PR #33370 完整报告

sgl-project/sglang

[Feature] Add process-local in-memory KV indexer and Router integration

合并时间：2026-08-20 10:45

原文链接：http://prhub.com.cn/sgl-project/sglang/pull/33370

执行摘要

本 PR 为实验性 KV Indexer 交付一个更小、无外部依赖的里程碑：在 experimental/sgl-router 工作区中新增 sgl-kv-indexer crate，用进程内 RwLock 内存后端取代原 Redis 方案，并与 Router 的 cache_aware_zmq 策略打通。端到端链路为 worker 通过 ZMQ 发出 KV 事件，Bridge 以 gRPC 转发，Indexer 记录块级组件放置并通过 MatchExternalKvPrefix 回答最长可复用前缀，Router 据此把请求路由到缓存命中 worker。索引器默认不生效，并以软状态和降级语义为代价换来可独立运行、可测试的单进程基线。

功能与动机

KV Indexer 是 RFC #31458 提出的元数据服务：跟踪哪个 worker 在哪个 tier 上缓存了哪些内容寻址 KV 块，使 Router 能把请求路由到缓存命中概率最高的 worker。此前 #32662 依赖 Redis / Dragonfly / Redis Cluster，部署重、难测试。本 PR 移除所有外部存储，把放置元数据保存在单个 Indexer 进程内，使 event → index → prefix-query 核心链路更小、更易运行，并直接可测。PR body 明确这是 "validated single-process soft-state baseline"：持久化、恢复与多副本属于后续工作。

实现拆解

1. 新增 crate 与契约：sgl-kv-indexer 成为 Router workspace 成员，共享 lockfile 与工具链。crate 内包含 proto 契约、Bridge、Service、内存后端、客户端和 kv-indexer-server 入口；CI 增加 protoc 安装。
2. 事件接收与转发（Bridge）：bridge.rs 订阅 ZMQ KV-event 流，解码为外部 KV action 后通过 ApplyExternalKvBatch 转发；超大批次按 MAX_ACTIONS_PER_BATCH 与 MAX_HASHES_PER_REQUEST 切分为有序 RPC，保持动作顺序。错误分类区分永久与瞬时错误，断线按 500 ms–10 s 退避重连，但刻意不做事件重放。
3. 进程内放置索引（memory_backend）：memory_backend.rs 以 RwLock<State> 保存块哈希 → (worker, tier) 组件掩码、worker 地址与 cache spec、以及按 tier 的反查索引。apply() 是唯一写路径，REPORT 为替换式快照，CLEAR_ALL_AT_TIER 通过 holdings 批量回收；读路径在一次读锁快照内构造所有候选 worker 的组件视图。
4. 组件感知前缀匹配（service）：service.rs 定义 KvIndexerBackend trait，默认的 match_external_kv_prefix 即前缀语义的权威定义，逐块应用 FULL / SWA / MAMBA 规则并返回每个 worker 的最长连续可复用前缀。协议层在进入后端前强制资源上限（16,384 hashes / 256 actions / 8 MiB 解码限额）。

5. Router 集成与降级：CLI 新增 --kv-indexer-endpoint、--kv-indexer-query-timeout-ms、--kv-indexer-query-max-inflight 等参数。cache_aware_zmq.rs 的 select_external 将外部前缀信号与候选 worker 求交，在最优前缀持有者中按最小活动负载选择；无信号（unreachable / timeout / overloaded / query-too-large / empty）一律降级 min-load 并 WARN，只有被拒（Rejected）才 503。启用外部 Indexer 后 Router 停止维护本地 radix 树。
6. 测试与 CI 配套：新增内存后端集成测试、gRPC 契约测试、deadline 脱落测试、CLI 校验测试，以及覆盖真实 worker → ZMQ → Bridge → Indexer → Router 的 E2E GPU 测试；同步更新文档与 workspace 配置。

