Ray worker metric

Date	Editor	Change List
2020/07/06	Lingxuan Zuo/ Zhongchun Yu	Create

1. Worker Metric

Worker metric includes two parts: **Metric Wrapper** and **Metric Registry**.

- Metric Wrapper encapsulates the stats interface in the core worker and provides JNI or Cython api for upper level workers.
- To avoid overhead caused by high frequency JNI calls, Metric Registry is proposed to register and aggregate metrics in the java worker, and finally write the metrics to the underlying stats module. **The python worker just makes cython calls without the registry** (Cython calls are efficient, so only implementing metric registry for the java worker).

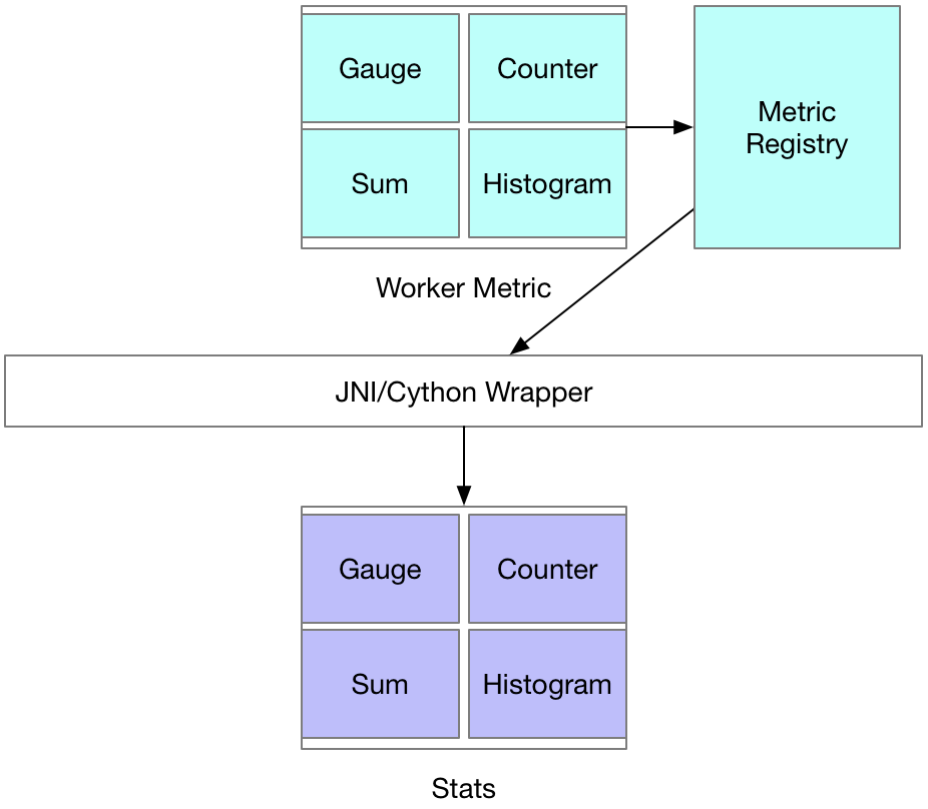
2. Metric Wrapper

At the Ray core level, a stats module has been implemented to collect and report data metrics. Currently, stats is only used in raylet. Both core worker and upper-layer java/python worker are temporarily unavailable. Considering that all components of ray and even the user layer also reuse this module, so there are two tasks in metric roadmap:

- Enable the stats module in other components, including raylet/gcs server/core worker
- Provide metric APIs for worker

How to use metric when core worker stats has been enabled:

- Core worker provides JNI/cython to build wrapper for user-level stats API (each tag and metric actually mapped to an object in cpp)
    - Declare Tag
    - Declare Metric
    - Metric Registry provides unified Metric/Tag lifecycle management to worker



There are some restrictions when binding Java/Python workers to Core workers:

- The lifecycle of tags and metrics must be consistent with that of workers or actors.
- Tag and metric objects created on stack are not recommended since they will not be released automatically.
- All metrics and tags need to be explicitly released at the end of their lifecycle (created in register method and released when shutdown).
- It's heavy overhead to call JNI frequently (**Optimization in MetricRegistry periodic/batch updating**)

