PR #25851 完整报告

sgl-project/sglang

sgl-router: experimental Rust HTTP router for SGLang worker pools

合并时间：2026-05-25 15:34

原文链接：http://prhub.com.cn/sgl-project/sglang/pull/25851

执行摘要

本 PR 在 experimental/sgl-router/ 下引入了一个独立的 Rust HTTP 路由代理，用于替代 Python 的 model-gateway 侧车（SMG），作为 SGLang 推理节点池的前端路由层。支持 K8s/ 静态文件发现、多策略路由（含 ZMQ KV 缓存感知策略）、PD 分离调度、断路器、活跃负载跟踪和 Prometheus 指标。当前为草案状态（158 commits，28623 行新增），按里程碑（M0~M6）组织，便于增量审查。

功能与动机

sgl-router 的目标是提供与 SMG 相同的功能，但作为单一直立 Rust 二进制程序，更轻量且无 Python 运行时依赖。通过 ZMQ 订阅 SGLang 工作节点的 KV 缓存事件，实现缓存感知路由——将共享相同前缀的请求导向已经缓存了该前缀的 worker，减少预填计算。同时支持 PD（预填 - 解码）分离部署，在 K8s 环境中自动发现和分类 worker。

> PR body: “Targets the same role as the model-gateway sidecar (SMG) but as a single self-contained Rust binary instead of a Python service.”

实现拆解

1. 项目脚手架与配置：在 experimental/sgl-router/ 下建立 Cargo workspace、工具链锁定和CI工作流。src/config/types.rs定义Config结构（含server、observability、models、discovery、proxy、active_load 等区块），通过 serde 支持 YAML/TOML 加载。PolicyKind 枚举将路由策略序列化为蛇形字符串，拒绝未知值。
2. 发现与工作节点注册：实现 Discovery trait 和 DiscoveryEvent（Added/Removed/ModeChanged）。K8s 后端（src/discovery/k8s.rs）通过 kube-rs 监听 EndpointSlice，使用客户侧标签匹配区分 Plain/Prefill/Decode 模式。WorkerManager 消费发现事件，维护 WorkerRegistry（DashMap），并在注册时通过 /server_info 内省补充模型名、事件配置和分离角色。移除早期的 static_file 后端，替换为更简洁的 static_urls。
3. 路由策略与断路器：四种路由策略——RoundRobin、Random、PowerOfTwoChoices 和 CacheAwareZmq（src/policies/cache_aware_zmq.rs）。缓存感知策略首先检查负载不均衡（绝对值＋相对阈值），若严重不均衡则直接选最低负载worker；否则tokenize请求体、计算块哈希、查询共享哈希树 (HashTree) 获取最长前缀匹配，若匹配率超过 cache_threshold 则选该 worker，否则回退为最低负载。断路器实现三状态机（Closed/Open/HalfOpen），cool_down 后允许单个探测请求。
4. KV 事件管道：src/policies/kv_events/ 下的四个模块：

- wire.rs：解析 SGLang ZmqEventPublisher 的 msgspec.msgpack 格式事件。
- subscriber.rs：为每个 (worker, dp_rank) 创建 ZMQ SUB 连接，消息解码后送入共享 mpsc channel。
- tree.rs：哈希键基数树，每个节点代表一个块哈希，维护持有该块的 worker 集合，支持最长前缀匹配。
- index.rs：KvEventIndex 串联订阅者与树，处理 worker 生命周期和事件泵。

5. HTTP 代理与聊天处理：src/proxy/mod.rs 实现 reqwest 转发和 SSE 流泵（64 槽有界通道，backpressure 到上游）。src/server/routes/chat.rs 处理 /v1/chat/completions：解析 model 字段 → 策略选择 → 获取候选 worker（普通模式或 PD 隔离池）→ PD 模式下注入 bootstrap_port 和 room_id → 预填请求后携解码亲和性发起解码请求。非流式和流式均支持。
6. 观测性与运维：/metrics 端点暴露 Prometheus 格式指标。支持 JSON 日志。信号优雅关闭（SIGTERM 后等待在途请求完成）。Docker 镜像构建（Helm chart 后续移除）。测试包含单元 / 组件 / 代理 / 端到端层级，K8s kind 集成测试和 4×H200 真实 GPU 验证。

experimental/sgl-router/src/discovery/k8s.rs

K8s 发现后端核心实现，包含 client-side 角色分类、EndpointSlice 到 WorkerSpec 的转换逻辑。

```
// files: experimental/sgl-router/src/discovery/k8s.rs
// 根据 K8s 发现模式将 EndpointSlice 分类为 WorkerMode
fn classify_mode(es: &EndpointSlice, mode: &K8sDiscoveryMode) -> Option<WorkerMode> {
    match mode {
        K8sDiscoveryMode::Plain { .. } => Some(WorkerMode::Plain),
        K8sDiscoveryMode::PdDisaggregation {
            prefill_selector,
            decode_selector,
        } => {
            let labels = es.metadata.labels.as_ref().cloned().unwrap_or_default();
            if labels_match_selector(&labels, prefill_selector) {
                Some(WorkerMode::Prefill)
            } else if labels_match_selector(&labels, decode_selector) {
                Some(WorkerMode::Decode)
            } else {
                None // 两个选择器都不匹配，过滤掉此分片
            }
        }
    }
}

// 匹配 K8s 标签集与逗号分隔的相等选择器（支持 `=` 和 `==`）
fn labels_match_selector(labels: &BTreeMap<String, String>, selector: &str) -> bool {
    for term in selector.split(',') {
        let term = term.trim();
        if term.is_empty() { continue; }
        let (key, expected) = if let Some((k, v)) = term.split_once("==") {
```

```
            (k.trim(), v.trim())
        } else if let Some((k, v)) = term.split_once('=') {
            (k.trim(), v.trim())
        } else {
            return false;
        };
        match labels.get(key) {
            Some(v) if v == expected => {}
            _ => return false,
        }
    }
    true
}
```

