OPAL Algorithms Documentation

Release 0.0.1

Shubham Jain, Axel Oehmichen

Sep 09, 2019

Contents:

opalalgorithms provides an interface to implement algorithms to be used in OPAL project. Each algorithm will be run using multiprocessing library.

CHAPTER 1

usage instructions

Creating algorithm has two main parts to it

- implementation
- testing

opalalgorithms provides you with utilities to do both.

1.1 implementation

We use *opalalgorithms.core.base* that provides utilities for implementing an algorithm for OPAL. An algorithm will look like as follows:

```
"""Sample algorithm 1 to return home of users."""
from __future__ import division, print_function
from opalalgorithms.core import OPALAlgorithm


class SampleAlgo1(OPALAlgorithm):
    """Calculate population density."""

    def __init__(self):
        """Initialize population density."""
        super(SampleAlgo1, self).__init__()

    def map(self, params, bandicoot_user):
        """Get home of the bandicoot user.

        Args:
            params (dict): Request parameters.
            bandicoot_user (bandicoot.core.User): Bandicoot user object.

        """
```

(continues on next page)

(continued from previous page)

```
        home = bandicoot_user.recompute_home()
        if not home:
            return None
        return {getattr(home, params["resolution"]): 1}
```

1.2 testing

We provide utilities to test the algorithm you create. Before running the algorithm, it is advised you setup the apparmor and codejail as mentioned here.

Use tests/generate_data.py to generate data after installing the opalalgorithms library. You can run your algorithm with the following code:

```
"""Test population density algorithm."""
from __future__ import division, print_function
from opalalgorithms.utils import AlgorithmRunner


num_threads = 3
data_path = 'data'


def get_algo(filename):
    algorithm = dict(
        code=open(filename).read(),
        className='ClassNameOfYourAlgo'
    )
    return algorithm


def run_algo(algorithm_filename, params):
    """Run an algorithm."""
    algorithm = get_algo(algorithm_filename)
    algorunner = AlgorithmRunner(
        algorithm, dev_mode=False, multiprocess=True, sandboxing=True)
    result = algorunner(params, data_path, num_threads)
    return result


if __name__ == '__main__':
    """Test that algorithm runner runs successfully."""
    params = dict(
        sample=0.2,
        resolution='location_level_1')
    assert run_algo('/path/to/your/algo.py', params)
```

CHAPTER 2

opalalgorithms

This library provides an interface and tools for implementation of algorithms to be used in OPAL project.

2.1 opalalgorithms.core

This is the core library of opalalgorithms which specifies the standard class which is to be inherited when implementing any algorithm for opal project.

2.1.1 opalalgorithms.core.base

This is the base module for all algorithms.

Base class for implementing any algorithms for OPAL computation.

class opalalgorithms.core.base.**OPALAlgorithm**

Base class for OPAL Algorithms.

The class can be used in the following way:

```
algo = OPALAlgorithm()
result = algo.map(params, bandicoot_user)
```

map(*params*, *bandicoot_user*)

Map users data to a single result.

Parameters

- **params** (*dict*) – Parameters to be used by each map of the algorithm.
- **bandicoot_user** (*bandicoot.user*) – Bandicoot user.

Returns A dictionary representing with keys as string or tuple, and values as int or float which will be aggregated by the reducer.

Return type dict

2.1.2 opalalgorithms.core.base

This is the base module for all privacy algorithms to be applied on result.

Base class for implementing any privacy algorithms for OPAL computation.

class opalalgorithms.core.privacy.**OPALPrivacy**
Base class for OPAL Privacy Algorithms.

The class can be used in the following way:

```
privacyalgo = OPALPrivacy()
result = privacyalgo(params, result, salt)
```

2.2 opalalgorithms.utils

Utilities for testing opalalgorithms and running them.

2.2.1 opalalgorithms.utils.datagenerator

Data generator class for generating data for testing purposes.

class opalalgorithms.utils.datagenerator.**OPALDataGenerator**(*num_antennas*, *num_antennas_per_user*, *num_records_per_user*, *bandicoot_extended=True*)

Generate data as per OPAL formats for testing purposes.

Parameters

- **num_antennas** (*int*) – Total number of antennas available.
- **num_antennas_per_user** (*int*) – Total number of different antennas a user can connect to.
- **num_records_per_user** (*int*) – Number of records generated for each user over the complete year.
- **bandicoot_extended** (*bool*) – To use bandicoot extended format or old format.

Todo:

- Remove bandicoot extended once that library is fixed.

---

generate_data()
Generate data for a single user.

2.2.2 opalalgorithms.utils.algorithmrunner

Algorithm runner class to run algorithms during computation.

Given an algorithm object, run the algorithm.

opalalgorithms.utils.algorithmrunner.**mapper**(*writing_queue*, *params*, *file_queue*, *algorithm*, *dev_mode=False*, *sandboxing=True*, *python_version=2*)

Call the map function and insert result into the queue if valid.