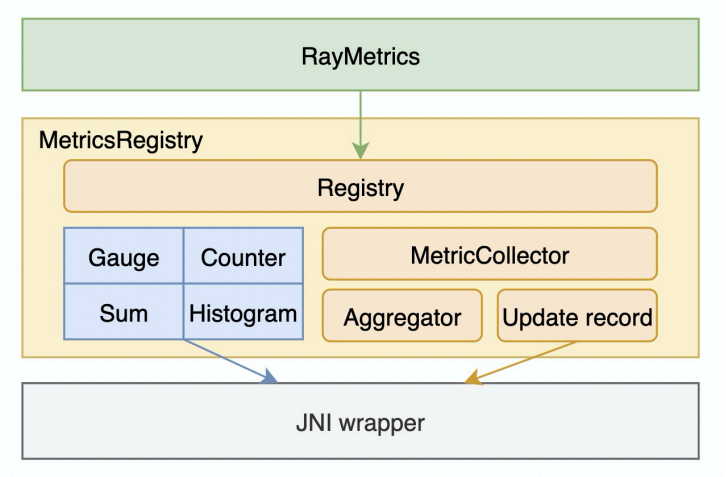
Take the example of using mapping relationship to implement Gauge in Java:w

```
public class TagKey {
    private long tagKeyPtr;
    private String tagKey;
    public TagKey(String key) {
        this.tagKey =  key;
        tagKeyPtr = registerTagKeyNative(key);
    }
    private long native registerTagkeyNative(String tagKey);
    public void unregisterTagKey();
    private void native unregisterTagKey();
```

```
    public long getNativePointer() {
        return tagKeyPtr;
    }

}

interface Metric {
    // Sync metric with core worker stats for registry.
    void record();
    // Update metric info for user.
    void update(double value);
    void update(double value, Map<TagKey, String> tags);
}

public class Gauge implements Metric {
    private String name;
    private double value;
    private long gaugePtr;
    Map<TagKey, String> tags;
    public Gague(String name, String description, String unit, Map<TagKey,
String> tags) {
        gaguePtr = registerGaugeNative(name, description, uint,

tags.keys().stream().map(TagKey::getNativePointer));
        this.name = name;
        this.tags = tags;
    }
    private long native registerGaugeNative(String name, String description,
                                            String unit, List<long>
nativeTagKeyPtrList);
    public void update(double value) {
        this.value = value;
    }
    public void update(double value, Map<TagKey, String> tags) {
        this.value = value;
        this.tags = tags;
    }
    public void record() {
        // Get tag key list from map;
        List<long> nativeTagKeyPtrList;
        // Get tag value list from map;
        List<String> tagValues;
        recordNative(gaugePtr, value, nativeTagkeyPtrList, tagValues);
    };
    private void native recordNative(long gaugePtr, double value);
    private void native record(long gaugePtr, double value,
                               List<long> nativeTagKeyPtrList,
                               List<String> tagValues);
}
```

3. Metric Registry

Metric Registry module is proposed for the java worker, which provides metrics register interface, calculates aggregations by a pre-defined aggregator(like sum, avg, max, min) and collectors registered metrics to update records through JNI by a background thread.



- RayMetrics: metric entry provides four basic metric types (Counter, Sum, Gauge, Histogram)
- MetricRegistry: metrics registry collects, aggregates, and invokes jni for metrics updating.
    - Registry: There are four kinds of metrics: Counter, Sum, Gauge and Histogram.
    - Aggregator: metric Aggregator. If users register a metric, they can specify the metric aggregation window.
    - Metric Collector: a calculation/updating thread used to aggregate values and update metric through JNI.

Usage

Take Java as an example

// Default updating interval is 5 seconds without explicit initialization of MetricRegistry.

```
RayMetrics.gauge()
    .name("test_gauge") // metric name
    .description("test gauge metric")  // metric description
    .tags(ImmuableMap.of("tag1", "value1", "tag2", "value2")) // metric
tags
    .register();
```

Users might also manually initialize RayMetrics in custom config before using MetricRegistry, as follows:

```
MetricConfig config =
```

```
MetricConfig.builder().timeIntervalMs(100).create();
RayMetrics.init(config);
RayMetrics.gauge()
    .name("test_gauge") // metric name
    .description("test gauge metric")  // metric description
    .tags(ImmuableMap.of("tag1", "value1", "tag1", "value2")) // metric
tags
    .register();
RayMetrics.shutdown()
```

4. Plan

- Enable core worker stats module
- Wrapping stats for JAVA worker
- Metric Registry for JAVA worker
- Wrapping stats for Python worker