

Introducing Live Visualization for Distributed RDataFrame with Dask

Silia Taider

CERN, Geneva, Switzerland
Summer Student 2023

Supervisors: Vincenzo Eduardo Padulano, Axel Naumann

Abstract

ROOT is a powerful data analysis framework for High Energy Physics providing the users with a wide range of tools, namely RDataFrame, a high-level interface for data analysis. This report highlights the enhancements made to RDataFrame during my Summer Student Internship. My primary focus was on introducing a live visualization feature for distributed RDataFrame computations, an extension aimed at allowing real-time data representation by visualizing intermediate results as they are computed across multiple nodes of a Dask cluster.

Keywords: ROOT; RDataframe; Dask; Python

1 Introduction

1.1 Background

1.1.1 ROOT and RDataFrame

ROOT is a widely used object oriented framework for efficient HEP (High Energy Physics) data analysis. It's a powerful tool developed at CERN for processing, analysis and data visualization that comes with a set of components customized to HEP use cases, RDataframe being one of them, and the focus of my project.

1.1.2 Distributed RDataFrame with Dask

RDataFrame is ROOT's declarative interface for HEP data analysis. It allows users to easily define a sequence of operations to be performed on their dataset, while abstracting the low-level details. RDataframe also provides an intuitive interface with distributed computing capabilities allowing users to run their code on multi-node computing clusters through a variety of distributed backends, namely Dask, an open-source Python library for parallel computing.

1.2 Motivation

In the context of a distributed workflow using RDataFrame, users define a sequence of actions and transformations to be applied to their dataset. These operations collectively form a computation graph, which is executed on each available worker.

Each worker is responsible for processing a specific portion of the entire dataset and carrying out the requested computations, yielding a partial result. These partial results are then collected and merged by the Dask scheduler, ultimately providing users with a final result, such as a histogram to be visualized on a canvas.

However, due to the inherent complexities of resource allocation and potential delays arising from various factors, workers do not always produce results simultaneously. Recognizing this challenge, we saw an opportunity to enhance the user experience by introducing the Live Visualization feature.

2 The Live Visualization feature

2.1 Workflow

The concept of this new workflow is to provide users with immediate access to partial results as they are generated.

In the traditional approach, users must wait for all workers to complete their tasks and merge all partial results before receiving a single final outcome. The suggested approach involves the continuous merging of partial results as they are produced. Simultaneously, we continuously update a canvas with the evolving intermediate state of the result, facilitating real-time visualization of the ongoing plot construction.

In essence, the Live Visualization feature offers real-time data representation of plots during the execution of a distributed RDataFrame application. Importantly, this feature does not compromise the user's access to the final result.

2.2 User Interface

The Live Visualization feature is a function designed to be called prior to initiating the computation graph.

Upon invoking the function, a secondary canvas will be opened, dedicated to displaying intermediate results. This backend canvas undergoes continuous updates as partial results become available. The `LiveVisualize()` function can be imported from the Python package `ROOT.RDF.Experimental.Distributed`:

```
1 import ROOT
2
3 LiveVisualize = ROOT.RDF.Experimental.Distributed.LiveVisualize
```

The function accepts drawable objects such as histograms, graphs, and more, along with optional callback functions to be applied on those drawables. It accommodates four different input formats:

- Passing a list or tuple of drawables:

```
1 LiveVisualize([h_gaus, h_exp, h_random])
```

- Passing a list or tuple of drawables with a global callback function:

```
1 def set_fill_color(hist):
2     hist.SetFillColor(ROOT.kBlue)
3
4 LiveVisualize([h_gaus, h_exp, h_random], set_fill_color)
```

- Passing a Dictionary of drawables and callback functions:

```
1 plot_callback_dict = {
2     graph: set_marker,
3     h_exp: fit_exp,
4     tprofile_2d: None
5 }
6
7 LiveVisualize(plot_callback_dict)
```

- Passing a Dictionary of drawables and callback functions with a global callback function:

```
1 LiveVisualize(plot_callback_dict, write_to_tfile)
```

For more comprehensive information and usage details, refer to the [RDataFrame Class Reference](#).¹

¹https://root.cern/doc/master/classROOT_1_1RDataFrame.html#distrdf

2.3 Demo

To provide a practical illustration of the Live Visualization feature in action, this script demonstrates how to integrate it into a typical data analysis workflow using RDataFrame.

