PyRDF

github.com/shravan97/PyRDF

- A Python library for distributed execution of ROOT RDataFrame analysis

Contributors

- Student
  - Shravan Murali
- Mentors
  - Enrico Guiraud
  - Enric Tejedor Saavedra
  - Diogo Castro
  - Prasanth Kothuri
  - Danilo Piparo
  - Javier Cervantes

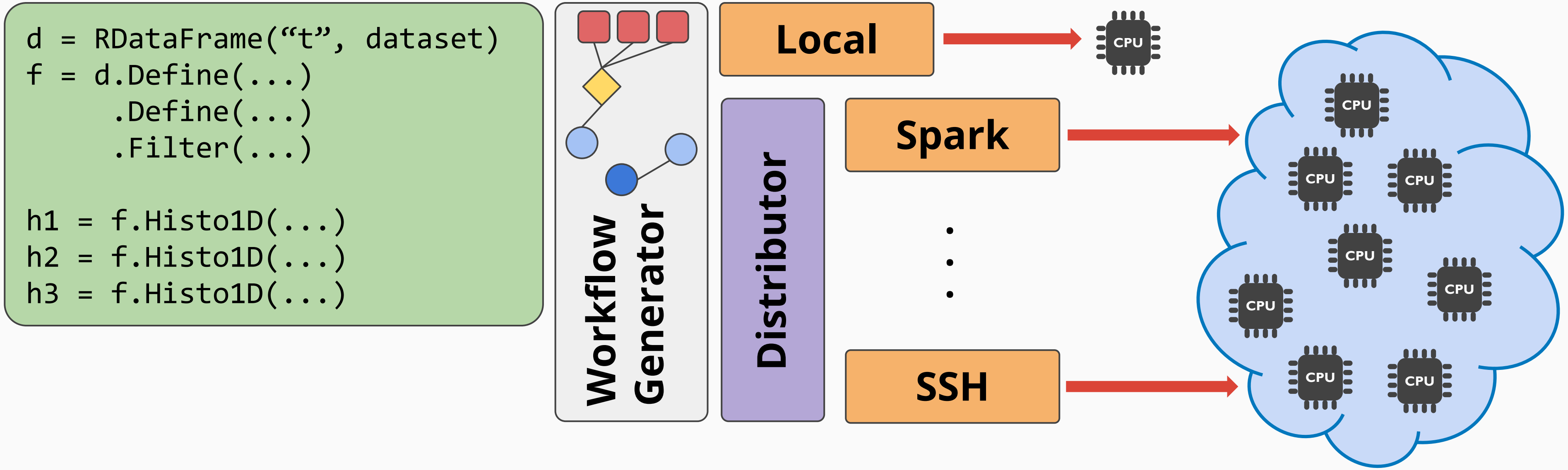
ROOT and RDataFrame

ROOT is a modular scientific software framework. It provides all the functionalities needed to deal with big data processing, statistical analysis, visualisation and storage. It is written in C++ and available for use in Python as PyROOT. [https://root.cern.ch/ for more info]

RDataFrame helps users to represent and operate on datasets easily. This is available in PyROOT.

My work in brief

A Python layer named PyRDF was implemented on top of ROOT's RDataFrame structure. The objective of PyRDF is to make it possible for RDataFrame analysis to run on distributed resources with no changes to its original code



```
d = RDataFrame("t", dataset)
f = d.Define(...)
     .Define(...)
     .Filter(...)

h1 = f.Histo1D(...)
h2 = f.Histo1D(...)
h3 = f.Histo1D(...)
```

Features

- Select and configure the execution backend you want to use.

```
PyRDF.use('spark', {'npartitions':4, 'spark.executor.instances':5})
```

- Execute your analyses both locally and on Spark without changing your RDataFrame code

  (example for this in the next slide)

- Easily include C++ header files to be used from your analysis

```
PyRDF.include("/path/to/A.cpp")
```

Usage

- *Choosing a backend*
  - You'll just need to pass in the backend name and additionally the config parameters as a dictionary
- *Instantiate the RDataFrame object*
  - This is just a mock of ROOT's RDataFrame constructors. In fact here, you can replace C++ vectors with Python lists !
- Define your analysis code and display
  - This is again very similar to ROOT's RDataFrame API

```python
import PyRDF

# Choose your backend
PyRDF.use('spark')

# Initialize a RDataFrame object
rdf = PyRDF.RDataFrame(...args...)

# Define your operations
rdf_op = rdf.Define(...)
rdf_filtered = rdf_op.Filter(...)
my_histogram = rdf_filtered.Histo1D(...)

# Simply Display your required output
my_histogram.Draw()
```

Computation is triggered **only after** the ***my_histogram.Draw()*** call !

