How to Run Raha and Reproduce Its Experimental Results

Mohammad Mahdavi
TU Berlin
mahdavilahijani@tu-berlin.de

Ziawasch Abedjan
TU Berlin
abedjan@tu-berlin.de

Raul Castro Fernandez
MIT
raulcf@csail.mit.edu

Samuel Madden
MIT
madden@csail.mit.edu

Mourad Ouzzani
QCRI, HBKU
mouzzani@hbku.edu.qa

Michael Stonebraker
MIT
stonebraker@csail.mit.edu

Nan Tang
QCRI, HBKU
ntang@hbku.edu.qa

1 INTRODUCTION

Raha is an error detection system and has been recently published in the proceedings of ACM SIGMOD 2019 [1, 2]. In this document, we describe how to run Raha and reproduce the experiments described in the paper. In particular, we elaborate on the installation and benchmarking scripts that are publicly available online [3]. Following the reproducibility chair's guidance, we only provide experiments on 6 out of 8 datasets, because two of the datasets were proprietary and cannot be shared. The set of experiments and the corresponding results are however in line with the remaining datasets.

2 INSTALLATION

To install Raha, you need a Linux platform (preferably Debian/Ubuntu) with an already installed Python 3. The system hardware should have a multicore processor (preferably with 4+ cores) and the memory size of a common novel notebook (preferably 16+ GB). As mentioned in the original paper [1], we ran all the experiments on an Ubuntu 16.04 LTS machine with 28 2.60 GHz cores and 264 GB memory.

Raha, with its datasets and experiments, is available online at a GitHub repository [3]. To download and install it, you need to open a terminal and follow the following steps:

(1) **Cloning the GitHub repository.**
```
git clone https://github.com/BigDaMa/raha
```
(2) **Setting up the project.**
```
cd raha
sudo python3 setup.py install
```

Note that, in case that you want to uninstall the project at some point, you can easily uninstall it with the following command:

```
sudo pip3 uninstall raha
```

3 BENCHMARK

We have prepared one master script to run all the experiments. The script is named *benchmark.py* and it can be found in *raha/benchmark.py*. While it is possible to run all the experiments with just one click, this may be time consuming. The reason is that the time complexity of running all the experiments at once is *O(ERDV)*, where E is the number of experiments, R is the number of experiment repetitions for reporting the average of results, D is the number of datasets, and V is the number of different versions of our system that need to be evaluated in a particular experiment. The value of these variables in our experiments are as follows:

- $E = 7$ **experiments.** We have $E = 7$ experiments that are presented in Sections 6.2 to 6.6 and Appendices B.1 and B.2 of the original research paper [1].
- $R = 10$ **independent runs.** We reported the results of each experiment as the average of $R = 10$ independent runs of each experiment.
- $D = 8$ **datasets.** We have $D = 8$ datasets in our experimental setting. Note that since 2 out of our 8 datasets are proprietary, where we signed non-disclosure agreements to be able to use them, we cannot release these 2 datasets.
- $V \approx 5$ **versions.** We have on average of around $V \approx 5$ different versions of our system in each particular experiment that need to be evaluated.

Therefore, running all the experiments at once means to run our system $O(ERDV) \approx E \times R \times D \times V \approx 7 \times 10 \times 8 \times 5 = 2800$ times, which is prohibitive.

To make the reproducibility of our experimental results easier for the reviewers, we provide three modifications. First, we provide the option of running one experiment at a time while omitting the number of experiment E from the time

```
Comparison with the stand-alone error detection tools. (Precision, recall, f1 score)
+-------------+------------------+------------------+------------------+------------------+------------------+
|   Approach  |     hospital     |     flights      |      beers       |      rayyan      |     movies_1     |
+-------------+------------------+------------------+------------------+------------------+------------------+
|    dBoost   | 0.41, 0.32, 0.36 | 0.76, 0.58, 0.65 | 0.59, 1.00, 0.75 | 0.15, 0.84, 0.25 | 0.25, 0.79, 0.38 |
|    NADEEF   | 0.05, 0.37, 0.09 | 0.42, 0.93, 0.58 | 0.13, 0.06, 0.08 | 0.30, 0.85, 0.44 | 0.80, 0.80, 0.80 |
|    KATARA   | 0.06, 0.37, 0.10 | 0.07, 0.10, 0.08 | 0.08, 0.26, 0.12 | 0.02, 0.10, 0.03 | 0.01, 0.03, 0.02 |
| ActiveClean | 0.03, 0.20, 0.05 | 0.30, 1.00, 0.46 | 0.16, 1.00, 0.28 | 0.09, 1.00, 0.16 | 0.06, 1.00, 0.12 |
|     Raha    | 0.81, 0.63, 0.70 | 0.79, 0.88, 0.83 | 0.99, 1.00, 0.99 | 0.82, 0.74, 0.77 | 0.84, 0.84, 0.84 |
+-------------+------------------+------------------+------------------+------------------+------------------+
```

Figure 1: An example output of an experiment in the form of a table.