关键源码片段

experimental/sgl-router/sgl-kv-indexer/src/service.rs

定义 gRPC 服务 trait 与协议边界，默认实现组件感知前缀匹配的权威语义，并强制请求级资源上限。

```
/// 存储后端 trait。所有变更走 `apply_external_kv_batch`，保持唯一有序写路径。
/// async 允许未来接入 I/O 后端；dyn-safe 让服务端持有 `Arc<dyn KvIndexerBackend>`。
#[tonic::async_trait]
pub trait KvIndexerBackend: Send + Sync + 'static {
    // 应用整个 KVEventBatch。动作预校验，必须按序应用；seq 仅作参考。
    async fn apply_external_kv_batch(
        &self,
        request: ApplyExternalKvBatchRequest,
    ) -> Result<ApplyExternalKvBatchResponse, Status>;

    async fn match_external_kv(
        &self,
        request: MatchExternalKvRequest,
    ) -> Result<MatchExternalKvResponse, Status>;

    // 收集各 worker 的组件放置数据，与 hashes 对齐。
    // 默认实现是组件盲的：持有块都作为 legacy 整块放置。
    // 组件感知后端应覆盖此方法并附加 WorkerCacheSpec 与驻留组件集。
    async fn collect_worker_prefix_inputs(
        &self,
        hashes: &[i64],
    ) -> Result<Vec<WorkerPrefixInput>, Status> {
        let matched = self
            .match_external_kv(MatchExternalKvRequest {
                hashes: hashes.to_vec(),
                count_as_hit: false,
            })
            .await?;
        Ok(legacy_inputs_from_match(hashes, &matched))
    }

    /// 回答每个 worker 持有的最长可复用请求前缀。
```

```
    /// 默认实现本身就是前缀语义的"书面定义"；后端若为性能覆盖，必须保持
    /// 逐字段一致（仅允许 `blocks_read` 不同，它是可观测性而非语义）。
    async fn match_external_kv_prefix(
        &self,
        request: MatchExternalKvPrefixRequest,
    ) -> Result<MatchExternalKvPrefixResponse, Status> {
        let limit = prefix_limit(request.hashes.len(), request.max_blocks);
        let hashes: Vec<i64> = request.hashes.into_iter().take(limit).collect();
        if hashes.is_empty() {
            return Ok(MatchExternalKvPrefixResponse::default());
        }
        let inputs = self.collect_worker_prefix_inputs(&hashes).await?;
        Ok(compute_prefix_response(&inputs, hashes.len() as u32))
    }
}
```

