MAPD-B distributed processing exam default projects

Latest revision: **08-06-2023**

Contents

Changelog

08-06-2023
1. Minor review and revision of the QUAX data processing project

28-05-2023
1. Included note on how to retrieve a slimmer version of the Kaggle dataset for the Covid-19 project
2. Included a **first (almost preliminary) version** of the QUAX stream data processing project

30-05-2022
1. Modified the pointer to the S3 buckets of the cloud storage files for both Batch and Streaming projects of Drift Tube detectors
2. Included Appendix on the S3 connection using plain python/Dask/Spark

1 Distributed Algorithms

Complexity degree: ■■□

1.1 Introduction

A number of common algorithms used in Data Science and Machine Learning are not by default designed to take advantage of the parallelization offered by distributed systems. Several alternative approaches have thus stemmed to modify the existing algorithms to embrace MapReduce-like architectures.
Nowadays, Spark and Dask do offer a variety of distributed algorithms pre-implemented (either in the Spark MLLib API, or in the large Python ecosystem used by Dask).
In this project, you will be asked to implement and benchmark alternatives of common algorithms in either Spark or Dask, without using the related already provided functions. The project is thus focused on the efficient implementation of algorithms in a distributed system, rather than on the processing of experimental/industrial datasets.

1.2 Task

Following the description of the documentation provided, you will have to implement either one of the listed algorithms, in the distributed framework of your choice.
The performance of the algorithm will have to be tested on standard synthetic datasets, commonly available on

1.2.1 Clustering - k-Means‖

One of the most used techniques for unsupervised clustering is the k-Means method.
One largely used method for weight initialization in k-Means is the so-called k-Means++. k-Means++ is however a largely sequential algorithm.
A scalable version of its implementation is available and often referred to as k-Means‖ (parallel k-Means), and has been described in reference https://arxiv.org/abs/1203.6402.
At the core of the k-Means‖ lies the initialization procedure, which can be represented with the following pseudo-code, as presented in the paper referred above.
To test and validate the performance of the distributed algorithm, use either one of the following real-world standard datasets made available by sci-kit learn.

- RCV1 dataset (https://scikit-learn.org/stable/datasets/real_world.html#rcv1-dataset)
- kddcup99 dataset (https://scikit-learn.org/stable/datasets/real_world.html#kddcup-99-dataset)

1.2.2 Clustering - Mini-batch k-Means

An alternative approach to k-Means++ and k-Means‖ is the so-called Mini-batch k-Means https://www.eecs.tufts.edu/ dsculley/papers/fastkmeans.pdf.
The algorithm uses small (mini) batches to optimize k-means clustering instead of using a single large-batch optimization procedure.
This is demonstrated to provide faster convergence and can be implemented to scale k-Means with low computation cost on large datasets.

To test and validate the performance of the distributed algorithm, use either one of the following real-world standard datasets made available by sci-kit learn.

- RCV1 dataset (https://scikit-learn.org/stable/datasets/real_world.html#rcv1-dataset)
- kddcup99 dataset (https://scikit-learn.org/stable/datasets/real_world.html#kddcup-99-dataset)

1.2.3 Dimensionality reduction

Dimensionality reduction can be achieved via matrix decomposition using for instance the SVD methods. An alternative approach is to use a so-called QR decomposition or QR factorization, where the matrix A is decomposed into a product of an orthogonal matrix Q and an upper triangular matrix R. In the case of very large datasets, where the number of features is much less than the number of rows (i.e. where the number of examples is extremely large), the matrix tent to be referred to as "tall-and-skinny", and efficient ways to perform QR decomposition can be defined.
Implement the Direct TSQR method proposed in paper https://arxiv.org/abs/1301.1071 and test its performance on one of the real-world standard datasets made available by scikitlearn, preferably:

- California housing dataset (https://scikit-learn.org/stable/datasets/real_world.html#california-housing-dataset)

1.3 Hints

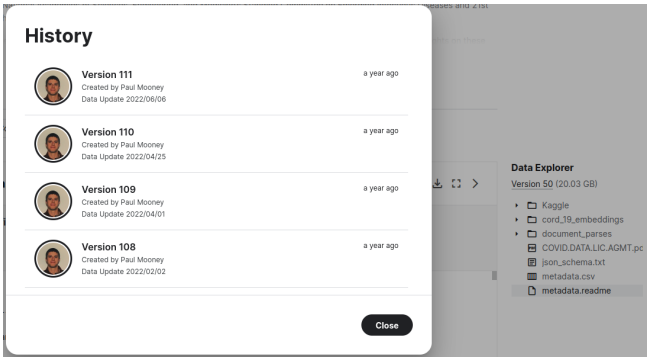
- Start prototyping the code in a local environment, possibly using a local cluster deployment of Spark or Dask.
- The test datasets are typically very large, and thus are very computationally expensive to run on them without any pruning or filtering stage. Do reduce your dataset to a more manageable size, if needed, due to the limited VM resources.
- If you need help or have questions contact us via email at the addresses: matteo.migliorini@pd.infn.it, jacopo.pazzini@unipd.it

2 Analysis of Covid-19 papers

Complexity degree: ■□□

2.1 Introduction

This distributed computing project will be focused on the analysis of a number of papers about COVID-19, SARS-CoV-2, and related coronaviruses. The dataset is a sub-sample taken from the original dataset that is composed of 75000 (and still increasing) papers. This dataset is a part of real-world research on COVID-19 named COVID-19 Open Research Dataset Challenge (CORD-19). The research and related challenges are available on the dedicated page on Kaggle: https://www.kaggle.com/allen-institute-for-ai/CORD-19-research-challenge
You can choose the amount of data to analyze by selecting whichever version of the dataset you prefer. At the moment of writing, the total dataset size is in the excess of 80 GB, but "going back in time" with the Kaggle `data explorer` functionality, you can select and use an older dataset version (e.g. v50), as in the following screenshot:



2.2 Structure of the Data

All the documents present in the *data* folder are structured in this way:

```
{
    "paper_id": <str>,      # 40-character sha1 of the PDF
    "metadata": {
        "title": <str>,
        "authors": [        # list of author dicts, in order
            {
                "first": <str>,
                "middle": <list of str>,
                "last": <str>,
                "suffix": <str>,
                "affiliation": <dict>,
                "email": <str>
            },
            ...
        ],
        "abstract": [          # list of paragraphs in the abstract
```

```
            {
                "text": <str>,
                "cite_spans": [ # list of character indices of inline
                   citations
                                # e.g. citation "[7]" occurs at positions
                                   151-154 in "text"
                                #linked to bibliography entry BIBREF3
                    {
                        "start": 151,
                        "end": 154,
                        "text": "[7]",
                        "ref_id": "BIBREF3"
                    },
                    ...
                ],
                "ref_spans": <list of dicts similar to cite_spans>,
                # e.g. inline reference to "Table 1"
                "section": "Abstract"
            },
            ...
        ],
        "body_text": [      # list of paragraphs in full body
                            # paragraph dicts look the same as above
            {
                "text": <str>,
                "cite_spans": [],
                "ref_spans": [],
                "eq_spans": [],
                "section": "Introduction"
            },
            ...
            {
                ...,
                "section": "Conclusion"
            }
        ],
        "bib_entries": {
            "BIBREF0": {
                "ref_id": <str>,
                "title": <str>,
                "authors": <list of dict>  # same structure as earlier,
                                           # but without `affiliation` or
                                              `email`
                "year": <int>,
                "venue": <str>,
                "volume": <str>,
                "issn": <str>,
                "pages": <str>,
```

```
                "other_ids": {
                    "DOI": [
                        <str>
                    ]
                }
            },
            "BIBREF1": {},
            ...
            "BIBREF25": {}
        },
        "ref_entries":
            "FIGREF0": {
                "text": <str>,         # figure caption text
                "type": "figure"
            },
            ...
            "TABREF13": {
                "text": <str>,        # table caption text
                "type": "table"
            }
        },
        "back_matter": <list of dict>  # same structure as body_text
    }
}
```

Data may contain empty fields and malformed values, so pay attention.

