TEEBench: How to Reproduce the Experimental Results

Kajetan Maliszewski
TU Berlin
maliszewski@tu-berlin.de

Jorge-Arnulfo Quiané-Ruiz
TU Berlin
jorge.quiane@tu-berlin.de

Jonas Traub
TU Berlin
jonas.traub@tu-berlin.de

Volker Markl
TU Berlin
volker.markl@tu-berlin.de

1 INTRODUCTION

TEEbench is a unified benchmarking framework for relational operators in Trusted Execution Environments (TEEs). We recently published TEEbench in PVLDB [3]. In this report, we describe how to run the framework and reproduce all experimental results from the paper. First, we describe the installation process (Section 2). Second, we provide instructions on reproducing the experimental results (i.e., all relevant figures and tables from the paper) in Section 3. All materials used in experiments (i.e., source code, scripts, and datasets) are publicly available online [2, 4].

2 INSTALLATION

TEEbench uses modern hardware features to run inside TEEs. Therefore, the installation process can seem complex and tricky. In this section, we try to simplify the process and describe all the necessary steps to successfully compile and run the source code.

The installation process consists of a few steps:

(1) Prepare hardware
(2) Download the source code
(3) Install dependencies
(4) Compile and run the source code

The following paragraphs explain each step in detail.

(1) Prepare hardware. To use TEEbench, you need a bare-metal machine with Ubuntu 18.04 OS and an Intel CPU with SGX support. SGX functionality must be enabled in BIOS. To enable it, you can use an application provided by Intel [1]. Follow the instructions in [1] to configure BIOS correctly.

As described in the paper [3], we used a machine with an Intel Core i7-8565U CPU, 38 GB of main memory, and Ubuntu 18.04.02 OS. While current CPUs support both 128 MB and 256 MB EPC, our machine supported 128 MB EPC. This is an important parameter as it will often be a performance threshold.

Additionally, for experiments in Section 8 in the paper (*Other Hardware Platforms*), we used two Google Cloud VMs to run experiments with AMD SEV. These machines use VM-based TEEs and do not require a complicated setup procedure. They only need Ubuntu OS installed and the SEV feature enabled/disabled.

(2) Download the source code. The source code with all datasets is available on the university cloud [4]. This version includes the IMDb dataset used in some experiments. Additionally, we published the source code in a public Github repository [2]. However, the repository in [2] does not contain the dataset due to space restrictions. We, therefore, recommend downloading the source code from the link in [4]. Unzip the archive to a preferred location.

(3) Install dependencies. This refers only to the SGX machines. We provide an installation script under *scripts/install.sh*. The script performs four steps. It installs (i) Linux packages, (ii) custom SGX driver, (iii) SGX-SDK, and (iv) Microsoft SEAL library. You will need sudo access to successfully execute all the steps, however, execute the script without the sudo command. Execute the following command to run the script:

```
$ cd scripts
$ ./install.sh
```

Bear in mind that this script does low-level configuration (e.g., the kernel driver, system libraries). Many elements of this configuration are not part of our contributions. Therefore, we can not guarantee that all hardware will be compatible with the required software and that the script will work on all machines. The script we provide is a courtesy towards the reviewers and everyone willing to run TEEbench to ease the process of reproducibility. We hope that the script will work out of the box and we are glad to support the installation process in case of any issues.

(4) Compile and run the source code. To check if the code compiles and runs, run the *scripts/compile.sh* script. If the code compiles and executes successfully, you should see a sample run of the RHO join with multiple measurements. Execute the following command to run the script:

```
$ cd scripts
$ ./compile.sh
```

3 BENCHMARK

We provide a master script to run all experiments. The script can be found in *scripts/benchmark.py*. Section 3.1 contains a short guide to using the *benchmark.py* script. Although it is possible to run all experiments at once, we do not recommend it. Executing all experiments would require a significant amount of time (i.e., a few days). The paper contains the results of eighteen independent experiments. In many of these experiments, we evaluate the performance of eight different join algorithms for two test datasets. As mentioned in the paper [3], we take all measurements five times and average the results. While some algorithms perform well, some encounter severe bottlenecks and may need even up to tens of minutes to perform a join operation (e.g., PHT join for dataset *cache-exceed* triggers EPC thrashing). This may lead to prohibitive time and resource consumption.