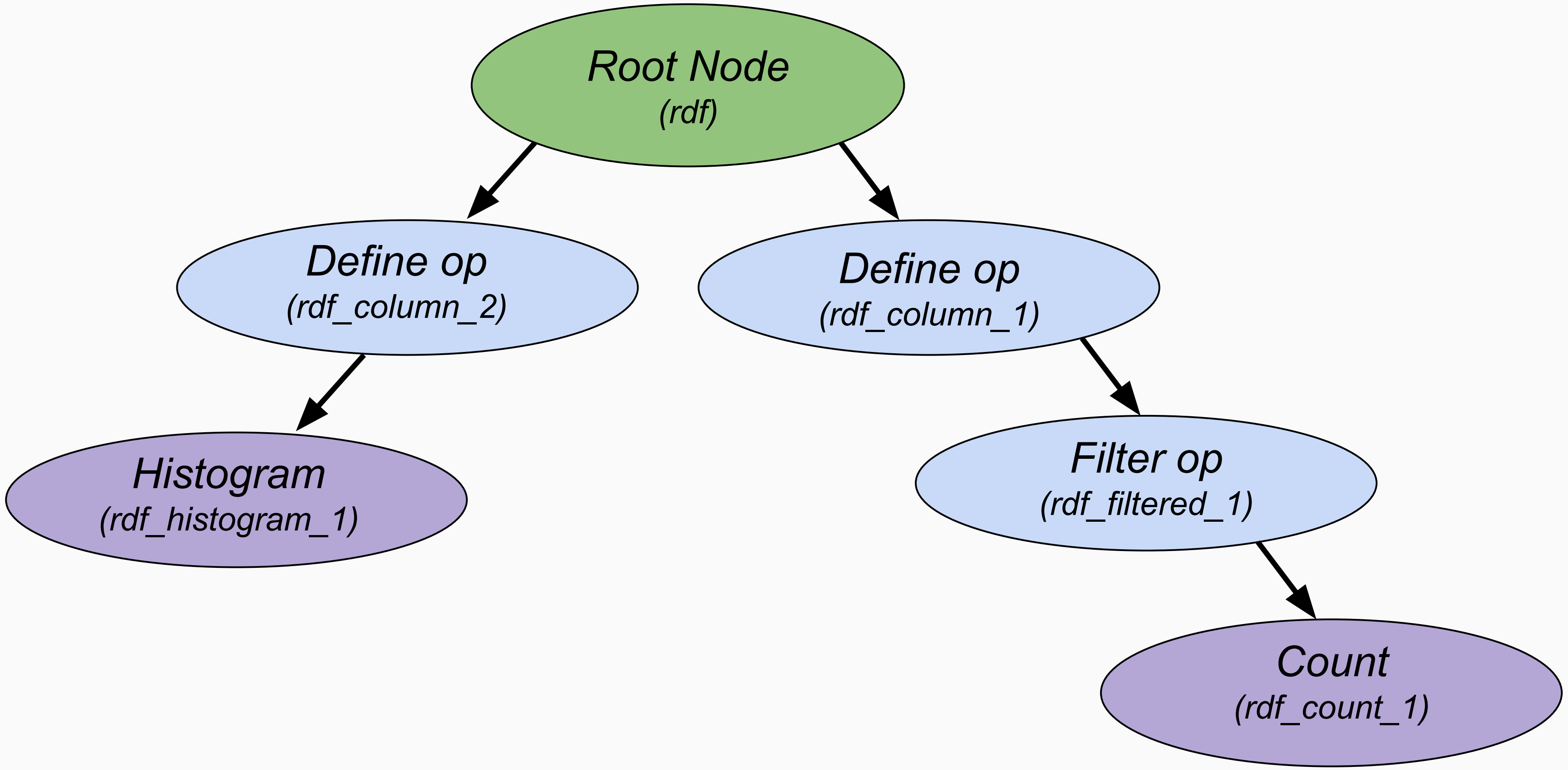
Computational graph illustration

```python
import PyRDF

# Initialize a RDataFrame object
rdf = PyRDF.RDataFrame(...args...) # This is the head node

# Define your operations
rdf_column_1 = rdf.Define(...) # Transformation
rdf_column_2 = rdf.Define(...) # Transformation
rdf_filtered_1 = rdf_column_1.Filter(...) # Transformation
rdf_count_1 = rdf_filtered_1.Count(...) # Action
rdf_histogram_1 = rdf_column_2.Histo1D(...) # Action
```

Computational graph illustration



Without PyRDF

(*mainly for distributed backends*)

- Declare a mapper function and write all your RDataFrame analysis there
- Declare a reducer function
- Configure your distributed backend (say Spark)
- Write functions to call the mapper and reducer using your backend environment
- Display output (like draw histograms)

```
rdf = ROOT.ROOT.RDataFrame(...)

def mapper(rdf):
  rdf_column_1 = rdf.Define(...) # Transformation
  rdf_column_2 = rdf.Define(...) # Transformation
  rdf_filtered_1 = rdf_column_1.Filter(...) # Transformation
  rdf_count_1 = rdf_filtered_1.Count(...) # Action
  rdf_histogram_1 = rdf_column_2.Histo1D(...) # Action

  return rdf_count_1, rdf_histogram_1

def reducer(value1, value2):
  .....

dTree = DistTree(...)

values = dTree.ProcessAndMerge(mapper, reducer)
values[1].Draw()
```

https://github.com/etejedor/GSoC_2018/

With PyRDF

(mainly for distributed backends)

- Choose a distributed backend
- Define your RDataFrame analysis (no need of a separate function !)
- Display your output

```python
PyRDF.use('spark')

rdf = PyRDF.RDataFrame(...)

rdf_column_1 = rdf.Define(...) # Transformation
rdf_column_2 = rdf.Define(...) # Transformation
rdf_filtered_1 = rdf_column_1.Filter(...) # Transformation
rdf_count_1 = rdf_filtered_1.Count(...) # Action
rdf_histogram_1 = rdf_column_2.Histo1D(...) # Action

rdf_histogram_1.Draw()
print(rdf_count_1.GetValue())
```

https://github.com/shravan97/PyRDF/blob/master/demos/RDF_demo.ipynb

Future Improvements

- Allow the user to run a function before the mapper
- Support more distributed backends like Dask
- Integrate PyRDF in upstream ROOT

THANK YOU !

Quick links

http://shravanmurali.com/PyRDF/
https://github.com/shravan97/PyRDF/