2.3 Assignment

2.3.1 Word counter distributed algorithm

First of all, you have to implement the following distributed algorithm to count the occurrences of all the words inside a list of documents. In NLP (Natural Language Processing) a document is a body of text, in this case, each paper is a document. The algorithm is defined as follows:

Map phase : For each document Di, produce the set of intermediate pairs $(w, cp(w))$, one for each word $w \in Di$, where $cp(w)$ is the number of occurrences of w in Di. E.g.: $('hello', 3)$

Reduce phase : For each word w, gather all the previous pairs $(w, cp(w))$ and return the final pair $(w, c(w))$ where $c(w)$ is the number of occurrences of w for all the Documents. In other words, $c(w)$ is equal to $\sum_{k=1}^{n} cp_k(w)$

The algorithm has to be run on the full text of the papers. To get the full text of the paper you have to transform the input data by concatenating the strings contained into the *body-text* fields of the JSONs.
To perform this transformation we strongly suggest you use the RDD/Bag data structure. Anyway if you prefer to implement the algorithm by using the DataFrame structure feel free to do it.

At the end of the algorithm analyze the top words and see how they are related to the viruses and the research (for example create a barplot of the top words)

2.3.2 Which are the worst and best-represented countries in the research?

In this part, you have to take the documents and convert them into a usable DataFrame data structure to figure out the countries that are most and less active in the research. To do this you can use the country of the authors. Do the same for the universities (affiliations).
Even in this case do multiple runs by changing the number of partitions and workers and then describe the behavior of the timings.

2.3.3 Get the embedding for the title of the papers

In NLP a common technique for performing analysis over a set of texts is to transform the text into a set of vectors each one representing a word inside a document. At the end of the pre-processing the document will be transformed into a list of vectors or a matrix of $n \times m$ where n is the number of words in the document and m is the size of the vector that represents the word n.
More information about word-embedding: https://towardsdatascience.com/introduction-to-word-embedding-and-word2vec-652d0c2060fa
What you can do is transform the title of the paper into its embedding version by using the pre-trained model available on the FastText page: https://fasttext.cc/docs/en/pretrained-vectors.html.
The pre-trained model that you have to download is the https://dl.fbaipublicfiles.com/fasttext/vectors-wiki/wiki.en.vec
Basically, the pre-trained model is more or less a huge dictionary in the following format $key : vector$.
To load the model follow this snippet of code which is slightly different from what you can find on this page https://fasttext.cc/docs/en/english-vectors.html

```
import io

def load_vectors(fname):
    fin = io.open(fname, 'r', encoding='utf-8', newline='\n', errors='ignore')
    n, d = map(int, fin.readline().split())
    data = {}
    for line in fin:
        tokens = line.rstrip().split(' ')
        data[tokens[0]] = list(map(float, tokens[1:]))
    return data

model = load_vectors('wiki.en.vec')

#to get the embedding of word 'hello':

model['hello']
```

Once you have loaded the model, use the map approach to create a DataFrame or a Bag/RDD that is composed by:

- paper-id

- title-embedding

The title embedding can be a list of vectors or can be flattened to a large vector.

2.3.4 Cosine Similarity

Use the previously generated vectors to compute the cosine similarity between each paper and to figure out a couple of papers with the highest cosine similarity score.

2.4 Hints

- The text of the documents must be sanitized before performing the analysis (remove punctuation, stop-words).
- Not all words are contained in the pre-trained model. In this case, skip the word and go to one.
- Remember that if you use the distributed "read file" the files must be in the same path for each worker
- If you need help or have questions contact us via email at the addresses: jacopo.pazzini@unipd.it

3 Anomaly detection and Predictive maintenance for industrial devices

Complexity degree: ■■■

3.1 Introduction

Nowadays, IoT and Big Data allowed retrieving significant benefits from the devices' data sensors. Typically in the academic world, this means, the possibility to perform some analysis as time series forecasting, data predictions, and data classification.
Despite this being extremely useful from the research side, from the enterprise point of view, the necessities are instead to exploit the huge amounts of data generated by devices, to perform some predictions for Anomaly detection (AD) and Predictive Maintenance (PM):

- Anomaly detection (also outlier detection) is the identification of rare items, events, or observations that raise suspicions by differing significantly from the majority of the data (Wikipedia Anomaly Detection).
- Predictive maintenance techniques are designed to help determine the condition of in-service equipment to estimate when maintenance should be performed. (Wikipedia Predictive Maintenance)

In the light of the above, it's clear, that both AD and PM are extremely important for a company to identify possible problems on their devices, anomalies during a production process, perform maintenance before a fault of a device, or reduce the cost of recurrent maintenance (that is done even if it is not necessary).

The following project is based on real data taken from sensors of a big company that operates in the refrigeration field.

3.2 Data Structure

Table 2 reports a small example of the dataset that you need to use to perform your analysis. The dataset has been extracted from a larger dataset and it contains the data of 4 different devices. The columns of the dataset are:

- **when**: is the UNIX timestamp of the data (the measurement time of the data)
- **hwid**: indicated the device of the measured metric
- **metric**: indicate the name of the measured metric
- **value**: indicate the value of the measured metric

All the data have a sampling frequency between the 30s to 1min that has to be normalized to the same frequency before to performs an analysis. The data cover the period from 1 Oct 2020 to 31 March 2021. If your cluster resources are not enough you can split use the data for just three months (1 Jan - 31 Mar).

The data are available to this **link** in a zipped csv format. The unzipped version of the dataset is around $\approx$ 5GB.

In Table 3 are reported the name, the physical meaning, the type of the measurement unit, and the number of decimals for each variable contained.

In addition, for the variables $A5$ and $A9$ (that represent Alarms on different circuits of a device) are a 16bit integer that *must be converted* into a 16bit string before being used. This is a typical industrial approach for edge computing.

Let's suppose that you have 16 different sensors on the same device, and each one is monitoring a particular piece of the device.
Each of these sensors, every N seconds, may send 0 if all it is ok or 1 if their monitored piece of the device is faulty.
This is not suitable for edge devices that have to be optimized much as possible. An idea is to combine the output of all those sensors into a 16-bit string, where the index of each bit represents the output of the same index sensor.

Let's imagine that at a certain time T your 16 sensors send the data on Table 1. This output can be aggregated into a single string: 0100000000000101 and then send to the final data storage with the corresponding integer: 8197.

Table 1: Alarm sensors outputs at certain time T

Sensor	**Output**
Sensor 1	1
Sensor 2	0
Sensor 3	1
Sensor 4	0
Sensor 5	0
Sensor 6	0
Sensor 7	0
Sensor 8	0
Sensor 9	0
Sensor 10	0
Sensor 11	0
Sensor 12	0
Sensor 13	0
Sensor 14	0
Sensor 15	0
Sensor 16	1

Table 2: Example of data taken from the dataset.