experimental/sgl-router/src/policies/active_load.rs

RAII 守卫和 janitor 机制，提供双轴活跃负载跟踪，是缓存感知策略的负载信号源。

```
// files: experimental/sgl-router/src/policies/active_load.rs
// 为每个在途请求创建的唯一标识符
#[derive(Clone, Eq, Hash, PartialEq, Debug)]
pub struct RequestId(pub Uuid);

impl RequestId {
    pub fn new_v4() -> Self {
        Self(Uuid::new_v4())
    }
}

// 每个工作节点的计数器：预填负载和解码负载分开跟踪
#[derive(Debug, Default)]
struct WorkerCounters {
    prefill_load: AtomicUsize,
    decode_load: AtomicUsize,
}

// 在途请求的元数据，janitor 通过此结构找到过期请求并触发取消
#[derive(Debug)]
struct RequestEntry {
    worker: WorkerId,
    cancel: CancellationToken, // janitor 发送取消信号
    counters: Arc<WorkerCounters>, // 直接持有 Arc，防止 worker 移除后下溢
    registered_at: Instant, // 注册的时间戳
}
```

experimental/sgl-router/src/config/types.rs

定义完整的配置 schema、策略枚举和默认值，是路由器的配置入口。

```
// files: experimental/sgl-router/src/config/types.rs
// 顶层配置结构，通过 serde 反序列化，支持 YAML/TOML 格式
```

```rust
#[derive(Debug, Clone, Serialize, Deserialize)]
pub struct Config {
    pub server: ServerConfig,
    #[serde(default)]
    pub observability: ObservabilityConfig,
    pub models: Vec<ModelConfig>,
    pub discovery: DiscoveryConfig,
    #[serde(default)]
    pub proxy: ProxyConfig,
    #[serde(default)]
    pub active_load: ActiveLoadConfig,
}

// 代理超时配置，默认 300 秒，匹配 SGLang 预填 / 解码延迟预算
#[derive(Debug, Clone, Copy, Serialize, Deserialize)]
pub struct ProxyConfig {
    #[serde(default = "default_proxy_request_timeout_secs")]
    pub request_timeout_secs: u64,
}
fn default_proxy_request_timeout_secs() -> u64 { 300 }

// 活跃负载跟踪超时，默认 600 秒，高于 proxy 超时，确保 proxy 超时优先触发
#[derive(Debug, Clone, Copy, Serialize, Deserialize)]
pub struct ActiveLoadConfig {
    #[serde(default = "default_stale_request_timeout_secs")]
    pub stale_request_timeout_secs: u64,
}
fn default_stale_request_timeout_secs() -> u64 { 600 }

// 路由策略枚举，使用 `#[serde(rename_all = "snake_case")]` 拒绝未知值
#[derive(Debug, Clone, Copy, PartialEq, Eq, Default, Serialize, Deserialize)]
#[serde(rename_all = "snake_case")]
pub enum PolicyKind {
    #[default]
    RoundRobin,
    Random,
    PowerOfTwo,
    CacheAwareZmq, // 需要模型加载 tokenizer，依赖 cache_aware 配置
}
```

评论区精华

PR 为草案，无正式代码审查评论。关键设计讨论出现在 PR body 的已知限制部分：

> “K8s + PD e2e in a real cluster: integration test covers the wiring; needs a follow-up e2e test…” “Mode flip after registration (worker rolls prefill → decode without restart): neither backend nor manager notices. Out of scope; document as known limitation.”

作者在 M4 review-findings 提交中自审了 9 项修复，包括断路器 allow() 解耦为 would_allow()、ActiveLoadGuard 使用 Option<RegistryHandle> 避免双 decrement、PD 池隔离时区分 NoHealthyWorkersInPool 和 NoHealthyWorkers 等。

风险与影响

- 新组件风险：全新代码，未经长期生产验证。
- 版本兼容性风险：与 SGLang 的 ZMQ 事件格式紧耦合，SGLang 改动可能导致路由器失效。
- 测试覆盖局限：K8s+PD 真实集群 *e2e* 未运行；GPU *e2e* 仅覆盖单路由器 / 单模型。
- 性能风险：哈希树读写锁可能在写密集场景成为瓶颈。
- 安全风险：缺少身份认证和请求验证（仅 1 MiB body 限制）。

不影响现有 SGLang 用户，但需要团队增加 Rust 维护能力。

关联脉络

本 PR 独立引入新组件，与历史 PR 无直接依赖关系。但在更宏观的演进中，sgl-router 的 PD 分离能力与近期 DeepSeek-V4（#25948）、KV 缓存事件基础设施（#26177）和调度优化（#25404）构成互补，共同完善 SGLang 的分布式推理路由能力。