EPJ Web of Conferences **245**, 03009 (2020) https://doi.org/10.1051/epjconf/202024503009
CHEP 2019

Distributed data analysis with ROOT RDataFrame

Vincenzo Eduardo Padulano[1,2,*], *Javier* Cervantes Villanueva[1,**], *Enrico* Guiraud[1,3,***], and *Enric* Tejedor Saavedra[1,****]

[1]CERN
[2]Università degli Studi di Milano-Bicocca (IT)
[3]University of Oldenburg (DE)

Abstract. Widespread distributed processing of big datasets has been around for more than a decade now thanks to Hadoop, but only recently higher-level abstractions have been proposed for programmers to easily operate on those datasets, e.g. Spark. ROOT has joined that trend with its RDataFrame tool for declarative analysis, which currently supports local multi-threaded parallelisation. However, RDataFrame's programming model is general enough to accommodate multiple implementations or backends: users could write their code once and execute it as-is locally or distributedly, just by selecting the corresponding backend.
This abstract introduces PyRDF, a new python library developed on top of RDataFrame to seamlessly switch from local to distributed environments with no changes in the application code. In addition, PyRDF has been integrated with a service for web-based analysis, SWAN, where users can dynamically plug in new resources, as well as write, execute, monitor and debug distributed applications via an intuitive interface.

1 Introduction

After a remarkable era with the LHC at the forefront of attempts to discover the fundamental nature of our universe, the roadmap of particle physics is full of big challenges. To extend its discovery potential, the LHC will need a major upgrade in the 2020s in order to increase the total number of collisions by a factor of ten beyond its design value. A more powerful LHC would provide more accurate measurements of new particles and enable observation of rare processes that occur below the current sensitivity level. How this should be done is at the heart of the novel machine configuration, the High Luminosity LHC (HL-LHC) [1].

The updated accelerator, scheduled to begin taking data in 2027, is planned to collect around 30 times more data than the LHC has produced so far. As this amount is close to an exabyte, it is clear that future data management will require approaches beyond simply scaling current solutions, given the future budget estimations and the expected technological evolution [2]. This means that software will have to bridge the gap between technology evolution and real computing needs, making the most out of current and future architectures.

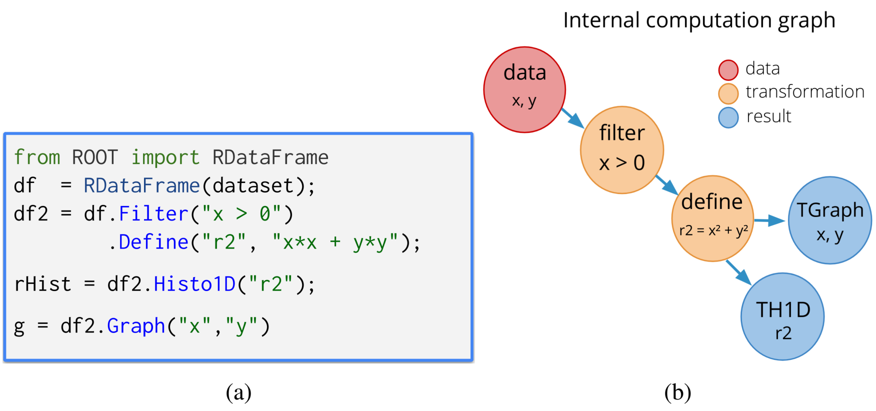
*e-mail: vincenzo.eduardo.padulano@cern.ch
**e-mail: javier.cervantes.villanueva@cern.ch
***e-mail: enrico.guiraud@cern.ch
****e-mail: enric.tejedor.saavedra@cern.ch

In this regard, the HEP community is aware of the challenges ahead and it is making continuous efforts to raise awareness that software upgrades have to happen in parallel with hardware improvements. Such advances in software must be aligned with the volume of data expected to be produced but also with the fact that software is written by many people in collaborations, with varying levels of expertise. These issues call for taking advantage of higher level programming models to increase the productivity of users while delivering results efficiently.

