PR #26252 完整报告

sgl-project/sglang

[observability] add Ray metric backend wrappers

合并时间：2026-06-18 10:41

原文链接：http://prhub.com.cn/sgl-project/sglang/pull/26252

执行摘要

- 一句话：为 SGLang 指标新增 Ray 后端包装器，支持集成到 Ray 监控系统。
- 推荐动作：值得精读，尤其是包装器设计（复制 + 标签绑定、边界过滤、名称净化）和测试策略（fake 模块注入实现 CPU 环境测试）。对于计划将 SGLang 嵌入 Ray Serve 的用户，本 PR 提供了零配置接入路径。

功能与动机

下游平台希望将 SGLang 指标通过 Ray 的 ray.util.metrics 代理展示，避免单独部署 prometheus_client；此前只能 fork 或自行封装。本 PR 应此需求，提供官方 Ray 后端。

实现拆解

1. 在 ray_wrappers.py 中实现四个包装器类（RayCounterWrapper、RayGaugeWrapper、RayHistogramWrapper、RaySummaryWrapper），每个继承自基类 RayPrometheusMetric，将 prometheus_client 的 inc/set/observe 调用转换为 ray.util.metrics 对应操作，并自动注入 ReplicaId 标签以区分 Ray Serve 副本。
2. 创建五个收集器子类（如 RaySchedulerMetricsCollector），继承自 MetricsCollector 模块的对应父类，仅覆盖父类使用的 _xxx_cls 类属性，实现类级别 DI。PR#24610 的 _StatLoggerDIMixin 自动处理其余属性默认值，无需修改仪表点。
3. 处理 Ray 平台差异：RaySummaryWrapper 因 Ray 无 Summary 原语，降级为 Histogram 并用保守默认边界；实现 _coerce_positive_boundaries 过滤非正值边界（如 queue_time_seconds 的 0.0 下界）；实现 _get_sanitized_opentelemetry_name 将冒号等标点替换为下划线，符合 OpenTelemetry 命名规范。
4. 新增测试基础设施 fake_ray.py，通过 make_fake_ray_modules 创建伪造的 ray.util.metrics 和 ray.serve 模块，注入 sys.modules，使单元测试无需安装 Ray 即可运行。该 fake 与 DI 单元测试共享。
5. 调整现有测试：在 test_metrics.py 中新增基于真实 Engine + FileRecordingMetric 的跨进程集成测试，验证 DI 发射流经 Ray 包装器；精简 test_stat_loggers_di.py 为纯 CPU 单元测试，仅验证类属性默认值和角色解析逻辑。

关键文件：

- python/sglang/srt/observability/ray_wrappers.py（模块 可观测性；类别 source；类型 dependency-wiring；符号 _get_replica_id, RayPrometheusMetric, init, _get_tag_keys）

：核心实现：定义 RayPrometheusMetric 基类、四个包装器、五个收集器子类，以及工具函数（边界过滤、名称净化）。所有 Ray 指标集成逻辑集中于此文件。

- test/registered/unit/observability/test_ray_wrappers.py（模块 测试；类别 test；类型 test-coverage；符号 TestRayWrapperBase, setUp, tearDown, TestNameSanitization）：单元测试：覆盖包装器的所有操作（inc/set/observe）、标签绑定、名称净化、边界过滤、DI 类的属性覆盖，以及无 Ray 环境导入。
- python/sglang/test/observability/fake_ray.py（模块 测试辅助；类别 test；类型 test-coverage；符号 FakeRayMetric, make_fake_ray_modules, FakeRayServeException, _ReplicaCtx）：测试基础设施：提供 FakeRayMetric 和 make_fake_ray_modules，使单元测试和 DI 集成测试无需安装 Ray 即可运行。
- test/registered/observability/test_metrics.py（模块 测试；类别 test；类型 test-coverage；符号 _FileRecordingMetric, _FileRecordingMetricBound, _record, _RecordingSchedulerCollector）：集成测试：新增基于 Engine 的 DI 验证，使用 FileRecordingMetric 记录指标发射，跨进程验证 Ray 包装器正确接入。
- test/registered/unit/observability/test_stat_loggers_di.py（模块 测试；类别 test；类型 test-coverage；符号 TestCollectorClassAttrs, TestResolveCollectorClass, TestRoleConstants）：DI 单元测试：验证类属性默认值、resolve_collector_class 逻辑、角色常量映射。重构后精简为纯 CPU 测试。

