Epic - Siddhi Query Execution Optimization with JIT Code Generation.

JIT code generation is used to gain optimizations by applying it in the places where some part of the code is frequently executed. In java at runtime java code is converted into java bytecode in order to run on the java virtual machine. So in JIT compilation frequently executed code blocks are converted into java bytecode to execute in the runtime. As a result of it there is an optimization due to reduction of time to convert java code to java bytecode.

In Siddhi a query is processed through a pipeline. So as a result of it some parts of the pipeline can be used to JIT code generation. Siddhi filter and Siddhi pattern are the places in the pipeline where JIT code generation can be applied. With the increment of complexity of a Siddhi filter query or a Siddhi pattern checking query, JIT code generation will give better optimizations.

JIT Code Generation - Siddhi Filter

Overview
As system, I want to replace currently existing tree structure of Siddhi, to filter events by JIT code generation, so that I can increase transactions per second value of a Siddhi query which has filters.

Prerequisites

1. 'ExpressionExecutor' for Siddhi filter expression has to be available.
2. At least one event has to be arrived to 'FilterProcessor'.
3. 'ASM' library has to be properly configured.

Narrative

1. System creates a class which implements interface 'ExpressionExecutor', after the arrival of initial 'ComplexEvent' to 'FilterProcessor'.
2. System adds instructions to 'execute' method of the created class according to the 'ExpressionExecutor' using 'ASM' library. If 'ExpressionExecutor' is a,
    - Constant Expression Executor.
      System takes value from the 'ExpressionExecutor' and puts it into the 'execute' method of the created class.

    - Variable Expression Executor.
      System takes 'position' array from the 'ExpressionExecutor' and puts it into the 'execute' method of the created class in order to take value from 'ComplexEvent' in runtime.

    - Siddhi Extensions.

System takes the ‘ExpressionExecutor’ of the extension and inserts it into an ArrayList inside the created class. System will select related ‘ExpressionExecutor’ for extension from the ArrayList in runtime and will execute it.

- Compare Condition Expression Executor.
  System takes two child nodes(‘ExpressionExecutors’) of the current ‘ExpressionExecutor’. Then System generates byte-code for the two child nodes.System inserts instructions into the ‘execute’ method of the created class to check whether two values obtained from child nodes are null and if null to give result as ‘false’. Then system inserts instructions into the ‘execute’ method of the created class to handle comparing operation. System follows this procedure for all ‘ExpressionExecutors’ which extend ‘CompareConditionExpressionExecutor’.

- Mathematical Expression Executors.
  System takes the ‘ExpressionExecutor’ of the mathematical expression executors and inserts it into an ArrayList inside the created class. System will select related ‘ExpressionExecutor’ for the mathematical expression executor from the ArrayList in runtime and will execute it.

- AND Expression Executor.
  System takes two child nodes(‘ExpressionExecutors’) of the current ‘ExpressionExecutor’. Then System generates byte-code for the two child nodes.System inserts instructions into the ‘execute’ method of the created class to check whether left result or right result obtained from child nodes is null and if null to give result as ‘false’. Then system inserts instructions into the ‘execute’ method of the created class to do ‘AND’ operation for two results. Also system inserts additional instructions into the ‘execute’ method of the created class to skip second ‘AND’ operator of two consecutive ‘AND’ operators if first ‘AND’ if ‘false’.

- OR Expression Executor.
  System takes two child nodes(‘ExpressionExecutors’) of the current ‘ExpressionExecutor’. Then System generates byte-code for the two child nodes.System inserts instructions into the ‘execute’ method of the created class to check whether left result or right result obtained from child nodes is null and if null to give result as ‘false’. Then system inserts instructions into the ‘execute’ method of the created class to do ‘OR’ operation for two results. Also system inserts additional instructions into the ‘execute’ method of the created class to skip second ‘OR’ operator of two consecutive ‘OR’ operators if first ‘OR’ is ‘true’.

- NOT Expression Executor.
  System takes child node('ExpressionExecutor') of the current 'ExpressionExecutor'. Then System generates byte-code for the child node.System inserts instructions into the 'execute' method of the created class to check whether current result obtained from child node is null and if null to give result as 'false'. Then system inserts instructions into the 'execute' method of the created class to do 'NOT' operation for the current result. Also system inserts additional instructions into the 'execute' method of the created class to skip 'NOT' operation if two consecutive 'NOT' operators are met.

- Bool Expression Executor.
  System takes child node('ExpressionExecutor') of the current 'ExpressionExecutor'. Then System generates byte-code for the child node.System inserts instructions into the 'execute' method of the created class to check whether current result obtained from child node is null and if null to give result as 'false'. Then system inserts instructions to give Boolean value of the current result.

- Is Null Expression Executor.
  System takes child node('ExpressionExecutor') of the current 'ExpressionExecutor'. Then System generates byte-code for the child node.System inserts instructions to check whether current result obtained from child node is null or not if null to give result as 'true'.

- Is Null Stream Expression Executor.
  System takes 'eventPosition' array from the 'ExpressionExecutor' and puts it into the 'execute' method of the created class. Then system inserts instructions to check whether event stream is null or not by using 'eventPosition'.

- In Condition Expression Executor.
  System takes the 'ExpressionExecutor' and inserts it into the ArrayList inside the created class. System will execute that 'ExpressionExecutor' in runtime and will obtain results.

3. System instantiates an object of the created class using a class loader.
4. System executes the 'execute' method of the instantiated object and finds whether current 'ComplexEvent' fulfills the filter expression .
5. System will execute the 'execute' method of the already instantiated object for future 'ComplexEvent' arrivals.

Acceptance Criteria

1. Results of JIT code generated Siddhi and results of currently existing Siddhi should be similar for a sample query.
2. JIT code generated Siddhi should support Siddhi extensions that can be used inside Siddhi filter.
3. ‘Null’ values should be properly handled.
    - ‘Null’ values should be properly handled when ‘VariableExpressionExecutor’ being used near a ‘CompareConditionExpressionExecutor’.

    - ‘Null’ values should be properly handled when ‘VariableExpressionExecutor’ being used near ‘AndConditionExpressionExecutor’, ‘OrConditionExpressionExecutor’, ‘NotConditionExpressionExecutor’ and ‘BoolConditionExpressionExecutor’.