In this example, we begin by setting up a Dask cluster using the `dask.distributed` library, which will enable distributed computations.

We import the necessary libraries, including ROOT, and define our `LiveVisualize` function for real-time data representation.

We initialize an RDataFrame object, by passing the Dask client's connection details to enable distributed processing.

We then create two 1D histograms, `h_1` and `h_2`, representing random and normal distributions, respectively. A ROOT TCanvas is prepared for visualization.

We then trigger live visualization by calling `LiveVisualize([h_2])`, which configures the application to continuously update and display the intermediate results of the `h_2` histogram as they are computed across the Dask cluster.

Finally, we draw the histograms, `h_1` and `h_2`, providing an interactive and real-time view of the data analysis process.

```
1 from dask.distributed import LocalCluster, Client
2 import ROOT
3
4 LiveVisualize = ROOT.RDF.Experimental.Distributed.LiveVisualize
5
6 df = RDataFrame(   treename      ,   filename   .root , daskclient=Client( tcp ://
7                   hostname:port ))
8
9 h_1 = df.Histo1D(("random", "Random distr", 50, 0, 30), "random")
10 h_2 = df.Histo1D(("gaus", "Normal distr", 50, -20, 20), "gaus")
11 c = ROOT.TCanvas("myCnvas", "MyCanvas", 1000, 600)
12 LiveVisualize([h_2])
13
14 h_1.Draw()
15 h_2.Draw()
```

Full tutorial available on the ROOT RDataFrame tutorials page.²

2.4 Implementation

The live visualization feature is implemented using Dask's `as_completed`³ method, which allows for monitoring and processing partial results as they become available during the execution of a distributed RDataFrame application.

When the user calls `LiveVisualize()`, a flag is enabled, triggering the start of the live visualization. This flag alters the map-reduce process, introducing a logic that continuously updates the live visualization canvas as partial results are computed across multiple nodes of the Dask cluster.

The logic behind the live visualization feature can be summarized as follows:

```
1 def live_visualization_logic():
2     # Enable live visualization
3     live_visualization_enabled = True
4
5     # Iterate over Dask futures representing ongoing computations
6     for future in as_completed(futures):
7
8         # Compute the partial result from a specific worker
9         partial_result = future.compute()
10
11        # Merge the partial result into the cumulative result
12        cumulative_result = reducer(cumulative_result, partial_result)
13
14        # Apply a user-defined callback function to the cumulative result
```

²https://root.cern.ch/doc/master/distrdf003_live_visualization_8py.html

³<https://docs.dask.org/en/latest/futures.html#waiting-on-futures>

```

15     callback(cumulative_result)
16
17     # Update the visualization canvas with the cumulative result
18     cumulative_result.Draw()
19
20     # Refresh the canvas to display the latest intermediate state
21     canvas.Update()

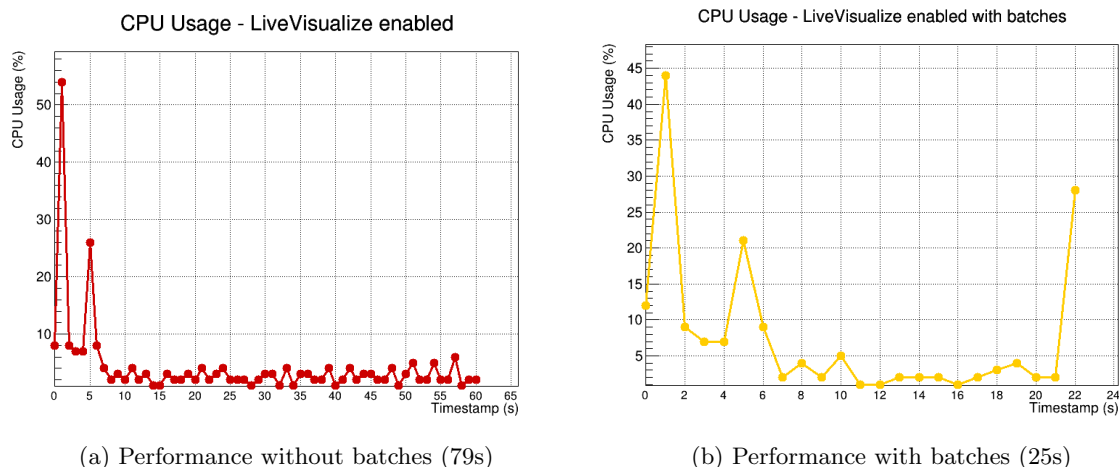
```

This implementation leverages Dask’s lazy execution logic, where computations are executed only when needed, and allows for efficient handling of distributed computations while providing an intuitive and interactive live visualization experience for the user.

