Automatic Computation Framework

Release 0.0.1

Antonello Aita

Jul 29, 2021

CONTENTS:

1 Data calculation **3**

2 Data collection **5**

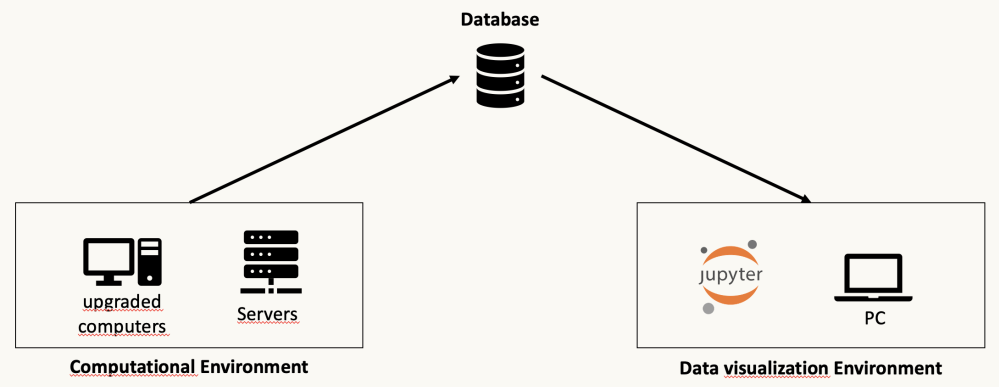
3 Data Plot **7**
- 3.1 How to use it 7
- 3.2 Code Description 7
- 3.3 Heisenberg 8

4 Indices and tables **9**

Index **11**

Conduct a research activity that implies modelling complex system and/or find optimal parameters in variational methods could be demanding in terms of time and resources. This is even more evident in the quantum computing realm, in this period of near term devices we are used to benchmark our methods with classical simulations.

This framework was born when we started our research project published here: link. We realized soon that the way we organized our simulation to exploit the usage of server/cluster or HPC environment could have been generalised to several use cases. For those reasons we publish it here so that we could be of help for researchers in their daily activities. The large amount of research project are conducted in teams and this implies having more than one environment and also data generated and/or collected by different people, could be a problem without having a centralized repository. For this reason we decided to design and build a flexible and light weighted system capable to be used by different people enabling data collection according to the same rules and structures. The main research tasks are split into three different pieces:



The first part enables parallel calculation on 'computational environment', the second part is related to the managing of data, and for this framework we provide facilities to connect the IBM Cloud cloudant database. The third part consists retrieving information from the database and eventually implement some plot functions where the user can manipulate data using a simple jupyter notebook or his own python script.

CHAPTER

ONE

DATA CALCULATION

This part is dedicated to the intensive numerical calculation (real quantum devices or simulated on classical devices). Usually this part is conducted on upgraded computers, HPC clusters or quantum cloud devices. Some simulated calculation need a huge amount of memory and execution time and with this tool we can keep our results organized and structured avoiding to lose results or manually move files through different environments. Moreover, in this configuration the obtained results are available to the whole team that have access to the database and with the possibility to configure also a dashboard.

CHAPTER

TWO

DATA COLLECTION

We decided to collect all the obtained results on a **No-SQL json based database** because the collection and serialization of data using the json format is the easiest way on a python class. In this direction we defined a json attribute named *final_json* on the constructor function of the metaclass to collect all the metadata coming from the performed run. We give the user the possibility to save all the results (raw and post processed) as a new record on the database at the end of the calculation.

> **Warning:** Remember to fill the `final_json` attribute with only serializable elements , otherwise the db saving will fail

CHAPTER

THREE

DATA PLOT

This last part of the framework component is dedicated to retrieve data from the database and create plot and reports in order to transform data for a most effective communicative power. In this framework also this part has been written in python in order to be used in the jupyter environment to allow an easy and quick implementation of post processing of saved data and provide graphical representation.

Considering that data are produced and stored in near real time and that each plot is built quering the database, we could say say that each plot is a live report of the obtained results.

3.1 How to use it

This documentation allow you to create your own class and implement your own methods while keeping all the basic functionalities as indicated in this first implementation.

3.2 Code Description

Let's provide the code description of the whole framework and implemented examples:

3.2.1 Execution

3.2.2 Metaclass

This part of the framework contains the backbone of the computation and plot part. There have been implemented metaclass of referement for the plot and computation class implementation tailored for your own case.

Important: Both classes take as input, when instantiated, data coming from a *private_config.json*, if the config file is missing then you can feed this information manually

---

class CAF.MetaClass.**ComputationClass**

This metaclass contains the backbone of the computation class . It is composed by 3 different elements:

1) **init** : the constructor function through we collect credential to log on database and IBMQ system
2) **run** : function designated to gather parameters for each run, then run calculation (on local or remote) and then save the results in the 'final_json' class attribute – To implement your class –
3) **save** : function designated to save results on database

NB: Instantiating the class will not require any argoument because an automatic feeding has been implemented using a decorator.

abstract run()
Run function

This function **must** be implemented in the class you are going to implement

save(*db_name: str*)
Save function

This function save the obtained results presente on the internal json named "final_json" that is populated during the perford calculation

Parameters **db_name** (*str*) – name of the db created on the cloud instance to collect data

class CAF.MetaClass.**PlotClass**
This metaclass is designed to collect all the routines to query and show results. In this metaclass are provided the basic functions to instantiate the connection with database and query on it. The contructor function will retrieve the db credential and the db name. THe db_name has to be manually feeded by the user, in order to connect with the correct data source.

query(*selector_dict: dict*, *field_list: list*)
Query

This function is designet to automatically query on db data just providing a selector and a field list.

Parameters

- **selector_dict** (*dict*) – dictionary containing db query following the cloudant standard
- **field_list** (*list*) – list of field to retrieve for the queried elements

3.2.3 Plot

3.3 Heisenberg

CHAPTER

FOUR

INDICES AND TABLES

- genindex
- modindex
- search

INDEX

C

ComputationClass (*class in CAF.MetaClass*), 7

P

PlotClass (*class in CAF.MetaClass*), 8

Q

query() (*CAF.MetaClass.PlotClass method*), 8

R

run() (*CAF.MetaClass.ComputationClass method*), 8

S

save() (*CAF.MetaClass.ComputationClass method*), 8