关键符号：_get_replica_id, RayPrometheusMetric.init, RayPrometheusMetric._get_tag_keys, RayPrometheusMetric._build_tags, RayPrometheusMetric.labels, RayPrometheusMetric._coerce_positive_boundaries, RayPrometheusMetric._get_sanitized_opentelemetry_name, RayCounterWrapper.inc, RayGaugeWrapper.set, RayHistogramWrapper.observe, RaySummaryWrapper.observe, FakeRayMetric.init, make_fake_ray_modules, FileRecordingMetric.labels, _FileRecordingMetricBound.inc, _RecordingSchedulerCollector.init

关键源码片段

python/sglang/srt/observability/ray_wrappers.py

核心实现：定义 RayPrometheusMetric 基类、四个包装器、五个收集器子类，以及工具函数（边界过滤、名称净化）。所有 Ray 指标集成逻辑集中于此文件。

```
def _get_replica_id() -> Optional[str]:
    """Return the current Ray Serve replica ID, or None outside Serve."""
    if ray_serve is None:
        return None
    try:
        # 注意：ray_serve.get_replica_context().replica_id 已经是字符串，无需 .unique_id
        return ray_serve.get_replica_context().replica_id
    except ray_serve.exceptions.RayServeException:
        return None


class RayPrometheusMetric:
    """Base wrapper exposing prometheus_client API on Ray metrics."""
```

```
    _is_labeled: bool = False

    def __init__(self) -> None:
        if ray_metrics is None:
            raise ImportError("RayPrometheusMetric requires Ray to be installed.")
        self.metric: Optional[Metric] = None
        # 初始化时注入 ReplicaId 标签
        self._tags: dict = {"ReplicaId": _get_replica_id() or ""}

    @staticmethod
    def _get_tag_keys(labelnames: Optional[List[str]]) -> tuple:
        """将用户标签键列表末尾追加 ReplicaId，返回完整元组。"""
        labels = list(labelnames) if labelnames else []
        labels.append("ReplicaId")
        return tuple(labels)

    def _build_tags(self, *labels: str, **labelskwargs: str) -> dict:
        """构建标签字典：位置参数按顺序填充键，关键字参数直接合并，ReplicaId 始终由内部设置。"""
        if labels:
            # 位置参数个数等于标签键数减 1（ReplicaId 自动填充）
            expected = len(self.metric.tag_keys) - 1 # 注意：Ray 属性名为 tag_keys
            if len(labels) != expected:
                raise ValueError(
                    f"Number of labels must match the number of tag keys. "
                    f"Expected {expected}, got {len(labels)}"
                )
            labelskwargs.update(zip(self.metric.tag_keys, labels))
        # 每次调用重新获取 ReplicaId，保证标签最新
        labelskwargs["ReplicaId"] = _get_replica_id() or ""
        return {k: v if isinstance(v, str) else str(v) for k, v in labelskwargs.items()}

    def labels(self, *labels: str, **labelskwargs: str) -> RayPrometheusMetric:
        """返回带绑定标签的包装器副本，禁止二次调用 labels()。"""
        if self._is_labeled:
            raise ValueError("labels() cannot be called on an already-labeled metric.")
        clone = copy.copy(self)
        clone._tags = self._build_tags(*labels, **labelskwargs)
        clone._is_labeled = True
        return clone
```