When	Hwid	Metric	Value
1601510485159	SW-065	SA4	0
1601510485159	SW-065	SA3	0
1601510485159	SW-065	SA2	0
1601510485159	SW-065	S34	0
1601510485159	SW-065	S33	1
1601510485159	SW-065	S25	0
1601510485159	SW-065	S19	0
1601510485159	SW-065	S17	0
1601510485159	SW-065	S16	0
1601510485159	SW-065	S15	0
1601510532889	SW-115	S41	153
1601510535401	SW-115	S143	100
1601510563238	SW-115	S41	152
1601510565967	SW-115	S143	100
1601510593606	SW-115	S41	153
1601510596120	SW-115	S143	100
1601510624047	SW-115	S41	152
1601510626780	SW-115	S143	100
1601510654443	SW-115	S41	152
1601510657046	SW-115	S143	100

Table 3: Variable description and overview

Name	Description	UDM	Decimals
A5	Circuit 1 Alarm 1		0.0
A9	Circuit 2 Alarm 1		0.0
ComError			
E1	Active power		1.0
E2	Reactive power		1.0
E3	Active energy (LOW)		0.0
E4	Active energy (HIGH)		0.0
E5	Reactive energy (LOW)		
E6	Reactive energy (HIGH)		
P1	Status Control		0.0
P10	Recovery Set-Point	°C	1.0
P15	Alarm Reset		0.0
P16	Set-Point Cooling	°C	1.0
P17	Set-Point Heating	°C	1.0
P18	Set-point ACS	°C	1.0
P2	Mode Control		0.0
P5	Unit Set-Point	°C	1.0
P6	Set-Point ECO Cooling	°C	1.0
P7	Set-Point ECO Heating	°C	1.0

Table 3: Variable description and overview

Name	Description	UDM	Decimals
P8	2° Set-Point Cooling	°C	1.0
P9	2° Set-Point Heating	°C	1.0
S1	Unit Mode		0.0
S10	Utility Request	%	0.0
S100	Suction Pressure Cooling Mode Circ 1	bar	1.0
S101	Suction Pressure Heating Mode Circ 1	bar	1.0
S102	Discharge Pressure Circ 1	bar	1.0
S106	Source inlet temp. circ 1	°C	1.0
S107	Liquid Temperature Circ 1	°C	1.0
S108	Source Temperature OUT Circ 1	°C	1.0
S109	Discharge Temperature C1,1	°C	1.0
S11	ACS Request		0.0
S110	Discharge Temperature C2,1	°C	1.0
S112	4 Way Valve Circ 1		0.0
S113	Chiller Valve Circ 1		0.0
S114	Recovery Valve Circ 1		0.0
S115	Empty Coils Valve Circ 1		0.0
S117	Status Comp 1,1		0.0
S118	Status Comp 2,1		0.0
S122	Signal Inverter Fan Circ 1	V	2.0
S123	Circ 1 mode		0.0
S124	Subcooling Circ 1	°C	1.0
S125	Capacity Circ 1	%	0.0
S126	Pressure Ratio Circ 1		1.0
S127	Defrost Mode Circ1		0.0
S128	Set-Point SH PID1 Circ2	°C	1.0
S129	Set-Point SH PID2 Circ2	°C	1.0
S130	Compressor ON C2		0.0
S137	Set-Point SH PID3 Circ 2	°C	1.0
S138	SH PID1 Circ 2		1.0
S140	SH PID2 Circ 2		1.0
S143	EEV Position PID2 Circ 2	%	0.0
S147	EEV Position PID3 Circ 2	%	0.0
S15	Source Pump 1		0.0
S151	EEV Position PID1 Circ 2	%	0.0
S154	Superheat circ 2 (PID3)		1.0
S157	Suction Pressure Cooling Mode Circ 2	bar	1.0
S158	Suction Pressure Heating Mode Circ 2	bar	1.0
S159	Discharge Pressure Circ 2	bar	1.0
S16	Source Pump 2		0.0
S163	Source Temperature IN Circ 2	°C	1.0
S164	Liquid Temperature Circ 2	°C	1.0
S165	Source Temperature OUT Circ 2	°C	1.0
S166	Discharge Temperature C1,2	°C	1.0
S167	Discharge Temperature C2,2	°C	1.0

Table 3: Variable description and overview

Name	Description	UDM	Decimals
S169	Status Comp 1,2		0.0
S17	Source Pump 3		0.0
S170	Status Comp 2,2		0.0
S171	4 Way Valve Circ 2		0.0
S172	Chiller Valve Circ 2		0.0
S173	Recovery Valve Circ 2		0.0
S174	Empty Coils Valve Circ 2		0.0
S175			0.0
S176			0.0
S178	Signal Inverter Fan Circ 2	V	2.0
S179	Circ 2 mode		0.0
S180	Subcooling Circ 2	°C	1.0
S181	Capacity Circ 2	%	0.0
S183	Defrost Mode Circ2		0.0
S19	Source Inverter Signal	%	1.0
S2	Unit Status		0.0
S201	Status Driver Module 1 Circ 1		0.0
S202	Status Driver Module 2 Circ 1		0.0
S203	Status Driver Module 3 Circ 1		0.0
S204	Status Driver Module 1 Circ 2		0.0
S205	Status Driver Module 2 Circ 2		0.0
S206	Status Driver Module 3 Circ 2		0.0
S25	Anti-Freeze Heater		0.0
S3	Set-Point Unit	°C	1.0
S33	Utility Pump 1		0.0
S34	Utility Pump 2		0.0
S35	Utility Pump 3		0.0
S37	Utility Inverter Signal	%	1.0
S39	Utility Water IN	°C	1.0
S40	Utility Water OUT	°C	1.0
S41	External temperature	°C	1.0
S42	Water Reset	V	1.0
S43	Demand Limit	V	1.0
S45	Free-Cooling Water Temp.	°C	1.0
S46	Recovery Water inlet	°C	1.0
S47	Recovery Water OUT	°C	1.0
S49	Source Water IN	°C	1.0
S5	Corrent Power Steps		0.0
S50	Source Water OUT	°C	1.0
S53	Comulative Alarm		0.0
S54	Indirect Free-cooling Pump 1		0.0
S55	Recovery Pump 1		0.0
S56	Indirect Free-cooling Pump 2		0.0
S57	Recovery Pump 2		0.0
S6	Recovery Set-Point	°C	1.0

Table 3: Variable description and overview

Name	Description	UDM	Decimals
S63	DHW valve		0.0
S64	Free-Cooling Valveù		0.0
S69	Free-Cooling Valve Signal	V	1.0
S7	Blocked Alarm		0.0
S70	Recovery Pump Signal	V	1.0
S71	Set-Point SH PID1 Circ1	°C	1.0
S72	Set-Point SH PID2 Circ1	°C	1.0
S73	Compressor ON C1		0.0
S8	Warning Alarm		0.0
S80	Set-Point SH PID3 Circ 1	°C	1.0
S81	SH PID1 Circ 1		1.0
S83	SH PID2 Circ 1		1.0
S86	EEV Position PID2 Circ 1	%	0.0
S9	Recovery Request	%	0.0
S90	EEV Position PID3 Circ 1	%	0.0
S94	EEV Position PID1 Circ 1	%	0.0
S97	SH PID3 Circ 1		1.0
SA1	Alarm 1 (status)		0.0
SA10	Circuit 2 Alarm 2 (status)		0.0
SA11	Circuit 2 Alarm 3 (status)		0.0
SA12	Circuit 2 Alarm 4 (status)		0.0
SA2	Alarm 2 (status)		0.0
SA3	Alarm 3 (status)		0.0
SA4	Alarm 4 (status)		0.0
SA5	Circuit 1 Alarm 1 (status)		0.0
SA6	Circuit 1 Alarm 2 (status)		0.0
SA7	Circuit 1 Alarm 3 (status)		0.0
SA8	Circuit 1 Alarm 4 (status)		0.0
SA9	Circuit 2 Alarm 1 (status)		0.0

3.3 Assignment

3.3.1 Anomaly Detection

Each one of the devices in the dataset has 4 different engine which has to compress a gas to chill or heat the environment. Continually turning ON and OFF these engines, with high frequency, is not recommended and could indicate problems that happened during the installation phase, deterioration of some mechanical parts, or not suitable external environment conditions. For this point there are two tasks:

- Perform some distributed analysis that allows the users to identify possible anomalies on the high number of turn on and off these engines (an anomaly happens even if just one of the engines of a device is turning on/off with high frequency)
- Identify some possible correlations between the higher frequency of turning on/off of the engines, and the other variables. There are some patterns?

