FLIP-485: UDF metrics

Table of Contents

Motivation

User-defined functions (UDFs) are widely used in data processing systems to allow custom computations, but they introduce significant challenges when it comes to debugging and performance optimization. UDFs often act as black boxes, making it difficult to trace issues or identify the root causes of failures, especially when exceptions occur. The lack of dedicated metrics to quantify critical aspects such as processing time and exception count complicates debugging as well as performance tuning and resource allocation. In addition, UDF metrics act as a critical identifier for autosizer so having those metrics available are critical in ensuring the stability and health for the entire platform.

Public Interface

Additional config to enable / disable UDF metrics.

Proposed Change

The following metrics will be added and available to all user UDFs. Since per-message metrics can significantly affect performance, those metrics should only be enabled via config and should be disabled by default.

Metric Name	**Type**	**Description**
UDFprocessingTime	Gauge	Measures the per record processing time for UDF at per TM

		level
UDFexceptionCount	Counter	Count the number of exceptions, it will be a Gauge in the code but in Observe we use regex to get the delta per minute. In the end Flink and user will get the exception count per minute

Implementation

To get UDF metrics at per TM UDF level, we define UdfMetricGroup with the following hierarchy and all metrics under this group should have the suffix “UDF”.

TaskManagerMetricGroup
- TaskManagerJobMetricGroup
    - TaskMetricGroup
        - UdfMetricGroup

Since per message metrics can impact performance, introduce enableUdfMetrics config and set the value to false by default.   When initializing TM metrics through the instantiateTaskManagerMetricGroup method, we would only instantiate and register UdfMetricGroup if the config is set to true.

```
None
instantiateTaskManagerMetricGroup() {
   ...
   if (enableUdfMetrics) {
       MetricGroup udfMetrics = xxx;
        instantiateProcessTime()
        instantiateExceptionCount();
   }
  ...
}
```

Registration for the metrics will be done via MetricRegistry and will be reported through MetricReporter.

With UDFs, we expect user to extend the UDF base classes and most of the time they will override the execution methods (ie. open()) in UDF base class. Therefore the measurement of exception count or processing time cannot be added to those methods directly.   To overcome this, we set the value for those metrics at CodeGen, so that these metrics are always available regardless of function overrides.

The following Scala classes are entrypoint to UDF CodeGen, and each of them will need to have access to runtimeContext since getMetricGroup() method can only be called if a method extends RichFunction and has access to runtimeContext. Access to context info is needed in order to get UDF metrics for each TM and set the value for those.

```
CommonExecMatch.java (pass in Flink context classLoader to MatchCodeGenerator)
  -> MatchCodeGenerator.scala
      -> ExprCodeGenerator.scala
            -> TableFunctionCallGen.scala
            -> ScalarFunctionCallGen.scala
```

Note that we will need to get runtimeContext via reflection.

The implementation for setting / incrementing metric values is trivial.

Rejected Alternatives

For this solution, we add metrics directly into each of the UDF base classes

Pros	**Cons**
- Logic is straightforward and implementation would be fast and simple	- Maintenance overhead: for each of the new UDF base added we need to add metrics - Limited options to user and we will consistently need to add additional classes to address user needs

Testing

1. Unit test
2. Test branch in LinkedIn OSS version and will roll out to LinkedIn internal users
3. [Stretch] Performance test
    a. Need to enhance benchmark services to measure performance downgrade