



30 July 2021
anisoara.ionela@gmail.com

CERN Summer Student Programme Report

Anisoara-Ionela Plesca
CERN, CH-1211 Geneva, Switzerland

Keywords: Resistive Plate Chambers, data acquisition, data visualisation

Summary

This document describes the activities that have been carried on during my 2021 Summer Student project. I was selected to work in the Experimental Physics - Detector Technologies - Fluidic Systems (EP-DT-FS) Gas Systems section. The project consists of implementing a data acquisition and control system for a particle detector setup installed at the Gamma Irradiation Facility (GIF++).

Contents

1	Introduction	2
2	The Project	2
2.1	Monitor and control of a High Voltage power supply	2
2.2	Data loading and visualization	3
2.3	Single run Analysis	3
3	Conclusion	4

1 Introduction

The detectors that need to be managed with the created system are called Resistive Plate Chambers (RPCs), a type of gas detectors employed at the CERN LHC experiments (ALICE, ATLAS and CMS).

Resistive plate chambers (RPC) are fast gaseous detectors used in the muon trigger system of LHC experiments. They consist of two parallel plates, a positively-charged anode and a negatively-charged cathode, both made of a high resistivity material and separated by a gas volume.

When a muon passes through the chamber, electrons are knocked out of gas atoms. These electrons in turn hit other atoms, causing an avalanche of electrons. The electrodes are transparent to the signal (the electrons). The signal is picked up by external metallic strips after a small but precise time delay. The pattern of hit strips gives a quick measure of the muon momentum, which is then used by the trigger to make immediate decisions about whether the data are worth keeping. [2]

The current gas mixture used in the detectors is made of about 95% of C₂H₂F₄ (also known as R-134a) with a Global Warming Potential (GWP) of 1430. This means the gas contributes heavily to the planet's greenhouse effect.

CERN Gas Group has adopted several strategies to reduce greenhouse gases emission from particle detector at LHC. One research branch consists of monitor and control the performance of RPCs operated with environmentally friendly gas mixtures. The experiments are run in laboratory environment and at the Gamma Irradiation Facility (GIF++).

The project developed during my Summer Programme will be used to acquire, analyse, visualize and compare data taken with the mentioned setups.

2 The Project

The project consists of a web app written in python with the framework [Dash by Plotly](#) and hosted on the CERN OpenShift platform.

2.1 Monitor and control of a High Voltage power supply

The first part of the project consisted in monitoring and controlling the detector's voltage and current. The detector voltage is applied using a CAEN High Voltage module that provides API to read and write parameters for each channel in the power supply.

I started working on a single page of the application, and it consisted of displaying a table that continuously updates with new data from a websocket. At first I tried connecting to the websocket in a different thread and updating the table with an [Interval component](#). This proved to be inflexible when trying to switch different websocket sources. The next attempt was made using a [Websocket component](#) from the `dash-extensions` package.

Besides the continuously updating table, the user needed to be able to control the channels parameters. At first the idea was to make the table editable, but that would interfere with the updating data. So a form was made for that purpose. In Figure 1 a screenshot of the resulting page is reported.

HV control test

hvrpc256 (LAB) × ▾

slot	channel	V0Set	I0Set	VMon	IMon	Pw	Status	Correction(V0Set)
0	0	6000	5	0	0	0	Off	false
0	1	9600	20	0	0	0	Off	false
0	2	0	1	0	0	0	Off	false
0	3	9723	0	0	0	0	Off	false
0	4	8600	1	0	0	0	Off	false
0	5	8600	10	0	0	0	Off	false
Slot 2 × ▾								
Channel 0 × ▾								
I0Set × ▾								
input type number ▾								
Submit								

Figure 1: Updating table with input form

2.2 Data loading and visualization

For the second part of the work, I had to fetch the data of different experiment runs and display plots for visualizing and comparing the results. The data was stored in a MongoDB instance. A CSV file together with a set of functions to connect to the database were provided as a starting point. I needed to modify the existing functions in order to make them work with each other. I also had to fix plot legends and titles as seen in Figure 2 and add features for saving run notes and downloading selected rows as CSV.

2.3 Single run Analysis

The third and last section I worked on, is a page for the analysis of a single run. The analysis uses the [Olefin](#) library written specially for the existing setup. For running it, the user is given a template analysis configuration like in Fig 3.

