UDFsUsingScriptingLanguages

This document captures the specification for using UDFs using scripting languages, this document will document the syntax, usage details and performance numbers for using this feature. This is tracked at PIG-928.

UDFs Using Scripting Languages

Pig needs to support user defined functions written in different scripting languages such as Python, Ruby, Groovy. Pig can make use of different modules such as jython, jruby which make these scripts available for java to use. Pig needs to support ways to register functions from script files written in different scripting languages as well as inline functions to define these functions in pig script.

Preparation

Before incorporating scripting language into the pig script make sure you local as well as Hadoop environment has access to classes specific to that language. To ensure that edit **hadoop-env.sh** file and added required **CLASSPATH**.

Syntax

Registering scripts

```
Register 'test.py' using jython as myfuncs;
```

This uses org.apache.pig.scripting.jython.JythonScriptEngine to interpret the python script. Users can develop and use custom script engines to support multiple programming languages and ways to interpret them. Currently, pig identifies jython as a keyword and ships the required scriptengine (jython) to interpret it.

Following syntax is also supported -

```
Register 'test.py' using org.apache.pig.scripting.jython.JythonScriptEngine as myfuncs;
```

myfuncs is the namespace created for all the functions inside test.py.

A typical test.py looks as follows -

```
#!/usr/bin/python

@outputSchema("word:chararray")
def helloworld():
  return ('Hello, World')

@outputSchema("t:(word:chararray,num:long)")
def complex(word):
  return (str(word),long(word)*long(word))

@outputSchemaFunction("squareSchema")
def square(num):
  return ((num)*(num))

@schemaFunction("squareSchema")
def squareSchema(input):
  return input

# No decorator - bytearray
def concat(str):
  return str+str
```

Registering test.py with pig makes under myfuncs namespace creates functions - myfuncs.helloworld(), myfuncs.complex(2), myfuncs.square(2.0) available as UDFs. These UDFs can be used with

```
b = foreach a generate myfuncs.helloworld(), myfuncs.square(3);
```

Decorators and Schemas

For annotating python script so that pig can identify their return types, we use python decorators to define output schema for a script UDF. '''outputSchema''' defines schema for a script udf in a format that pig understands and is able to parse.

'''outputFunctionSchema''' defines a script delegate function that defines schema for this function depending upon the input type. This is needed for functions that can accept generic types and perform generic operations on these types. A simple example is "square" which can accept multiple types. SchemaFunction for this type is a simple identity function (same schema as input).

'''schemaFunction''' defines delegate function and is not registered to pig.

When no decorator is specified, pig assumes the output datatype as bytearray and converts the output generated by script function to bytearray. This is consistent with pig's behavior in case of Java UDFs.

"Sample Schema String" - y:{t:(word:chararray,num:long)}, variable names inside schema string are not used anywhere, they are used just to make syntax identifiable to the parser.

Inline Scripts

As of today, Pig doesn't support UDFs using inline scripts. This feature is being tracked at PIG-1471.

Sample Script UDFs

Simple tasks like string manipulation, mathematical computations, reorganizing data types can be easily done using python scripts without having to develop long and complex UDFs in Java. The overall overhead of using scripting language is much less and development cost is almost negligible. Following are a few examples of UDFs developed in python that can be used with Pig.

```
mySampleLib.py
----------------------
#!/usr/bin/python

###################
# Math functions #
###################
#Square - Square of a number of any data type
@outputSchemaFunction("squareSchema")
def square(num):
  return ((num)*(num))
@schemaFunction("squareSchema")
def squareSchema(input):
  return input

#Percent- Percentage
@outputSchema("t:(percent:double)")
def percent(num, total):
  return num * 100 / total

####################
# String Functions #
####################
#commaFormat- format a number with commas, 12345-> 12,345
@outputSchema("t:(numformat:chararray)")
def commaFormat(num):
  return '{:,}'.format(num)

#concatMultiple- concat multiple words
@outputSchema("t:(numformat:chararray)")
def concatMult4(word1, word2, word3, word4):
  return word1+word2+word3+word4

########################
# Data Type Functions #
########################
#collectBag- collect elements of a bag into other bag
#This is useful UDF after group operation
@outputSchema("bag:{(y:{t:(word:chararray)}}")
def collectBag(bag):
  outBag = []
  for word in bag:
    tup=(len(bag), word[1])
    outBag.append(tup)
  return outBag

# Few comments-
# pig mandates that a bag should be a bag of tuples, python UDFs should follow this pattern.
# tuple in python are immutable, appending to a tuple is not possible.
```

Performance

Jython

References

1. PIG-928, "UDFs in scripting languages", https://issues.apache.org/jira/browse/PIG-928
2. Jython, "The jython project", http://www.jython.org/
3. Jruby, "100% pure-java implementation of ruby programming language", http://jruby.org/
4. PIG-1471, "inline UDFs in scripting languages", https://issues.apache.org/jira/browse/PIG-1471