The metrics/variable that containes the information for the turning on/off the engines are:

- S117 (engine 1)
- S118 (engine 2)
- S169 (engine 3)
- S170 (engine 4)

3.3.2 Anomaly Detection 2

As before, the devices in the dataset have 4 different engines which have to compress a gas to chill or heat the environment. The percentage of working/loading of those units is extremely important due to it could indicate problems related to under-dimensioned or over-dimensioned devices, deterioration of some mechanical parts, or not suitable external environment conditions. For this point, the task is:

- Identify some possible correlations between the percentage of device loading and the external temperature

The metrics/variables that contain the information about the loading percentage of the devices are:

- S125 (ergated capacity by circuit 1)
- S1181 (ergated capacity by circuit 2)

A device can have one or both variables.

3.3.3 Predictive Maintenace

As written in the previous section, variables $A5$ and $A9$ contain alarms about the devices. More specifically, if one (or all) of the bit in positions 6, 7, and 8, of one/both the alarm/s, are equal to 1 this means that there is/are one/some faulty/ies. In fact, if at least one of the cited bits is at 1, it means the engines are overheating.

For this point the task is:

- Convert the alarms from their integer encoding to their bit string and then identify correctly the required alarms
- Once the alarms have been identified find some possible correlation between the variables and those alarms.
- With the figured out correlations, try to predict an alarm in the future. E.g: given a set of correlated features (variable) at time T_{i-1} try to predict if at time T_i there will a fault or not.

The metrics/variables that contain the information about the alarms are:

- A5
- A9

3.4 Hints

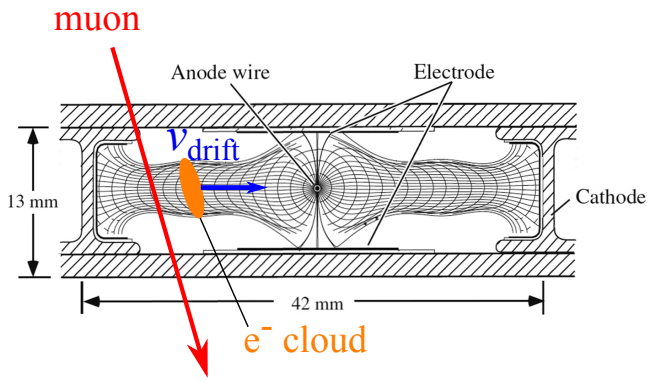
- When you have to resample a variable with respect to its frequency acquisition (eg: from 30s to 1min) keep into account the nature of the variable itself to perform an aggregation. (eg: if the variable is an integer, resampling using the mean is not the right solution, using the min or max aggregation could be better)
- If you need help or have questions contact us via email at the addresses: jacopo.pazzini@unipd.it

4 Batch analysis of cosmic rays using Drift Tubes detectors

Complexity degree: ■■□ The goal of this project is to analyze real data collected in a particle physics detector to reconstruct the trajectory of muons from cosmic rays.

4.1 Introduction

A set of muon detectors have been built and installed in Legnaro INFN Laboratories ($\sim$ 10 km far from Padova) and are currently used to collect signals from cosmic rays. The working principles of these detectors (named mini Drift Tubes or miniDTs) are based on ionization: charged particles traversing the volume of the detectors will ionize the gas mixture inside, thus producing electron-ion pairs. Specifically shaped electric fields make the electron cloud travel throughout the volume with an almost constant drift velocity ($v_{drift} = 53.8 \frac{\mu m}{ns}$). Close to a sensing anode wire, the intense electric field will produce further ionization and avalanche, thus increasing the signal gain.
Signals are collected on the anodic wire, then amplified, digitized, discriminated, and sent through a DAQ system where a set of FPGAs perform the Time-to-Digital Conversion (TDC): to each electron cloud reaching the wire a digital signal is associated corresponding to the time of collection.
Each digitized signal is commonly referred to as an "hit".

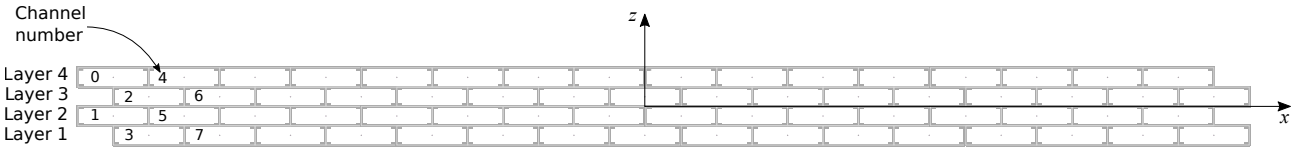


If the time of passage of the muon (usually referred to as t_0) is known, the TDC of a hit can be translated to a position inside the cell thanks to the constant v_{drift}, i.e.:

$$x_{hit} = (t_{hit} - t_0) v_{drift}$$

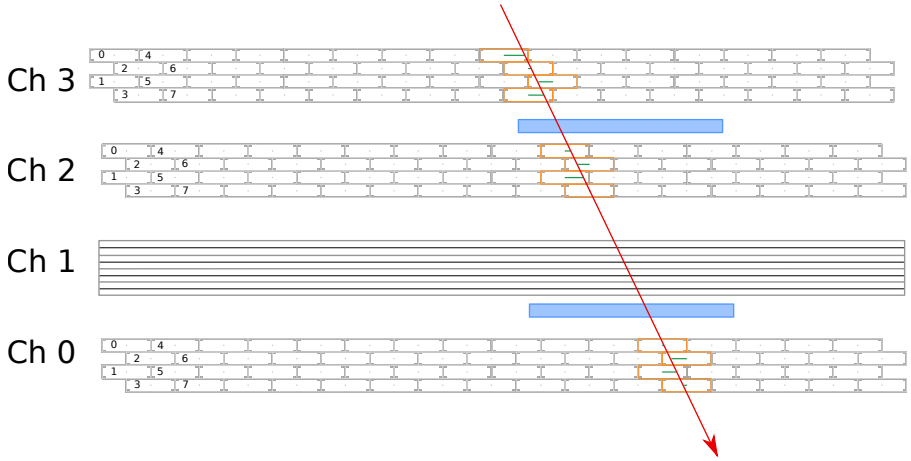
The time of the passage of the muon is however not generally known a priori and must be extracted from external information (or geometrical considerations).
Furthermore, an intrinsic left-right ambiguity is associated with the time-to-space conversion: the time information of a single hit is not enough to discriminate whether the muon passed in the right- or left-half of the cell.
A full reconstruction of a track spanning multiple cells is necessary to solve the ambiguity and identify the track trajectory throughout 1 or more DT chambers.

The geometry of one single detector (a miniDT "chamber") is composed of 64 cells arranged in 4 layers of 16 cells each. Adjacent layers are staggered by half of a cell width. Each cell has a transversal dimension of $42 \times 13\, mm^2$ (width$\times$height). The schema for one DT chamber is reported in the following figure:



Four DT chambers are stacked on top of each other in a “muon-telescope” configuration, where 3 chambers (0,2,3) are in the same orientation, and the fourth (1) is rotated 90 degrees to measure in the orthogonal view.

Two external detectors (plastic scintillator palettes, in blue in the figure) provide the external timing information for the particle passage, hence the t_0 information.



4.2 Task

Starting from the raw data collected by the DAQ of the detector, this project will require reconstructing the local and global track, relying on the external timing reference provided by the scintillator.

4.2.1 Data strucure

The raw dataset is provided on a cloud storage s3 bucket hosted on Cloud Veneto.

Name of the Bucket: `mapd-minidt-batch`
Link to visualize the list all contained files: `https://cloud-areapd.pd.infn.it:5210/swift/v1/AUTH_d2e941ce4b324467b6b3d467a923a9bc/mapd-minidt-batch/`

Instructions: `http://userguide.cloudveneto.it/en/latest/ManagingStorage.html#object-storage-experimental`

The dataset is composed of multiple binary files encoding the stream of hits collected:

- Every hit is encoded as a 64bit word
- Each word has the exact same structure

The 64b data-format complies to the following standard:

- 0-4 (5 bits) $\longrightarrow$ `TDC`
- 5-16 (12 bits) $\longrightarrow$ `BX`
- 17-48 (32 bits) $\longrightarrow$ `ORBIT`
- 49-57 (9 bits) $\longrightarrow$ `CHAN`
- 58-60 (3 bits) $\longrightarrow$ `FPGA`
- 61-63 (3 bits) $\longrightarrow$ `HEAD`

The data should be "unpacked" from raw objects into a structured format (a dataframe, or any equivalent).