The main sections in the config are: [1]

- acquisition: for acquisition related configs such as record length, strip channels map, etc.
- digitizer: mainly parameters identifying the specs of the digitizer used
- rpc: related mostly to the construction of the RPC itself
- signal: related to the signal analysis class. They are mostly parameters for low level algorithms
- run: global configs related to the run

Efficiency and streamer probability

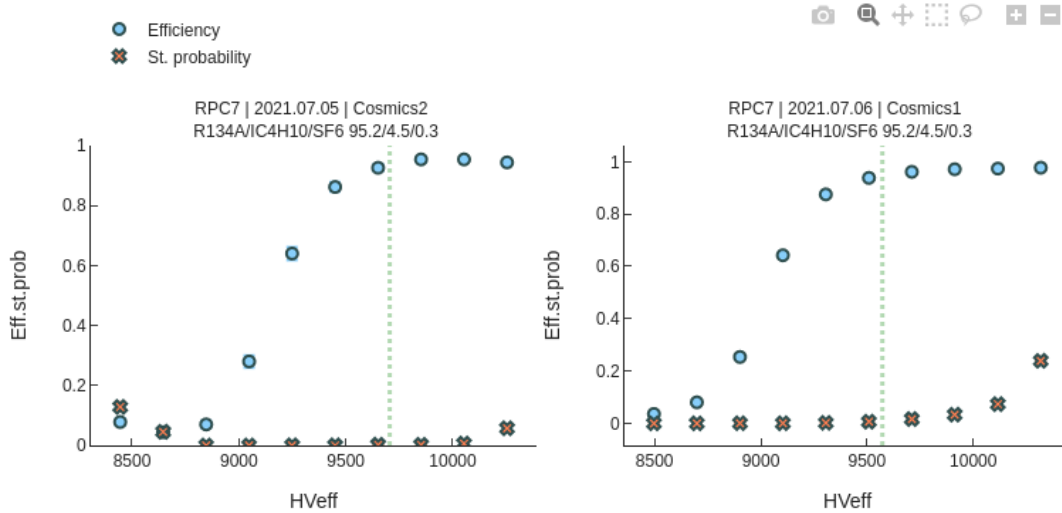


Figure 2: Plots with updated legend and titles

The configuration is displayed in a YAML format in order to be easier for a person to understand its structure. The user can then can modify it and try different variations using an [Ace Editor component](#) and validated using a [Pydantic](#) model in order to make sure there is no confusion regarding data types or structure. The validated data is run through an Olefin method that analyses the data in the folder from the path given by the user.

The method is sent to a job queue on a [redis](#) instance, and then run on a different worker pod using the [Python Redis Queue](#) library. After the method is run, the result is temporarily stored on the redis server in order to be accessed by our app. Figures 4 and 5 display the resource usage for one job run for all three instances.

The result of the Olefin analysis is an object that allows interaction with the analysed data. One property of the resulting object is a [Pandas](#) dataframe that is then displayed using the [DataTable](#) component as seen in Fig. 6.

3 Conclusion

I could say that my CERN experience was splendid from the very beginning! It started with a warm welcome from my supervisors, Gianluca Rigoletti and Beatrice Mandelli, and from the program coordinators Ana and Anastasiia.

It continued with a very interesting project where I found out more about

- the experiments at CERN
- how a muon detector works
- some interesting software tools, for example OpenShift and the redis job queue

Unfortunately, my time here ended, but I had a fantastic time here and hopefully it won't be the last!

References

- [1] Olefin analysis guide. <https://gitlab.cern.ch/grigolet/olefin/-/blob/master/notebooks/1.%20Analysis%20guide.ipynb>, 2021.
- [2] Resistive plate chambers. <https://cms.cern/detector/detecting-muons/resistive-plate-chambers>, 2021.

Single Run Analysis

```

1 acquisition:
2   header: 520
3   header_lines: null
4   record_length: 520
5   strip_map: {}
6   trigger: wave0
7 digitizer:
8   adc_to_mv: 0.1220703125
9   bit_resolution: 14
10  model: 1730
11  sample_frequency: 500
12  time_resolution: 2.0
13  vpp: 2
14  notes: ''
15 rate:
16   area: 240.0
17   distance: 30
18   folder: Rate
19   height: 2
20   polyorder: 5
21   record_length: 622340
22   window_length: 21
23 rpc:
24   resistance: 56
25 run:
26   currents_path: currents.xlsx
27   efficiency_parameters_limits: !!python/tuple
28   - !!python/tuple
29     - 0
30     - 0.0001
31     - 6000
32   - !!python/tuple
33     - 1
34     - 0.2
35     - 12000
36 fit_params:
37   - 0.9
38   - 0.01
39   - 9800
40 gas_mixture: {}
41 initial_params:
42   - 0.9
43   - 0.01
44   - 9800
45 mix_folder: MIX0
46 p: 965
47 p0: 965
48 parameter_map: null
49 t: 20
50 t0: 20
51 working_point_delta: 200
52 signal:
53   axis: 1
54 baseline_interval:
55   - 0
56   - 200
57 baseline_threshold: 5
58 charge_factor: 0.004359654017857143
59 charge_region:
60   - 5
61   - 30
62 charge_thresh: 9
63 charge_thresh_av: 16
64 charge_threshold: 0.15
65 height_thresh: 2
66 height_thresh_av: 12
67 heights_difference_threshold: 1