Therefore, we provide several enhancements to make reproducibility easier. First, we provide the option to run only one experiment. Second, we provide the option to run one experiment

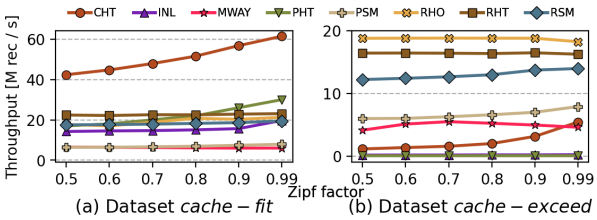

Figure 1: Example figure with experimental results.

with a reduced number of independent runs from the default 5 to 1. This second option provides fewer quality results that can have more uncertainty. Nonetheless, it allows generating figures with good-enough results often within minutes. Using this feature, each experiment (except Figure 8) should finish in less than half an hour. Section 3.1 contains the details on using both options.

The *benchmark.py* script runs the experiment, saves the data to the *scripts/data* directory, and automatically generates figures. All the figures are saved in the *scripts/img* directory. The script outputs the name of the saved file, e.g.:

```
Saved image file: ../img/Figure-13-Throughput-when-
    varying-join-selectivity.png
```

Figure 1 shows example results.

Note that Table 3 in the paper is the only table that contains experimental results. To perform this experiment run the benchmark script with the *t3* option. The results will be printed in the command line window and will be saved under *scripts/img/Table-03-Hardware-performance-counters-for-secure-joins.csv*.

3.1 How to use *benchmark.py*

To run all experiments at once (not recommended - see Section 3):

```
$ ./benchmark.py all
```

To run a specific experiment, first, identify its ID by finding the relevant figure or table in the paper. The ID is created as follows. IDs for figures start with *f* and follow with the number of the figure from the paper [3]. For example, Figure 14 has ID **f14**. IDs for tables start with *t* and follow with the number of the table from the paper. For example, Table 3 has ID **t3**. Once you identified its ID (e.g., f14), run the script using the following command:

```
$ ./benchmark.py f14
```

To run a specific experiment with a reduced number of repetitions add *fast* option at the end of the command:

```
$ ./benchmark.py f14 fast
```

You can also execute all experiments with *fast*:

```
$ ./benchmark.py all fast
```

As previously mentioned, this reduces the number of runs from five to one. For many experiments, it decreases the execution time from hours to minutes. You can also specify the number of runs by providing any positive integer as the second argument. For instance, to generate Figure 14 with three repetitions, run:

```
$ ./benchmark.py f14 3
```

3.2 Important notes on some experiments

- Results from Table 3 in the paper (hardware performance counters) use the PCM tool to collect measurements from the CPU. Therefore, this experiment requires to execute with sudo access.
- Figure 8 from the paper scales up the test dataset. As mentioned in the paper, this experiment was difficult to perform because of the unexpected behavior of SGX enclaves when disk swapping occurred. Moreover, in this experiment, we use 32GB enclaves, which take a significant amount of time to set up. The cost multiples as each run creates a new enclave. Therefore, this experiment takes long hours to complete even when using the *fast* option. In fact, we performed some measurements manually, added them to the CSV file with results (*scripts/data/throughput-scale-input-output.csv*), and generated the final figure. In case, one would like to generate only the figure from a CSV file without running the experiment, she can go to file *scripts/helpers/config.yaml*, change the *experiment* parameter to 0, and re-run the benchmark script. We recommend reverting the parameter to its default value (i.e., 1) after generating the figure as it is used by other scripts.
- Figures 17 and 19 from the paper require using a machine with an AMD CPU. This requires the orchestration of AMD virtual machines and, therefore, exceeds the scope of automation we aim at. We provide thorough instructions on how to manually perform these experiments instead of automated scripts. In the paper, we used two Google Cloud VMs to perform these experiments. We recommend following the same approach. For detailed instructions execute the following commands:

  ```
  $ ./benchmark.py f17
  $ ./benchmark.py f19
  ```

REFERENCES

[1] Github. 2022. *Intel(R) Software Guard Extensions Software Enabling Application for Linux.* Retrieved May 1, 2022 from https://github.com/intel/sgx-software-enable
[2] Github. 2022. *TEE Relational Operator Benchmark Suite .* Retrieved May 1, 2022 from https://github.com/agora-ecosystem/tee-bench
[3] Kajetan Maliszewski, Jorge-Arnulfo Quiané-Ruiz, Jonas Traub, and Volker Markl. 2021. What is the price for joining securely? benchmarking equi-joins in trusted execution environments. *Proceedings of the VLDB Endowment* 15, 3 (2021), 659–672.
[4] tubCloud. 2022. *tee-bench.zip.* Retrieved May 31, 2022 from https://tubcloud.tu-berlin.de/s/KKXjkkS6GwD5pwn