4.2.2 Dataset cleansing

Before performing any attempt to reconstruct events, the dataset has to be sanitized to remove additional data not pertaining to the TDC hits.

The `HEAD` information in the data is an header informing which word is associated to TDC hits (marked with `HEAD == 2`) with respect to ancillary data (whose `HEAD` can be 0,1,3,4,5).

$\rightarrow$ All entries with `(HEAD != 2)` must therefore be dropped from the dataset/dataframe for the following analysis.

4.2.3 Detector mapping

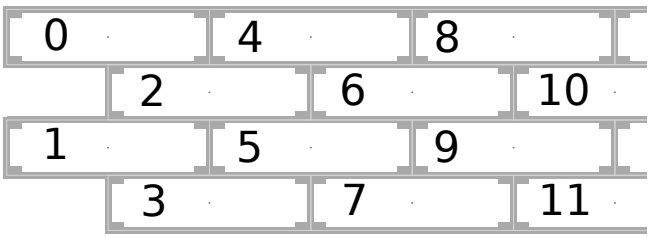
The mapping between the data-format and the chamber and cell geometry is the following:

- Chamber 0 $\rightarrow$ `(FPGA = 0) AND (CHANNEL in [0-63])`
- Chamber 1 $\rightarrow$ `(FPGA = 0) AND (CHANNEL in [64-127])`
- Chamber 2 $\rightarrow$ `(FPGA = 1) AND (CHANNEL in [0-63])`
- Chamber 3 $\rightarrow$ `(FPGA = 1) AND (CHANNEL in [64-127])`

The overall positioning of the reference frame (described in the previous figures) of the 4 chambers, as measured from the floor of the experimental hall where the detecors are located, is the following:

```
# shift of the reference frame for each Chamber (x, y, z)
shift_chamber = {
  0: {x: 0, y: 0, z:  219.8}, # Ch 0
  1: {x: 0, y: 0, z:  977.3}, # Ch 1
  2: {x: 0, y: 0, z: 1035.6}, # Ch 2
  3: {x: 0, y: 0, z: 1819.8}, # Ch 3
  }
```

Within each Chamber, the channel numbering follows a specific recurrent pattern:



4.2.4 Hit time information

The TDC time information for each signal is provided as a set of 3 counters, mimicking the clock distribution of the LHC collider:

- 1 `ORBIT` = 3564 `BX`
- 1 `BX` = 25 ns
- 1 `TDC` = 1/30 `BX`

An absolute time can be associated to each hit by converting the counters as follows

$$t_{\text{TDC hit}}(\text{in ns}) = 25 * (\texttt{ORBIT} * 3564 + \texttt{BX} + \texttt{TDC}/30)$$

The passage time of any muon, t_0, is provided by the external scintillator signal, which is encoded in the TDC hits corresponding to the selection:

- `(FPGA == 1) AND (CHANNEL == 128)`

Due to the delay induced by the signal transmission, the t_0 signal is delayed by roughly 95 ns with respect to the hits collected in the chambers.
This means that the time information from the scintillator will be roughly +95 ns with respect to the actual time of passage of a muon, and must be corrected when comparing the t_0 with the hit $t_{\text{TDC hit}}$.
Dedicated delays can be applied to fine-tune the correction accounting for even finer chamber-by-chamber differences:

```
# scintillator time offset by Chamber
time_offset_by_chamber = {
  0: 95.0 - 1.1, # Ch 0
  1: 95.0 + 6.4, # Ch 1
  2: 95.0 + 0.5, # Ch 2
  3: 95.0 - 2.6, # Ch 3
  }
```

4.2.5 Track reconstruction

To reconstruct the muon trajectory inside the detector, a basic pre-selection should be performed:

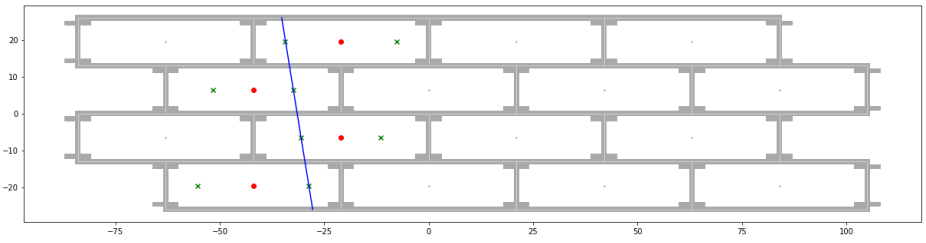
1. Hits must be grouped by Orbit
2. Select all and only hits for which a scintillator hit is found in the same orbit

3. Position each hit in the detector geometry by converting the hit time information into a spatial one. The nominal drift velocity is $v_{drift} = 53.8\,\frac{\mu m}{ns}$. At this stage the left/right ambiguity is not resolved, so 2 hit X positions (e.g. `XLEFT`, `XRIGHT`) should be assigned to each hit at this stage.

With the dataset correctly preprocessed, it is possible to estimate the trajectory of the muons inside the telescope. With no magnetic field to bend the passage of the particle, the path of the muons is expected to be a straight line.

Local tracks For each chamber, a linear fit should be performed to select the best combination of left/right hits and extract the parameters (angular parameter and intercept) of the "local-track".
The least-squares fit can be sufficient, and the combination with the least $\chi^2/ndof$ can be selected as the one with the highest quality.
(Alternative and smarter approaches can be attempted to speed up the fit, for example, based on alternative linear regression models to be trained on a fraction of the data.)

It could be very useful at this stage to produce plots of the fitted local track, and compare it to the location of the hits inside the detector geometry, to check the result of the fit on a small selection of orbits.



Global tracks Once the local-track selection is performed per each orbit and Chamber, the hits are univocally placed in the detector geometry, and a "global-track" fit can be performed using only chambers 0,2,3 (the ones measuring the same view).
Once more, a least-squares fit can be sufficient, although alternative approaches can be used.

The difference between the angle (in milli-radians) of the global track and the local track measured from Chamber 2 should be plotted, as this provides an estimate of the detector angular resolution.

4.3 Hints

- Orbits with a very large number of hits are enormously tricky to handle (multiple muons might have crossed the detector). To simplify the task, consider dropping the entire orbit if the number of hits exceeds a given amount (e.g. 15, but you should investigate which number can be considered reasonable)
- Try to avoid as much as possible performing loops over all the possible left/right combinations in an orbit.

- For the "unpacking" of the raw data, the use of RDDs/Bags is most suitable. Once the data is interpreted as a structured dataset, it might be useful (depending on the processing you have in mind) to move to Dask/pySpark DataFrames.
- If you need help or have questions contact us via email at the addresses: matteo.migliorini@pd.infn.it, jacopo.pazzini@unipd.it

5 Streaming processing of cosmic rays using Drift Tubes detectors

Complexity degree: ■■■ The goal of this project is to reproduce a real-time processing of real data collected in a particle physics detector and publish the results in a dashboard for live monitoring.

5.1 Introduction

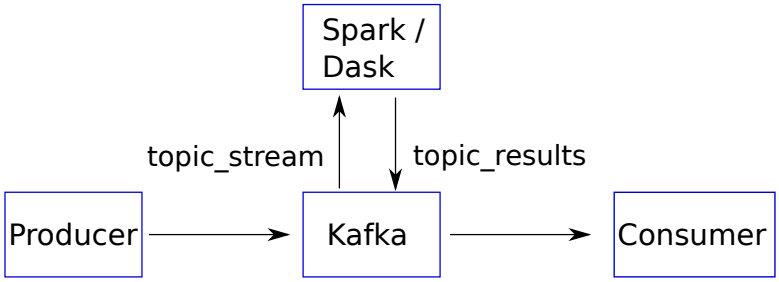
The introductory set of information on the underlying physics and detector details can be found in Section 4.1 of this document.