experimental/sgl-router/sgl-kv-indexer/src/admission.rs

查询路径过载保护：基于调用方 deadline 做脱落，并用翻倍计数抑制日志风暴。

```
//! 基于调用方截止时间的查询路径负载脱落。
//! 一个等待超过调用方完整截止时间的查询已经无法被有价值地回答，
//! 服务它只会拖延其余积压。预算来自调用方自己的 `grpc-timeout`；
//! 没有服务器侧阈值需要调，未声明截止时间的调用方永不被脱落。

use std::sync::atomic::{AtomicU64, Ordering};
use std::time::{Duration, Instant};
use tonic::metadata::MetadataMap;
use tonic::{Extensions, Request, Status};

/// 请求到达时刻，在请求头被读取之后打上。
#[derive(Clone, Copy)]
struct Arrival(Instant);

/// 按类型统计拒绝次数，并在总数翻倍时上报，避免日志风暴。
pub(crate) struct RejectionLog(AtomicU64);

impl RejectionLog {
    pub(crate) const fn new() -> Self { Self(AtomicU64::new(0)) }

    // 记录一次拒绝；在总数达到 2 的幂时返回总数供日志使用。
    pub(crate) fn record(&self) -> Option<u64> {
        let total = self.0.fetch_add(1, Ordering::Relaxed) + 1;
        total.is_power_of_two().then_some(total)
    }
}

static DEADLINE_SHED_LOG: RejectionLog = RejectionLog::new();

// 在请求被调度前打上到达时间戳；没有该拦截器就不会发生脱落。
```

```
pub fn stamp_arrival(mut request: Request<()>) -> Result<Request<()>, Status> {
    request.extensions_mut().insert(Arrival(Instant::now()));
    Ok(request)
}

// 对耗时超过调用方声明 deadline 的查询返回脱落。
// 绝不用于 apply 路径：丢弃 KV 事件会让索引与 worker 报告永久分叉。
pub(crate) fn reject_if_deadline_passed(
    metadata: &MetadataMap,
    extensions: &Extensions,
) -> Result<(), Status> {
    let (Some(arrival), Some(budget)) = (extensions.get::<Arrival>(), caller_deadline(metadata))
    else {
        return Ok(());
    };
    let waited = arrival.0.elapsed();
    if waited < budget {
        return Ok(());
    }
    if let Some(shed_total) = DEADLINE_SHED_LOG.record() {
        tracing::info!(
            shed_total,
            waited_ms = waited.as_millis(),
            budget_ms = budget.as_millis(),
            "shedding prefix query whose caller deadline already passed"
        );
    }
    Err(Status::deadline_exceeded(
        "prefix query waited longer than its caller deadline",
    ))
}
```

评论区精华

hzh0425（memory_backend.rs）：为什么限制扫描长度为 2048 块？Router 用全量请求长度计算匹配率，未扫描块会被当作 miss，4096 块的完全缓存请求会显示成 50% 命中并回退到 min-load 路由。 wuyl1：2048 上限是 Redis 时代为保住 10 ms 截止时间引入的；进程内后端可移除上限，改为按 16,384 哈希分块扫描，并在跨块保持匹配状态；加上回归测试。 hzh0425（client.rs）：page_size=1 时 payload 过大可能造成 gRPC 通信失败，是否假定 page_size 默认为 64？ wuyl1：不假定 page_size；tonic 0.14.6 默认 4 MiB 接收限制，约 19 万 hashes 触发。建议把 hash 从约 22 字节的字符串改为打包 sfixed64（8 字节），并把接收上限提到 8 MiB，支持约 100 万块。 hzh0425：采用这个简单的方案？ wuyl1：已在 39b5bdeaaa 落地，并补充大请求回归测试。 hzh0425（CI）：现有 Tier-3 E2E 只跑了 Router 既有测试，没有配置外部 Indexer endpoint 或启动 Bridge；需要真实 worker → ZMQ → Bridge → Indexer → Router 全链路测试。 wuyl1：已新增 E2E 测试，验证路由到缓存 worker。

风险与影响

- 软状态与可用性：Indexer 进程重启丢失全部放置元数据；Bridge 断连期间事件不恢复，缓存亲和度暂时降级为无信号状态。
- 单进程 RwLock 瓶颈：apply 串行，查询共享读锁；在 worker 数或事件频率极高时可能成为热点，需要后续演进。
- gRPC 缓冲风险：8 MiB 解码限额限制了单条消息，但 MAX_CONCURRENT_STREAMS=64 仅约束并发流，解码仍由 tonic codec 在方法体之前完成，对端可让服务器缓冲 8 MiB × 64 的字节。
- Router 行为变化：启用外部 Indexer 后本地 radix 树不再维护；endpoint 不可拨号时启动即失败（有意设计）。
- 构建依赖：proto 编译需要 protoc，本地开发与 CI 需安装。

关联脉络

本 PR 是 #31458 的延续，取代 #32662，并把后端从 Redis 收敛到进程内内存；同时消费 #32514 引入的 BlockStored.component_types，以及 #32537 修复的重计算 GPU 块事件，形成 worker 事件源 → 索引 → 路由的完整闭环。后续持久化、事件回放、分布式多副本和故障转移都是自然的演进方向。