This paper describes recent efforts towards that goal in ROOT, the most commonly used software for High Energy Physics (HEP) analysis. Section 2 introduces RDataFrame, a high-level interface for HEP data processing that allows to deliver multi-threaded analysis on a single machine. Section 3 introduces PyRDF, a wrapper around RDataFrame that enhances its functionalities to include data processing on remote and distributed resources. Finally Section 4 gives a few demonstrations of the capabilities provided by this tool.

2 The HEP DataFrame

ROOT [3] is a scientific software toolkit offering a complete framework and environment for developing and running physics analysis, and for storing data in an efficient way. RDataFrame is one of the latest additions to ROOT, first released with version 6.14 [4].



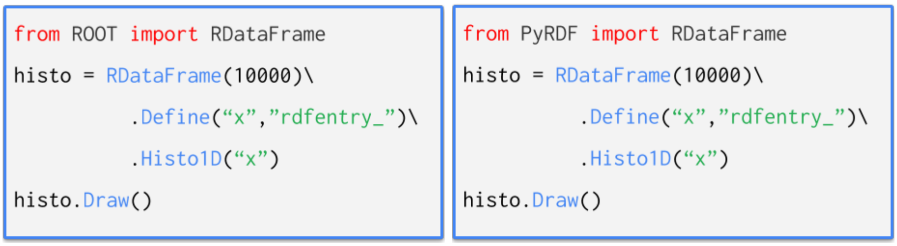
```
from ROOT import RDataFrame
df  = RDataFrame(dataset);
df2 = df.Filter("x > 0")
        .Define("r2", "x*x + y*y");

rHist = df2.Histo1D("r2");

g = df2.Graph("x","y")
```

Figure 1: Simple operations on a dataset using RDataFrame. (a): analysis code using PyROOT, ROOT's Python API. (b): the corresponding computation graph.

With RDataFrame, users can focus on their analysis as a sequence of operations to be performed on the dataset, while the framework takes care of the management of the loop over entries as well as low-level details such as IO operations and parallelisation, effectively creating a *computation graph*, that is a directed graph where each node corresponds to one of the operations to be performed on data (see Figure 1). RDataFrame provides methods to perform most common operations required by ROOT analyses, such as `Define` to create a new column in the dataset or `Histo1D` to create a histogram out of a set of values; at the same time, users can just as easily specify custom code that will be executed in the event loop via the `Foreach` method.

3 Distributed physics analyses on computing clusters

While RDataFrame may be used to make the most out of a single machine, thanks to its support for implicit multi-threading, it cannot be used as-is to run the analysis in a distributed environment. To this end, a substantial R&D effort has been put towards a tool to enable a distributed and interactive analysis tool for HEP users that had RDataFrame at its core. This goal has been achieved thanks to PyRDF, a Python wrapper around RDataFrame that allows for distributed computation on the data, without changing the programming model and the original ROOT analysis code (see Figure 2).



```
from ROOT import RDataFrame
histo = RDataFrame(10000)\
          .Define("x","rdfentry_")\
          .Histo1D("x")
histo.Draw()
```

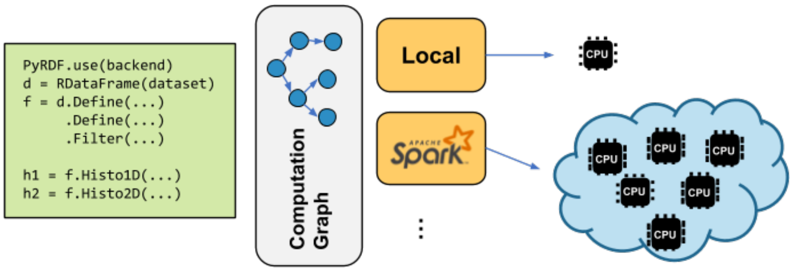
(a)

```
from PyRDF import RDataFrame
histo = RDataFrame(10000)\
          .Define("x","rdfentry_")\
          .Histo1D("x")
histo.Draw()
```

(b)

Figure 2: Code to generate a dataset with one column and plot an histogram out of it. (a): ROOT RDataFrame code. (b): PyRDF equivalent code: only the import directive changes.