5.2 Task

Starting from data collected by the DAQ of the detector, this project will require creating a streaming application to monitor basic detector quality plots as an online streaming application.

Data should be produced to a Kafka topic by a stream-emulator script.
The processing of the hits should then be performed in a distributed framework such as Dask or pySpark.
The results of the processing will have to be re-injected into a new Kafka topic hosted on the same brokers.
A final consumer (ideally a simple python script) will perform the plotting and display live updates of the quantities evaluated online.
(`bokeh` or `pyplot` are good choices to create simple "dashboards", but other solutions can be applied for this task).

The overall schema of the architecture for this project can be simplified as follows:



5.2.1 Data strucure

The dataset is provided on a cloud storage s3 bucket hosted on Cloud Veneto.

Name of the Bucket: `mapd-minidt-stream`
Link to visualize the list all contained files: `https://cloud-areapd.pd.infn.it:5210/swift/v1/AUTH_d2e941ce4b324467b6b3d467a923a9bc/mapd-minidt-stream/`

Instructions: `http://userguide.cloudveneto.it/en/latest/ManagingStorage.html#object-storage-experimental`

The dataset is composed of multiple `txt` files formatted as comma-separated values, where each record (row) can be associated with each new signal processed in the DAQ of the detector.

The data-format is the following:

- `TDC_MEAS`
- `BX_COUNTER`
- `ORBIT_CNT`
- `TDC_CHANNEL`
- `FPGA`
- `HEAD`

A simple inspection of the data format will provide something like the following:

	HEAD	FPGA	TDC_CHANNEL	ORBIT_CNT	BX_COUNTER	TDC_MEAS
0	1	1	0	3387315431	0	130.000000
1	0	1	2	3387315431	1119	24.000000
2	4	1	0	3387315431	0	-0.573730
3	5	1	0	3387315431	0	45.500000
4	2	0	75	3387200947	2922	2.000000
5	2	0	105	3387200955	2227	29.000000
6	2	0	107	3387200955	2234	7.000000
7	2	0	126	3387200973	476	29.000000
8	2	1	55	3387200955	1797	12.000000
9	2	1	36	3387200956	2165	28.000000
10	2	1	51	3387200970	249	14.000000
11	2	1	90	3387200973	475	28.000000
12	4	0	0	3387339329	0	-0.537109
13	5	0	0	3387339329	0	-41.531250
14	3	0	0	3387339394	4353	0.000000

Data should be injected into a Kafka topic by a process emulating a continuous DAQ stream.

A realistic data production rate is about 1000 records/sec (i.e. 1000 rows/sec).
However, it might be beneficial to create a tunable-rate producer to adjust the emulated hit-rate.

5.2.2 Dataset processing

The processing can be performed with either Spark or Dask.
It is suggested to work with batches of data of a few (1 to 5) seconds.
Remember that the shorter the batch size, the fastest the processing should be to "keep-up" with the polling of the data; however, the longer the batch size, the larger the dataset to be processed for each batch.

The data processing will have to start with a data-cleansing step where all entries with `HEAD != 2` must be dropped from the dataset, as they will provide ancillary information not useful in the context of the project.

The mapping between the data-format and the detectors is the following:

- Chamber 0 → `(FPGA = 0) AND (TDC_CHANNEL in [0-63])`
- Chamber 1 → `(FPGA = 0) AND (TDC_CHANNEL in [64-127])`
- Chamber 2 → `(FPGA = 1) AND (TDC_CHANNEL in [0-63])`
- Chamber 3 → `(FPGA = 1) AND (TDC_CHANNEL in [64-127])`

The following information should be produced per each batch:

1. total number of processed hits, post-clensing (1 value per batch)
2. total number of processed hits, post-clensing, per chamber (4 values per batch)
3. histogram of the counts of active `TDC_CHANNEL`, per chamber (4 arrays per batch)
4. histogram of the total number of active `TDC_CHANNEL` in each `ORBIT_CNT`, per chamber (4 arrays per batch)

These pieces of information should be wrapped in one message per batch and injected into a new Kafka topic.

5.2.3 Live plotting

The results of the processing should be presented in the form of a continuously updating dashboard.

A python (or another language) script should consume data from the topic, and create plots and histograms of the aforementioned quantities.
Simple python modules such as `bokeh` or `pyplot` (in its free version) can help create a continuously updating webpage where to visualize the plots. You are however encouraged to explore and apply any other solution for this part of the task you might consider viable.

5.2.4 Extras

Provided the main goal of the project is met, additional bonus points can be obtained by extending the distributed processing studies to produce additional features.

A piece of external timing information, measuring the exact time of passage of a muon, is provided in the telescope by plastic scintillator palettes.
By selecting only the hits compatible with a scintillator signal, we can observe interesting features in the data.
For instance, a "muon-radiography" can be performed, showing only the number of active channels compatible with the passage of the muons.

Additionally, the properties of the detector provide an upper boundary to the hit collection time: the electron cloud produced by any given muon cannot travel further than 1/2 of a cell before being collected on the sensing wire.
By plotting the difference of the hit collection times minus the time of passage of the related muon, the resulting distribution is thus expected to show a box-shaped profile, with sharp rising and falling edges.

To be able to define these quantities, the hit time information should be translated into an absolute timing feature.
The TDC time information for each signal is provided as a set of 3 counters, mimicking the clock distribution of the LHC collider:

- 1 `ORBIT_CNT` = 3564 `BX_COUNTER`
- 1 `BX_COUNTER` = 25 ns
- 1 `TDC_MEAS` = 1/30 `BX_COUNTER`

An absolute time (in nanoseconds) can be associated with each hit by converting the counters as follows

$$t_{\text{TDC hit}}(\text{in ns}) = 25 * (\texttt{ORBIT\_CNT} * 3564 + \texttt{BX\_COUNTER} + \texttt{TDC\_MEAS}/30)$$

The passage time of any muon, t_0, is provided by the external scintillator signal, which is encoded in the TDC hits corresponding to the selection:

- `(FPGA == 1) AND (TDC_CHANNEL == 128)`

Due to the delay induced by the signal transmission, the t_0 signal is delayed by roughly 95 ns with respect to the hits collected in the chambers.
This means that the time information from the scintillator will be roughly +95 ns with respect to the actual time of passage of a muon, and must be corrected when comparing the t_0 with the hit $t_{\text{TDC hit}}$.
Dedicated delays can be applied to fine-tune the correction accounting for even finer chamber-by-chamber differences:

```
# scintillator time offset by Chamber
time_offset_by_chamber = {
  0: 95.0 - 1.1, # Ch 0
  1: 95.0 + 6.4, # Ch 1
  2: 95.0 + 0.5, # Ch 2
  3: 95.0 - 2.6, # Ch 3
  }
```

An `ABSOLUTETIME` information should thus be associated with all hits during the processing phase by converting the counters to the proper time information in ns.
Only for those hits with a scintillator signal within the same orbit, a `DRIFTIME` can be finally defined, corresponding to the `ABSOLUTETIME` difference between each hit and the scintillator (from the same orbit).
In case of multiple scintillator hits in the same orbit, arbitrarily choose one (e.g. the one `min`).
Two additional types of results can be added to the list of the processing, and displayed on the live visualization:

1. histogram of the counts of active `TDC_CHANNEL`, per chamber, ONLY for those orbits with at least one scintillator signal in it (4 arrays per batch)
2. histogram of the `DRIFTIME`, per chamber (4 arrays per batch)

5.3 Hints

- Start by prototyping the processing for a single static batch before extending to the streaming part
- Dealing with a structured dataset, it might be useful to use directly Dask/pySpark DataFrames instead of Bags/RDDs
- The Kafka messages results of the distributed processing could be formatted in a way to encode, for each histogram, an array of bins and counts. For instance, a `json` structured as follows might be one (although not the only one!) viable alternative:

```
msg_json = {
  "histo_1" = { "bin_edges" = [...],
                "bin_counts"= [...]
              },
  "histo_2" = {...},
  ...
  }
```

- If you need help or have questions contact us via email at the addresses: matteo.migliorini@pd.infn.it, jacopo.pazzini@unipd.it

6 Streaming processing of the QUAX experiment data for the detection of galactic axions

Complexity degree: ■■■ The goal of this project is to create a quasi real-time processing chain of the data produced by the QUAX experimental apparatus, and to create a live monitoring system of the detector data.