python/sglang/test/observability/fake_ray.py

测试基础设施：提供 FakeRayMetric 和 make_fake_ray_modules，使单元测试和 DI 集成测试无需安装 Ray 即可运行。

```
class FakeRayMetric:
    """Stand-in for ray.util.metrics.{Counter, Gauge, Histogram}.

    记录每次 inc / set / observe 调用及其标签，方便测试断言。
    """
```

```
    def __init__(self, name="", description="", tag_keys=(), boundaries=None):
        self.name = name
        self.description = description
        self.tag_keys = tuple(tag_keys) # 使用 tag_keys 匹配 Ray API
        self.boundaries = list(boundaries) if boundaries is not None else None
        self.calls = [] # 列表，元素为 (op, value, tags)

    def inc(self, value, tags=None):
        self.calls.append(("inc", value, dict(tags or {})))

    def set(self, value, tags=None):
        self.calls.append(("set", value, dict(tags or {})))

    def observe(self, value, tags=None):
        self.calls.append(("observe", value, dict(tags or {})))


def make_fake_ray_modules(replica_id: str = "test-replica") -> dict:
    """创建伪造的 ray 模块，注入 sys.modules，使得测试代码可以透明使用。"""
    # 创建 ray.util.metrics 模块并将三个 metric 类型设置为 FakeRayMetric
    ray_util_metrics = types.ModuleType("ray.util.metrics")
    ray_util_metrics.Counter = FakeRayMetric
    ray_util_metrics.Gauge = FakeRayMetric
    ray_util_metrics.Histogram = FakeRayMetric
    ray_util_metrics.Metric = FakeRayMetric

    # 模拟 ray.serve 上下文
    ray_serve = types.ModuleType("ray.serve")
    ray_serve_exc = types.ModuleType("ray.serve.exceptions")
    ray_serve_exc.RayServeException = FakeRayServeException
    ray_serve.exceptions = ray_serve_exc

    class _ReplicaCtx:
        # 真实 Ray 中 replica_id 是字符串；此处保持与实现一致
        replica_id = replica_id

    ray_serve.get_replica_context = lambda: _ReplicaCtx()
    return {
        "ray": ...,
        "ray.util": ...,
        "ray.util.metrics": ray_util_metrics,
        "ray.serve": ray_serve,
        "ray.serve.exceptions": ray_serve_exc,
    }
```

评论区精华

Review 中 Gemini Code Assist 指出两个高优先级 bug：`_get_replica_id()` 中 `replica_id` 已是字符串不应使用 `.unique_id`；Ray Metric 类属性是 `tag_keys` 而非 `_tag_keys`。提交者均已修

正。审核者 sufeng-buaa 建议将 Engine 级 DI 测试整合到 test_metrics.py 以利用 FakeRayMetric 进行完整发射验证，作者采纳并重构了测试分层。

- Ray replica_id 属性访问错误 (correctness): 作者已修复，移除 .unique_id 访问。
- tag_keys 属性名错误（下划线前缀） (correctness): 作者已修复所有引用和测试 fake 中的属性名。
- 建议重构 Engine 级 DI 测试到 test_metrics.py (testing): 作者执行了迁移，在 test_metrics.py 中添加了 _FileRecordingMetric 和 _RecordingSchedulerCollector，通过跨进程文件标记验证发射值。

风险与影响

- 风险：风险较低：Ray 为可选依赖，模块在无 Ray 环境可导入但构造时抛出清晰 ImportError；Histogram 边界过滤可能略微改变统计特性（影响极低）；ReplicaId 标签在非 Serve 环境中为空字符串；默认行为不变，已有 prometheus_client 后端不受影响。测试覆盖 31 个单元测试和跨进程集成测试，已覆盖边界值和错误路径。
- 影响：对用户：启用 Ray 指标集成只需在 ServerArgs.stat_loggers 中映射对应收集器类。对团队：需维护与 vLLM 对齐的包装器代码，但核心逻辑封装在单一文件中。对系统：无运行时性能开销，仅增加可选导入分支。测试保障了主要功能和回归防止。
- 风险标记：可选依赖 Ray，Histogram 边界过滤可能改变分布，包装器与 prometheus_client 后端并行，ReplicaId 标签非 Serve 环境为空

关联脉络

- PR #24610 [observability] add class-level DI on MetricsCollectors and ServerArgs.stat_loggers role-to-subclass map: 本 PR 依赖 PR#24610 的 _StatLoggerDIMixin 和 stat_loggers 映射机制，提供具体的 Ray 后端实现。