The goal of the package is to expose RDataFrame to a variety of distributed backends. The current implementation fully supports Spark, a widespread framework for cluster computing [5], and it is designed to be extensible to other backends. In fact, since PyRDF is using Spark and ROOT's Python APIs, the user is given the unique possibility to write Python code that actually distributes C++ computations to the cluster (see Figure 3).



```
PyRDF.use(backend)
d = RDataFrame(dataset)
f = d.Define(...)
     .Define(...)
     .Filter(...)

h1 = f.Histo1D(...)
h2 = f.Histo2D(...)
```

Figure 3: PyRDF workflow. From left to right: The user code gives input to create a computation graph, which is then either executed locally or sent to a distributed environment, according to the user's needs.

In PyRDF, a user can execute the analysis on a local machine as shown in Figure 2 or scale it to a Spark cluster by simply adding `PyRDF.use("spark")` at the beginning of the application, without additional changes to the analysis code. RDataFrame operations have been made available for use in the distributed environment, including several that require special care when executed on multiple remote nodes rather than a single machine. Notable examples are `AsNumpy` and `Snapshot`.

On the one hand, `AsNumpy` transforms columns of the RDataFrame into `numpy.array` objects, which are used as input for widely used tools (e.g. TensorFlow). When using Spark, each node will return an array related to the range of data it is currently processing, that will be merged with all the arrays coming from the other nodes and eventually sent to the user.

On the other hand, `Snapshot` gives the user the option to save the RDataFrame on their machine as a ROOT file. Locally, the user would be able to call the function on the RDataFrame object currently processed, indicating a path where that should be saved as a ROOT file. Contextually, a copy of the RDataFrame object would be created and given to the user for additional processing. When working distributedly, each node processes a separate range of the original dataset and saves its output file to a location specified by the user. This location should correspond to a storage system that both the Spark workers and the user have access to. Finally, `Snapshot` returns to the user an RDataFrame object ready to process the concatenation of all files created during the process.

4 Selection of explored use cases

PyRDF is currently at its second functioning release. The package has been tested on SWAN [6], a web service for analysis developed at CERN that allows submitting the applications from a notebook interface to a Spark cluster while providing a homogeneous software stack thanks to the CERN software distribution service, CVMFS [7].

4.1 Scaling a TOTEM analysis

The TOTEM experiment [8] is dedicated to the precise measurement of the proton-proton interaction cross section , as well as to the in-depth study of the proton structure which is still poorly understood. In 2015 a special LHC fill (optics parameter $\beta^* = 90m$) collected 4.7 TB of data in ROOT format.

Originally these data were processed with the traditional imperative ROOT model. The original code described each step of the analysis in an event-based loop, thus repeating the same lines for every variable and then for all the output histograms (see Listing 1). The model provided by RDataFrame is declarative instead, focusing on what the program should accomplish without specifying how the program should achieve the result (see Listing 2).

```
// Suppose input data structure has been created
EventRed ev; // Declare the event data type instance
TTree outT = new TTree("distilled", "tree"); // Create output tree

for (int evi = 0; evi < ch->GetEntries() && !interrupt_loop; evi++){
  ch->GetEvent(evi);
  // Retrieve the branch and filter data, repeat for all branches
  unsigned N_L = 0;
  ev.h.L_1_F.v = rp_L_1_F->valid;
  if (ev.h.L_1_F.v) N_L++;
  outT->Fill(); // Fill the output tree
}
```

Listing 1: Original C++ code for filtering events. The user has to take full control of the event loop on each branch.

```
r = rdf.Filter(filter_code)  \ # rdf holds the input data
        .Define("v_L_1_F", track_rp_5.valid) \ # Define all needed branches
        .Snapshot(outTreeName, outFileName, outbranchlist)
```

Listing 2: Python code equivalent to that shown in Listing 1. With the API of `RDataFrame` the user can declare the operations to process the branches and the event loop will be managed implicitly.

