13. ML for smart caching

Author(s)

Name	Institution	Mail Address	Social Contacts
Mirco Tracolli	INFN Section Perugia	mirco.tracolli@pg.infn.it	N/A

How to Obtain Support

Mail	mirco.tracolli@pg.infnit
Social	N/A
Jira	N/A

General Information

ML/DL Technologies	ML/RL
Science Fields	High Energy Physics, Computing, Cache
Difficulty	medium
Language	English
Type	runnable, external resource

Software and Tools

Programming Language	Python3, Go
ML Toolset	Keras, Tensorflow, sklearn
Additional libraries	
Suggested Environments	bare Linux Node

Needed datasets

Data Creator	Ad hoc tool
Data Type	logs of data analysis file requests
Data Size	depending on the configuration of the generator tool
Data Source	data generator tool: https://github.com/Cloud-PG/dataset-generator

Short Description of the Use Case

Accessing data is a very important task in the data analysis flow and usually, there are several frameworks and software layers that make it possible to accomplish such a target. In particular, recent studies are focused on Data Lake Cache management to optimize the data flow through the clients. The caching layer is a very important part of the data flow that should be optimized, especially if the data are distributed and also the compute centers are decentralized.

Since the infrastructure part is in continuous development, a simulation environment is needed to test and experiments with different approaches to improve the caching performances in a Data Lake. As a result, this project allows you to have a playground where to test new features or algorithms.

How to execute it

Base requirement packages (Debian based distro)

- git
- python3 (python3-dev, python3-pip)
- golang

Get the tools

First, you need the data generator to create a synthetic dataset. The data generator used in this project is the following:

- https://github.com/Cloud-PG/dataset-generator

With such a generator, you can create a dataset that has requests similar to the HEP context in which this project was born.

Note: all the `Python` commands refers to the `Python 3` environment

```
# Create a folder for the whole project
mkdir myProject
cd myProject

# Download the repository
git clone https://github.com/Cloud-PG/dataset-generator.git

# Enter the project folder
cd dataset-generator

# Install dependencies
pip3 install -r requirements.txt

# Back to main project folder
cd ..
```

Secondly, you can download the simulation environment:

```
# Download the repository
git clone --branch v2.0.2 https://github.com/Cloud-PG/smart-cache.git

# Enter the project folder
cd smart-cache

# Install the Utilities
cd SmartCache/sim/Utilities
pip3 install -e .
cd ../../..

# Install the Probe module
cd Probe
python3 setup.py install
cd ..

# Install general requirements
pip3 install coloredlogs colorama dash_daq biokit

# Back to main project folder
cd ..
```

Create a dataset

You can use a preset config to generate a dataset with the following command:

```
python3 dataset-generator/dataset_generator.py gen dataset-generator/configs/HighFreqDataset.json --dest-folder .
/dataset
```

Of course, you can edit the `HighFreqDataset.json` file in the `configs` folder to customize your data generator. Here you can see an example of such a configuration:

```
{
    "seed": 42,
    "num_days": 365,
    "num_req_x_day": -1,
    "dest_folder": "HighFrequencyDataset",
    "function": {
        "function_name": "HighFrequencyDataset",
        "kwargs": {
            "num_files": 1000,
            "min_file_size": 1000,
            "max_file_size": 4000,
            "lambda_less_req_files": 1.0,
            "lambda_more_req_files": 10.0,
            "perc_more_req_files": 25.0,
            "perc_files_x_day": 1.0,
            "size_generator_function": "gen_random_sizes"
        }
    }
}
```

After the dataset creation, you will see the dataset files into the `dataset` folder in the main of the project.

Run the simulation

First, you need to compile the simulator:

```
# Compile the simulator
python3 -m utils compile --release --fast
```

Then, you can get the simulator executable with the following command:

```
# Get simulator exec path
export SIM=$(python3 -m utils sim-path)

# Check simulator executable is working
$SIM help
```

After that, you can run a simulation using the datatset previously generated. To do this, you need to create a proper simulation config file, like the following:

```
--- # Simulation parameters
sim:
  data: ./dataset
  outputFolder: ./results/
  type: normal
  window:
    start: 0
    stop: 52
  region: it
  overwrite: true
  cache:
    # Use Reinforcement learning AI
    type: aiRL
    watermarks: false
    # Create a cache with 10G size
    size:
      value: 10
      unit: G
    bandwidth:
      value: 1
      redirect: true
  ai:
    rl:
      epsilon:
        decay: 0.001
      addition:
        featuremap: ./smart-cache/featureMaps/rlAdditionFeatureMap.json
      eviction:
        featuremap: ./smart-cache/featureMaps/rlEvictionFeatureMap.json
```

Create the above config with the following command:

```
cat << EOF > simulation.conf.yaml
--- # Simulation parameters
sim:
  data: $(pwd)/dataset
  outputFolder: $(pwd)/results
  type: normal
  window:
    start: 0
    stop: 52
  region: it
  overwrite: true
  cache:
    # Use Reinforcement learning AI
    type: aiRL
    watermarks: false
    # Create a cache with 10G size
    size:
      value: 10
      unit: G
    bandwidth:
      value: 1
      redirect: true
  ai:
    rl:
      epsilon:
        decay: 0.001
      addition:
        featuremap: $(pwd)/smart-cache/featureMaps/rlAdditionFeatureMap.json
      eviction:
        featuremap: $(pwd)/smart-cache/featureMaps/rlEvictionFeatureMap.json
EOF
```

Finally, run the simulation with:

```
$SIM sim simulation.conf.yaml
```

Explore the results

The simulation results will be stored in a folder named `results/run_full_normal/aiRL_SCDL2-onK_10G_1Gbit_it/`, that may change based on the simulation configuration file. The folder contains a `.csv` file with the simulation results and and other files containing some simulation statistics. You can load these results using a `Python` library like `pandas` or you can examine them using the dashboard from the `Probe` module:

```
python3 -m probe.results dashboard results
```

Finally, the dashboard will be available at http://localhost:8050/ by default. If you need a specific ip for the dashboard, you can set the proper parameter (e. g. `-dash-ip 0.0.0.0`).

Annotated Description

N/A

References

- https://github.com/Cloud-PG/dataset-generator
- https://github.com/Cloud-PG/smart-cache

Attachments

N/A