Parameters

- **writing_queue** (*mp.manager.Queue*) – Queue for inserting results.
- **params** (*dict*) – Parameters to be used by each map of the algorithm.
- **users_csv_files** (*list*) – List of paths of csv files of users.
- **algorithm** (*dict*) – Dictionary with keys *code* and *className* specifying algorithm code and className.
- **dev_mode** (*bool*) – Should the algorithm run in development mode or production mode.
- **sandboxing** (*bool*) – Should sandboxing be used or not.
- **python_version** (*int*) – Python version being used for sandboxing.

opalalgorithms.utils.algorithmrunner.**collector**(*writing_queue*, *params*, *dev_mode=False*)

Collect the results in writing queue and post to aggregator.

Parameters

- **writing_queue** (*mp.manager.Queue*) – Queue from which collect results.
- **results_csv_path** (*str*) – CSV where we have to save results.
- **dev_mode** (*bool*) – Whether to run algorithm in development mode.

Returns True on successful exit if *dev_mode* is set to False.

Return type bool

Note: If *dev_mode* is set to true, then collector will just return all the results in a list format.

---

opalalgorithms.utils.algorithmrunner.**is_valid_result**(*result*)

Check if result is valid.

Parameters **result** – Output of the algorithm.

Note: Result is valid if it is a dict. All keys of the dict must be be a string. All values must be numbers. These results are sent to reducer which will sum, count, mean, median, mode of the values belonging to same key.

Example:

- {"alpha1": 1, "ant199": 1, ..}

Returns Specifying if the result is valid or not.

Return type bool

Todo:

- Define what is valid with privacy and other concerns

---

opalalgorithms.utils.algorithmrunner.**process_user_csv**(*params*, *user_csv_file*, *algorithm*, *dev_mode*, *sandboxing*, *jail*)

Process a single user csv file.

Parameters

- **params** (*dict*) – Parameters for the request.
- **user_csv_file** (*string*) – Path to user csv file.
- **algorithm** (*dict*) – Dictionary with keys *code* and *className* specifying algorithm code and className.
- **dev_mode** (*bool*) – Should the algorithm run in development mode or production mode.
- **sandboxing** (*bool*) – Should sandboxing be used or not.
- **jail** (*codejail.Jail*) – Jail object.

Returns Result of the execution.

Raises SafeExecException – If the execution wasn't successful.

opalalgorithms.utils.algorithmrunner.**get_jail**(*python_version=2*)

Return codejail object.

Note:

- **Please set environmental variables *OPALALGO_SANDBOX_VENV*** and *OPALALGO_SANDBOX_USER* before calling this function.
- ***OPALALGO_SANDBOX_VENV* must be set to the path of the sandbox** virtual environment.
- ***OPALALGO_SANDBOX_USER* must be set to the user running the** sandboxed algorithms.

---

class opalalgorithms.utils.algorithmrunner.**AlgorithmRunner**(*algorithm*, *dev_mode=False*, *multiprocess=True*, *sandboxing=True*)

Algorithm runner.

Parameters

- **algorithm** (*dict*) – Dictionary containing *code* and *className*.
- **dev_mode** (*bool*) – Development mode switch
- **multiprocess** (*bool*) – Use multiprocessing or single process for complete execution.
- **sandboxing** (*bool*) – Use sandboxing for execution or execute in unsafe environment.

__call__(*params*, *data_dir*, *num_threads*, *weights_file=None*)

Run algorithm.

Selects the csv files from the data directory. Divides the csv files into chunks of equal size across the *num_threads* threads. Each thread performs calls map function of the csv file and processes the result. The collector thread, waits for results before posting it to aggregator service.

Parameters

- **params** (*dict*) – Dictionary containing all the parameters for the algorithm
- **data_dir** (*str*) – Data directory with csv files.

- **num_threads** (*int*) – Number of threads
- **weights_file** (*str*) – Path to the json file containing weights.

Returns Amount of time required for computation in microseconds.

Return type int

2.2.3 opalalgorithms.utils.date_helper

Utility functions to help manipulate dates within different algorithms.

opalalgorithms.utils.date_helper.**is_date_between**(*start*, *end*, *date*)
Check if data is between start and end datetime.

Parameters

- **start** (*string*) – Starting datetime, must be of form '%Y-%m-%d %H:%M:%S'
- **end** (*string*) – Ending datetime, must be of form '%Y-%m-%d %H:%M:%S'
- **date** (*string*) – Date to be checked, must be of form '%Y-%m-%d %H:%M:%S'

Returns Whether date is between end and start date.

Return type bool

opalalgorithms.utils.date_helper.**is_date_greater**(*ref*, *date*)
Check if date is greate than reference time.

Parameters

- **ref** (*string*) – Reference datetime against which we need to check, must be of form '%Y-%m-%d %H:%M:%S'.
- **date** (*string*) – Date which is to be checked, must be of form '%Y-%m-%d %H:%M:%S'

Returns Whether date is greater than reference.

Return type bool

CHAPTER 3

Indices and tables

- genindex
- modindex
- search

Python Module Index

o

Index

Symbols

A

C

G

I

M

O

P