The conversion to the RDataFrame interface is essential to ensure compatibility with Spark. There are examples in the HEP literature [9] where in order to use Spark the data have to be first converted to some other format (e.g. HDF5, Parquet). Within the new framework provided by RDataFrame and PyRDF data keep their original format (a must in the context of LHC experiments), while the programming model is converted to provide new features and be more user friendly [10, 11].

To prove the validity of this new model the two versions of the analysis, the original and the RDataFrame porting, were run and compared on the same machine. This initial comparison showed that code equivalent to the original analysis expressed with RDataFrame was on average 3 times faster.

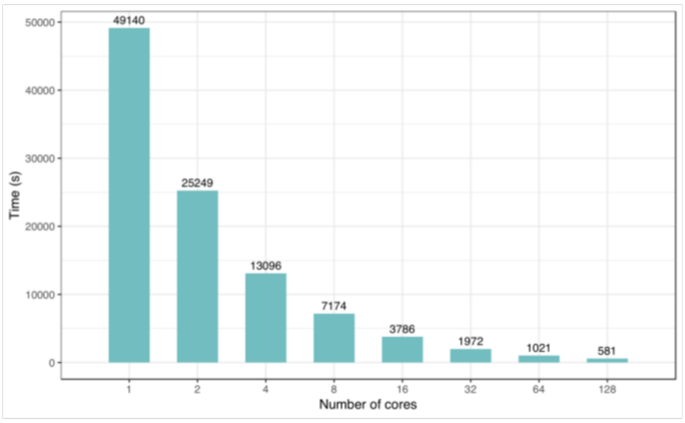


Figure 4: Execution time of the TOTEM dataset analysis with increasing number of partitions (cores)

Scaling the execution to a cluster of Spark nodes provided dramatically lower runtimes. Spark splits data into partitions, then each partition can be assigned to a node. The whole dataset was processed with the RDataFrame porting with a different number of partitions ranging from 1 to 128. Running with a single core, the analysis takes around 13 hours and 39 minutes. By contrast, creating 128 partitions and assigning each one to a different node, the execution time reaches its minimum value, taking less than 10 minutes to process the whole dataset (see Figure 4).

4.2 Dimuon spectrum analysis

This analysis takes data recorded in 2012 by the CMS experiment during LHC Run B and C and extracts the di-muon spectrum, that is the distribution of mass in the reference frame of the collision between two muons [12]. The input data is comprised of two files that collectively amount to ~8 GB and ~60 million events.

The reference code for the analysis is available as a Python script in the official ROOT tutorials (link). Since this analysis already makes use of RDataFrame, the conversion to PyRDF requires minimal modifications to the code, limited to activating the Spark backend.

A first comparison was drawn between the original RDataFrame code running with multithreading enabled on 4 threads and the PyRDF code running on a Spark cluster with 4 single-core nodes. A total of 1000 runs were executed for each program and the two time distributions were compared. In this case the analysis took on average the same time to run on the two configurations.

At a second stage the scalability of the analysis was addressed through a new set of measures with increasing core count up to 128 cores. Initial results suggested that the limited size of the dataset provided small computing needs. This is desirable for a tutorial, but useless for testing in a distributed environment. To this end, the dataset was augmented to ~208 GB by replicating the initial entries. It is worth noticing that this procedure does not change the numerical results of the analysis, since physics collision data is made of statistically independent events.

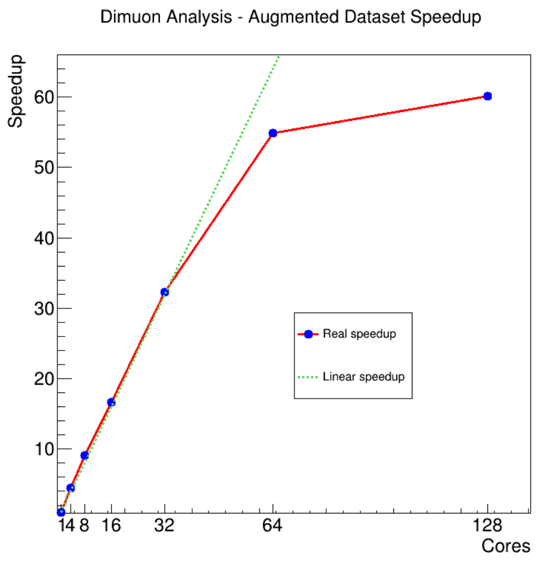


Figure 5: Speedup of the Dimuon analysis with the augmented input dataset. The green dashed line shows the ideal scaling, that is linear with the number of cores