```

Validate YAML

Figure 3: Olefin Configuration Template

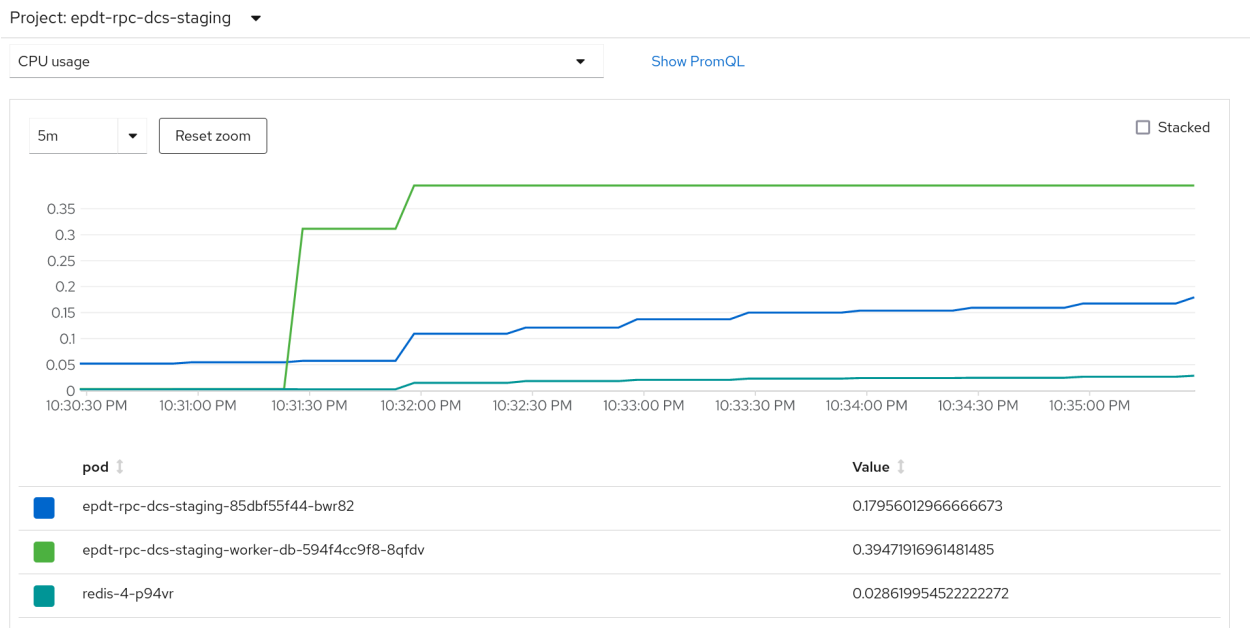


Figure 4: CPU Usage on job run

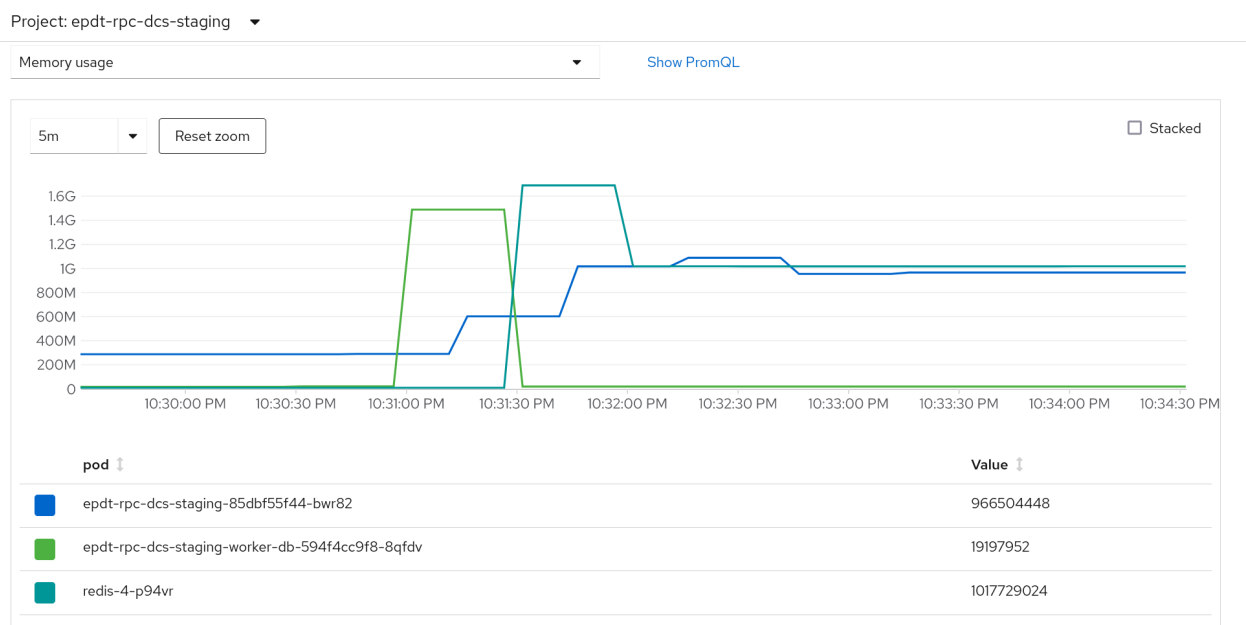


Figure 5: Memory Usage on job run