6.1 Introduction

The axion is a hypothetical particle introduced to solve the strong CP problem of Quantum Chromo Dynamics.

It is speculated that axions may also constitute the dark matter (DM) content in our Galaxy. Astrophysical observations and cosmological considerations suggest a favoured mass range of $1\,\mu\mathrm{eV} < m_a < 10\,\mathrm{meV}$.

The QUAX experiment aims at the axion detection by using a copper cavity immersed in a static magnetic field of $8.1\,\mathrm{T}$, cooled down at a working temperature of about $150\,\mathrm{mK}$.

The axion is expected to couple with the spin of the electron, interacting with the cavity and inducing a radio-frequency that can be sensed via a Josephson parametric amplifier.

For a given configuration of the RF cavity, a scan of the phase of the electromagnetic field is performed to be able to possibly identify a localised excess, a hint of the coupling of an axion with the photon.

The data acquisition system of the QUAX experiment generates two streams of digitized reading of the amplifiers, representing the real and imaginary components of the measured phase.

A single phase scan will prove to be dominated by noise. To improve the signal over noise ratio, a QUAX data-taking run extends over a long time (up to weeks), repeating the scans over multiple times.

Data are saved locally on the DAQ servers in the form of binary files. Each binary file corresponds to a multitude of continuous scans performed in the entire frequency range. A single pair of raw files is thus representative of only a few seconds of data taking, but are already including several (thousands) scans.

The processing of the raw data comprises two phases:

1. for each scan, run a Fourier transform to move from the time domain to the frequency domain
2. average (in bins of frequency) all scans in a data-taking run, to extract a single frequency scan

This (relatively simple) procedure is highly parallelizable, and should be implemented in a quasi-online pipeline for two main reasons:

- Monitoring the scans during the data taking is paramount to promptly spot and identify possible issues in the detector setup or instabilities in the condition of the experiment. Being able to run this procedure on-the-fly and visualize the results in a live-monitoring dashboard can provide valuable information and prevent weeks-long runs to be wasted.
- Data is continuously produced with a very large rate, and the local storage provided by the DAQ server of the QUAX experiment is not really suited for large-volume and long-lasting datasets. A real-time ETL (Extract-Transform-Load) pipeline is thus envisioned.

6.2 Task

Starting from data collected by the DAQ of the detector, this project will require creating a streaming application to monitor basic detector quality plots as an online streaming application.

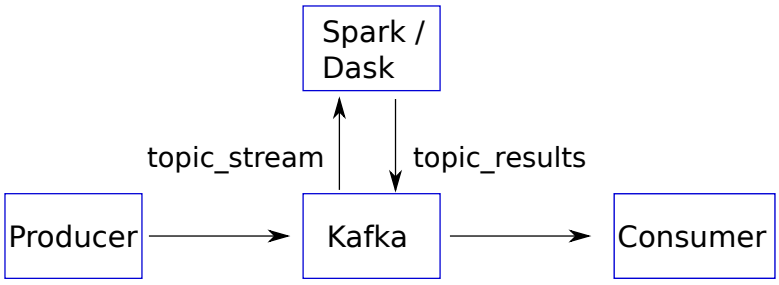
Data should be produced to a Kafka topic by a stream-emulator script, starting from the files of a previous QUAX data-taking run provided in the CloudVeneto object store area. Each file is supposedly produced every few (~ 10) seconds due to the fixed ADC scanning rate and the fixed size of files written to disk. However, it is useful to produce a simple python script capable of tuning the file processing rate.

The processing of each file runs should then be performed in a distributed framework such as Dask (preferrably) or pySpark. The processing pipeline should start from the pair of files (real/imaginary phase components), unpack them according to their schema (discussed below), and split into scans. For each scan, a FFT should be executed in parallel, and the results of all FFTs should finally be merged in a single "averaged-FFT".

The results of the processing (the "averaged-FFT" for each file) will have to be re-injected into a new Kafka topic hosted on the same brokers.

A final consumer (ideally a simple python script) will perform the plotting and display live updates of the scans evaluated online (thus publishing 1 new plot every ~ 10 seconds), and continuously update the entire "run-wide" scan. (`bokeh` or `pyplot` are good choices to create simple "dashboards", but other solutions can be applied for this task).

Analogously to the mini-DT stream processing project discussed in Sec. 5, the overall schema of the architecture for this project can be simplified as follows:



6.2.1 Data strucure

The dataset is provided on a cloud storage s3 bucket hosted on Cloud Veneto.

Name of the Bucket: `quax`
Link to visualize the list all contained files: `https://cloud-areapd.pd.infn.it:5210/swift/v1/AUTH_d2e941ce4b324467b6b3d467a923a9bc/quax/`

Instructions: `http://userguide.cloudveneto.it/en/latest/ManagingStorage.html#object-storage-experimental`

The dataset is composed of 2 sets (named `duck_i` and `duck_q` respectively) of `.dat` binary files, each one comprised of a continuous series of ADC readings from the amplifier. Each ADC reading is written in the raw files as a `32 bit` floating point value. For instance this can be interpreted in plain Python using a `struct.unpack` call.
The ADC readout frequency is $2 \times 10^6\,\mathrm{Hz}$ (2 MegaSample per second, or 2MS/s), thus resulting in a raw data throughput of 128 Mbps (16 MB/s).
During data taking the readouts are formatted in `.dat` file such that each file is comprised of 8193×2^{10} samples.
This results in producing a pair of `.dat` files (`duck_i` and `duck_q`) every $4.2\,\mathrm{s}$.

The raw data files should be injected in pairs into a Kafka topic by a process emulating a quasi-continuous DAQ stream of files.
A realistic data production rate is about 1 file pair every 4-5 seconds.
However, as anticipated, it might be beneficial to create a tunable-rate producer to adjust the rate, and opt for a slower data ingestion rate in the case of issues with the processing layer.

6.2.2 Dataset processing

The data processing must with the readout of the pairs of files, and the conversion from binary to floating point values of their content.

The real and imaginary components should be sliced according to the (32 bits) float data format and the readout frequency, from which it should result $8193 * 2^{10}$ float values from each file. Slices of the real and imaginary components must be combined together consistently.

Per each slice, a FFT is to be performed to extract the relative power spectrum.
The Fourier Transform will have a full bandwidth of $2\,\mathrm{MHz}$, spanning the range $(-1\,\mathrm{MHz}, 1\,\mathrm{MHz})$.
To perform the FFT it should be used a number of bins equal to $n_{bins} = 3 \times 2^{10} = 3027$.
A total of $\frac{8193 \times 2^{10}}{3 \times 2^{10}} = 2731$ FFTs will thus result out of each pair of files.

Eventually, for each pair of files (thus per each batch), an average of all FFTs is to be performed to produce an averaged-out power spectrum, where the noise contribution is reduced.

The output of the processing of each batch should thus be one message with all the appropriate features to be representative of the input data. At least, the message should

contain the power spectra values per each of the FFT bins, and the standard deviation of each bin.

The processed data should be sent to another Kafka topic for later consumption.

6.2.3 Live plotting

The results of the processing should be presented in the form of a continuously updating dashboard.