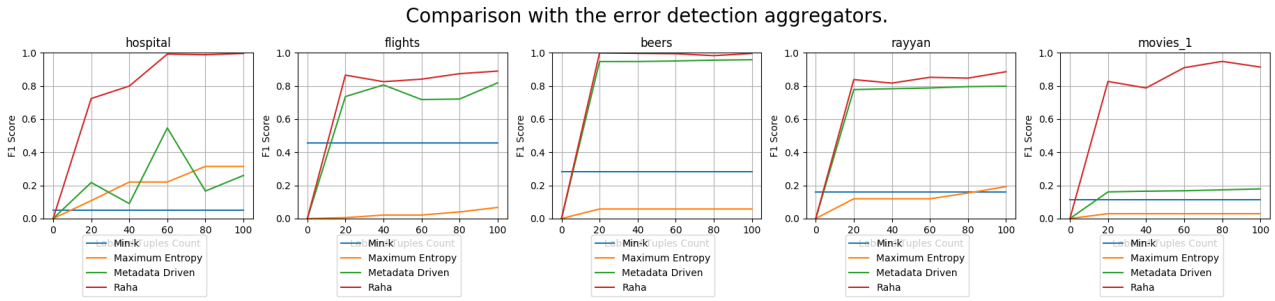

Figure 2: An example output of an experiment in the form of charts.

complexity. Thus, you can run each experiment separately with the command

```
python3 benchmark.py n
```

where *n* is the name of the experiment that you want to run, i.e., 1, 2, . . . , 7. For example, the command

```
python3 benchmark.py 1
```

runs only the first experiment. Second, we also provide the option of running one experiment only once while omitting the number of independent runs R from the time complexity. Thus, you can run one experiment only one time (instead of 10 independent runs) with the command

```
python3 benchmark.py n fast
```

where *fast* is an input argument that reduces the number of independent runs of an experiment from 10 runs to 1 run. Although the resulting numbers would not be accurate and may not match the original research paper's result numbers, this option enables the reviewers to observe the output of one run of an experiment quickly. For example,

```
python3 benchmark.py 1 fast
```

runs the first experiment only once. Therefore, these relaxations reduce the aforementioned time complexity from $O(ERDV) \approx 2800$ to $O(DV) \approx 40$. Third, we also store intermediate results for later experiments except those that measure the efficiency of the system. That is why we strongly recommend to run experiment 1 first to generate some intermediate results for the rest of the experiments. After running experiment 1, the rest of the effectiveness experiments can be run in any order.

Each experiment should finish in less than half an hour if run in the *fast* mode. The only exception is experiment 6. Since this experiment tests scalability, we have to run Raha on a large dataset without storing any intermediate results to measure the actual runtime. That is why this experiment will take a couple of hours and we strongly recommend to run it in the *fast* mode, i.e., run the command `python3 benchmark.py 6 fast`.

The results of each experiment will be printed in the standard output in the form of tables, as Figure 1 shows an example. Furthermore, the charts of the original research paper will be redrawn, as Figure 2 shows an example.

Custom benchmark. To benchmark Raha on your own dataset, you need the dirty dataset itself and its corresponding ground truth in CSV format. The file *raha/detection.py* contains a simple example for running Raha on such new datasets.

REFERENCES

[1] Mohammad Mahdavi, Ziawasch Abedjan, Raul Castro Fernandez, Samuel Madden, Mourad Ouzzani, Michael Stonebraker, and Nan Tang. 2019. Raha: A configuration-free error detection system. In *SIGMOD*. 865–882.

[2] Mohammad Mahdavi, Ziawasch Abedjan, Raul Castro Fernandez, Samuel Madden, Mourad Ouzzani, Michael Stonebraker, and Nan Tang. 2019. Raha's Page in the ACM Digital Library. https://dl.acm.org/citation.cfm?id=3324956. Accessed: 22.11.2019.

[3] Mohammad Mahdavi, Ziawasch Abedjan, Raul Castro Fernandez, Samuel Madden, Mourad Ouzzani, Michael Stonebraker, and Nan Tang. 2019. Raha's Repository in GitHub. https://github.com/BigDaMa/raha. Accessed: 22.11.2019.