不会超前于任何 stage 的完成进度，但如果 follower 通常更快完成，leader 统计会成为保守瓶颈，可能抵消部分性能收益；测试只覆盖了 leader/follower 数量一致与全 1 填充两种情形，未覆盖 leader 为 None 或队列为空的全 0 退化路径。
  - 热路径回归风险：check_hicache_events() 每个 scheduler step 调用，改动位于 PP > 1 分支，PP = 1 路径未触碰；但 writing_check / loading_check 的带参接口语义在 PP 分支上复用，需要 HiCache PP 场景的回归验证。
  - benchmark 覆盖有限：唯一量化证据是 32K 输入高压力场景（+37%），3.5K 与 128K 未见明确数据，且启动参数未在可见记录中回复，可复现性存疑。
- 影响：
  - 性能影响：使用 HiCache 且开启 PP 的部署，每个 scheduler step 的 PP 同步次数从 2 次降为 1 次；在共享前缀、高并发 prefill 负载下吞吐提升显著（32K 场景 +37%）。
  - 系统影响：unified_radix_cache.py 是 HiCache 与 UnifiedRadixCache 共用的中枢，check_hicache_events() 是调度热路径，改动会影响所有启用 HiCache PP 的部署；storage 队列同步语义保持不变，降低了存储侧回归面。
  - 团队影响：本次讨论确立了 "leader 统计 + 单次向量 all_reduce + MIN 归约 " 的同步范式，并明确了 storage 队列必须保持 stage-local 的边界，为后续进一步减少 L3 同步留下演进路径。
  - 风险标记：核心调度热路径变更，依赖 ReduceOp.MIN 语义，仅 leader 统计 ACK 队列，benchmark 覆盖有限，storage 队列语义保持不变

关联脉络

- PR #34798 [HiCache] Buffer-only mode for HiCache host memory layer: 同一 HiCache 功能线，且同样修改 unified_radix_cache.py，与本 PR 在缓存路径上直接关联。

- PR #35130 Fix NIXL cleaner grouping for hybrid cache keys: HiCache 存储侧（NIXL cleaner）修复，与本 PR 的 storage 队列 stage-local 语义讨论属于同一子系统。
- PR #35164 Refactor kv cache event mixin into a recorder: KV 缓存事件基础设施重构，涉及 mem_cache 下多个文件，与本 PR 的缓存事件同步路径有重叠。