2.5 Performance

To evaluate the performance of the Live Visualization feature in distributed RDataFrame, we conducted a synthetic benchmark which involved generating a dataset of one million entries in memory and creating one histogram plot. The benchmark was executed on a local cluster⁴ with four workers. The tests monitor the CPU usage over time and record the total execution time of the program. The initial benchmark revealed some notable insights into performance bottlenecks. One key observation was the need to process partial results in batches⁵, which significantly improved the total execution time.

To illustrate the impact of the live visualization on performance, we created graphical representations of CPU usage over time for specific scenarios. In this example, we compared the execution time of two benchmarks involving a 3D histogram with Live Visualization exploring the impact of enabling the `.batches()` option.



(a) Performance without batches (79s)

(b) Performance with batches (25s)

Figure 1: Comparison of execution times with and without `.batches()` option enabled.

Furthermore, We conducted additional benchmarks on different clusters, including an HPC batch cluster and an SSH cluster⁶, to assess the feature’s compatibility. In all cases, the live visualization feature was able to successfully provide real-time data representation, enhancing the interactive experience for users. Overall, the performance evaluation demonstrates the potential benefits of the live visualization feature for interactive data analysis in distributed RDataFrame, with opportunities for further optimizations and fine-tuning.

2.6 Limitations

The Live Visualization feature in distributed RDataFrame offers significant advantages for real-time data representation and interactive data analysis. However, it does come with some limitations that should be considered:

⁴<https://docs.dask.org/en/stable/deploying-python.html#localcluster>

⁵<https://docs.dask.org/en/latest/futures.html#waiting-on-futures>

⁶<https://docs.dask.org/en/stable/deploying-ssh.html#dask.distributed.SSHCluster>

- **Callback Function Restrictions:** The callback functions used with `LiveVisualize` are parsed⁷ and checked against a predefined list of blocked actions. These checks are designed to prevent unintended behaviors. Extending the list of blocked actions or providing more granular control over callback functions could enhance security and customization.
- **Jupyter Notebook Integration:** The Live Visualization feature primarily targets standard Python scripts and does not provide seamless integration with Jupyter Notebooks or other interactive environments. Future enhancements could explore better integration options for notebook users.
- **Real-World Use Cases:** While the feature has been demonstrated with synthetic benchmarks, more extensive testing on a variety of real-world use cases is needed. This can help identify potential issues and areas for improvement in real-world scenarios.
- **Limited to Drawables:** Currently, the Live Visualization feature is designed to work primarily with drawables, such as histograms and graphs. Extending support to other types of objects and data representations could broaden its applicability.
- **Single RDataFrame:** The feature is optimized to work with objects from the same `RDataFrame` instance. Support for visualizing results from multiple `RDataFrame` instances in a single canvas could be explored.

3 Conclusion

Throughout this summer project, I had the opportunity of developing a compelling feature for Distributed `RDataFrame`, aimed at enhancing interactive data analysis. Moreover, I seized the opportunity to delve into various ROOT-related GitHub issues, offering me a valuable glimpse into the intricacies of the ROOT framework, both in Python and C++.

4 Acknowledgements

I would like to express my heartfelt gratitude to my supervisor, Vincenzo, for his exceptional guidance, patience, and invaluable insights throughout this project. His constant support allowed me to push this project beyond what I had initially planned, all while having an incredible experience at CERN overall.

I would also like to extend my sincere appreciation to Axel and the entire ROOT team for their continuous feedback, and the incredible opportunity to contribute to this remarkable open-source project.

Thank you all for making this project an enriching summer on every level!

⁷<https://docs.python.org/3/library/ast.html>