At minimum, one plot should be shown and update for each batch, representing the power spectrum of the newly processed batch, and its error.
This is going to mimic the live data-monitoring of the detector.

In addition, a "cumulative" power spectrum, resulting from the cumulative average of all batches processed during the run, may be included.

A python (or any other language) script should consume data from the topic, and create the plots.
Simple python modules such as `bokeh` or `pyplot` (in its free version) can help create a continuously updating webpage where to visualize the plots. You are however encouraged to explore and apply any other solution for this part of the task you might consider viable.

6.3 Hints

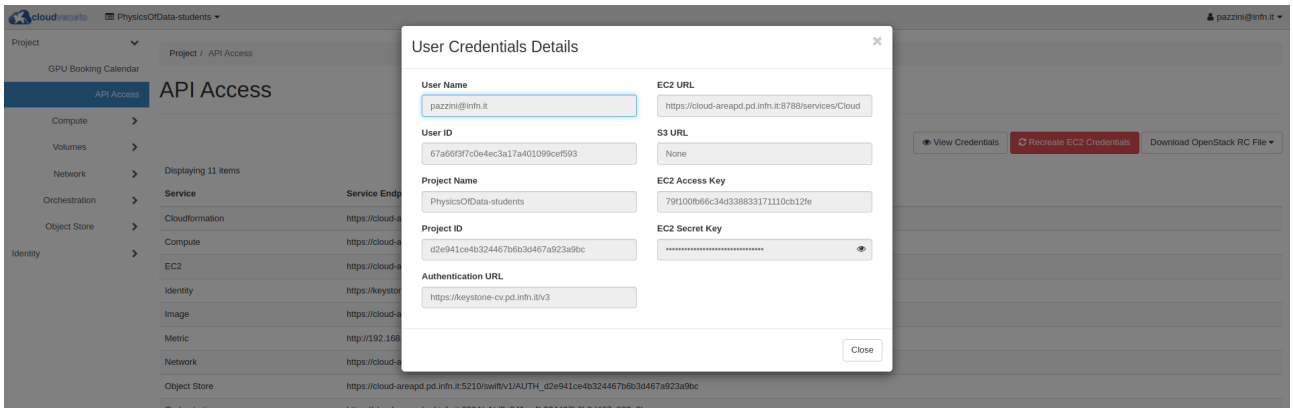
- Start by prototyping the processing for a single static batch before extending to the streaming part
- Dealing with a unstructed dataset, it might be useful to first use Dask/pySpark low-level APIs (Bags/RDDs) for the first data ingestion part of the task
- The Kafka messages results of the distributed processing could be formatted in a way to encode, for each spectrum, an array of bins and values. For instance, a `json` structured as follows might be one (although not the only one!) viable alternative:

```
msg_json = {
  "average" = { "frequency" = [...],
                "value"= [...],
                "rms"= [...]
              },
}
```

- If you need help or have questions contact us via email at the addresses: jacopo.pazzini@unipd.it

A Using the CloudVeneto cloud storage

In CloudVeneto is implemented an object store solution (a Cloud Storage). It is possible to interact with the object store via the Amazon S3 (which stands for Simple Storage Service) API.
Please refer to the CloudVeneto documentation guide before starting using it.
Most importantly, read the part concerning the access of the storage using the S3 interface, which also describes how to get your private credentials, at the link.
In order to get your own EC2 credentials, in the CloudVeneto Dashboard you will have to navigate into `Project`, then `API Access` and then click on `View Credentials`.



Accessing a file from python for testing purposes If we want to download a few files from S3, to use them in a local environment, we can use the `boto3` python module:

```
import boto3
s3_client = boto3.client('s3',
                         endpoint_url='https://cloud-areapd.pd.infn.it:5210',
                         aws_access_key_id='79f100fb66c34d338833171110cb12fe',
                         aws_secret_access_key='--------------------------------',
                         verify=False)
s3_client.download_file('mapd-minidt-stream', 'data_000001.txt','my_local_file.txt')
```

This example will download the `data_000001.txt` file (in proper terms, the *object* having `data_000001.txt` as the relative *key*) from the `mapd-minidt-stream` folder (in proper terms, *bucket*), into your local folder under the name `my_local_file.txt`.

Accessing data using Dask Dask uses the `boto3` library to handle requests to remote object stores via S3, so a very similar implementation to the previous one can be performed.
For instance, if willing to load all objects in a bucket into a Dask DataFrame, one could do:

```
import dask.dataframe as dd
df = dd.read_csv('s3://mapd-minidt-stream/data_*.txt',
                 storage_options={
                     'key':'79f100fb66c34d338833171110cb12fe',
                     'secret':'--------------------------------',
                     'client_kwargs': {
                         'endpoint_url': 'https://cloud-areapd.pd.infn.it:5210',
                         'verify':False
```

```
                    }
                })
# print the content of the df
df.head()
```

Accessing data using Spark Spark does not rely on `boto3` to handle requests to the remote object store via S3, but instead it relies on a `hadoop`-S3 "connector".
To instruct Spark to handle the requests to the remote object store one should follow a few steps.

1. In each node of the cluster, import the CloudVeneto certificate

```
wget https://repo1.maven.org/maven2/org/apache/hadoop/hadoop-aws/3.2.0/hadoop-aws-3.2.0.jar \
    -P $SPARK_HOME/jars/
wget https://repo1.maven.org/maven2/com/amazonaws/aws-java-sdk-bundle/1.11.375/aws-java-sdk-bundle-1.11.375.jar \
    -P $SPARK_HOME/jars/
```

2. Also, import the `jar` files to instruct Spark to handle connections via the S3 interface

```
wget https://raw.githubusercontent.com/CloudVeneto/CertCA/master/CloudVenetoCAs.pem
keytool -trustcacerts \
        -keystore /usr/lib/jvm/java-11-openjdk-amd64/lib/security/cacerts \
        -storepass changeit \
        -alias CloudVeneto \
        -import -file CloudVenetoCAs.pem
```

3. Start the Spark master and workers normally

4. When creating a Spark Session, load the proper java modules

```
from pyspark.sql import SparkSession

spark = SparkSession.builder \
    .master("spark://localhost:7077")\
    .appName("Your Spark Application Name")\
    .config('spark.jars.packages', 'org.apache.hadoop:hadoop-common:3.2.0')\
    .config('spark.jars.packages', 'org.apache.hadoop:hadoop-aws:3.2.0')\
    .config('spark.jars.packages', 'com.amazonaws:aws-java-sdk:1.11.375')\
    .config("spark.executor.memory", "512m")\
    .config("spark.sql.execution.arrow.pyspark.enabled", "true")\
    .config("spark.sql.execution.arrow.pyspark.fallback.enabled", "false")\
    .config('spark.hadoop.fs.s3a.aws.credentials.provider', 'org.apache.hadoop.fs.s3a.SimpleAWSCredentialsProvider')\
    .config('spark.hadoop.fs.s3a.access.key', '79f100fb66c34d338833171110cb12fe')\
    .config('spark.hadoop.fs.s3a.secret.key', '--------------------------------')\
    .config('spark.hadoop.fs.s3a.endpoint', 'https://cloud-areapd.pd.infn.it:5210')\
    .config("spark.hadoop.fs.s3a.impl", "org.apache.hadoop.fs.s3a.S3AFileSystem") \
    .config("spark.hadoop.fs.s3a.metadatastore.impl", "org.apache.hadoop.fs.s3a.s3guard.NullMetadataStore") \
    .config("spark.hadoop.fs.s3a.path.style.access", "true") \
    .config("spark.hadoop.fs.s3a.connection.ssl.enabled","false") \
    .config("com.amazonaws.sdk.disableCertChecking","true") \
    .getOrCreate()
# create and print the content of the data rdd
rdd = spark.sparkContext.textFile('s3a://mapd-minidt-stream/data_*.txt')
rdd.take(3)
```