Figure 5 shows the speedup achieved with different core counts. They collectively confirm the initial suppositions on the data size, that is software can achieve perfectly linear scaling up to a certain number of cores given the right amount of data to process. The fall of the curve at 128 cores may be due still to the limited size of data and this behaviour will be further investigated in future work.

5 Conclusions

The challenges provided by the future of High Energy Physics at CERN cannot be addressed without research in the software and computing domains. ROOT development embodies this kind of effort, also in regard to parallelisation.

RDataFrame provides an ergonomic and modern interface for data processing, implementing a declarative approach that allows to write efficient physics analyses more easily. PyRDF extends that interface, exposing it to distributed infrastructures while keeping the user code unchanged. A functioning implementation using Spark is already available to the public through CVMFS. It has been tested on several physics analyses, including real-world use cases, and provided results that demonstrate this new approach is a viable, promising alternative to standard data processing techniques.

References

[1] CERN. *The High Luminosity LHC project*. https://hilumilhc.web.cern.ch/content/hl-lhc-project. Accessed on: 02/02/2020.

[2] Elsen, E. *A Roadmap for HEP Software and Computing R&D for the 2020s*. Comput Softw Big Sci **3**, 16 (2019). https://doi.org/10.1007/s41781-019-0031-6

[3] Rene Brun and Fons Rademakers, *ROOT - An Object Oriented Data Analysis Framework*. Proceedings AIHENP'96 Workshop, Lausanne, Sep. 1996, Nucl. Inst. & Meth. in Phys. Res. A 389 (1997) 81-86. DOI: 10.5281/zenodo.3457396

[4] Danilo Piparo, Philippe Canal, Enrico Guiraud, Xavier Pla, Gerardo Ganis, Guilherme Amadio, Axel Naumann and Enric Tejedor. *RDataFrame: Easy Parallel ROOT Analysis at 100 Threads*. The European Physical Journal Conferences **214**. 10.1051/epjconf/201921406029.

[5] Matei Zaharia, Mosharaf Chowdhury, Michael J. Franklin, Scott Shenker, Ion Stoica. *Spark: Cluster Computing with Working Sets*. HotCloud 2010 (2010)

[6] Danilo Piparo, Enric Tejedor, Pere Mato, Luca Mascetti, Jakub Moscicki and Massimo Lamanna. *SWAN: A service for interactive analysis in the cloud*. Future Generation Computer Systems, 78 (2016).

[7] Jakob Blomer, Predrag Buncic, and Thomas Fuhrmann. *CernVM-FS: Delivering scientific software to globally distributed computing resources*. In Proceedings of the First International Workshop on Network-aware Data Management, pages 49–56 (2011)

[8] *The TOTEM experiment at the LHC*. http://totem-experiment.web.cern.ch/totem-experiment/

[9] Saba Sehrish, Jim Kowalkowski, and Marc Paterno. *Spark and HPC for High Energy Physics Data Analyses*. In 31st IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW) Proceedings, pages 1048–1057 (2017)

[10] Valentina Avati et al. *Declarative Big Data Analysis for High-Energy Physics: TOTEM Use Case*. In Euro-Par 2019: Parallel Processing, pages 241-255 (2019).

[11] Javier Cervantes Villanueva. *Parallelization and optimization of a High Energy Physics analysis with ROOT's RDataFrame and Spark*. Universidad de Murcia (2018)

[12] Stefan Wunsch. *Analysis of the di-muon spectrum using data from the CMS detector taken in 2012*, CERN Open Data Portal (2019).