Abstract

This artifact contains all the source codes and experimental data necessary to reproduce our evaluation in Section 6, excluding the results of SeqC. In particular, source codes of the tools SPDOffline and SPDOnline as well as all the benchmarks used in our evaluations are provided. The compared techniques Dirk [Kalhauge and Palsberg 2018], and DeadlockFuzzer [Joshi et al. 2009] are also included. Python scripts that fully automate the process of reproducing our evaluation are provided with the artifact.

Checklist

- **Algorithm:** SPDOnline, SPDOffline.
- **Program:** Extension of Rapid [Mathur 2019] with SPDOffline, extension of DeadlockFuzzer [Joshi et al. 2009] with SPDOnline and Java benchmarks.
- **Run-time environment:** Docker.
- **Metrics:** Execution time, and number of bugs found.
- **Output:** CSV files.
- **Experiments:** Scripts that fully automate our workflow are provided.
- **How much disk space required (approximately)?:** ~40GB.
- **How much time is needed to complete experiments (approximately)?:** Reproducing all the results: ~30 hours (without parallelization). We also provide instructions for reducing the timeout durations and reproducing only a subset of our results.
- **Publicly available?:** Yes.
- **Code licenses (if publicly available)?:** MIT License.
- **Archived (provide DOI)?:** doi.org/10.5281/zenodo.7794125.

Description

Hardware dependencies

Replicating the results of certain large benchmarks requires at least 16 GB RAM. Moreover, we remark that users may encounter problems in running the artifact on Apple M1 silicon due to certain incompatibility issues in running Docker on these machines. There are otherwise no special hardware requirements.

Software dependencies

The artifact requires a Docker installation.

Getting Started Guide

- Install Docker (https://www.docker.com).
- Obtain the artifact's docker image from doi.org/10.5281/zenodo.7794125.
- Import the image:

  ```
  docker import pldi23-578.docker pldi23-578
  ```

- Start a container:

  ```
  docker run -it pldi23-578 bash
  ```

- Run basic tests:

  ```
  python3 /root/table1/run.py -b Bensalem

  cd /root/table2/calfuzzer/ ; python3 /root/table2/calfuzzer/run.py -b Bensalem -i 1
  ```

- Compile the results:

  ```
  python3 /root/table1/compile_results.py

  python3 /root/table2/calfuzzer/compile_results.py
  ```

- If the installation was successful, the scripts will produce the following CSV files `/root/table1/out.csv` and `/root/table2/calfuzzer/out.csv` with relevant statistics.

  The outputs of these files can be displayed in the console as follows:

  ```
  csvtool readable /root/table1/out.csv

  csvtool readable /root/table2/calfuzzer/out.csv
  ```

Experiment workflow

Below we display the directory structure of the artifact. The two main directories are `table1` and `table2/calfuzzer`. Both of these directories are equipped with the scripts `run.py` and `compile_results.py` where the former is used to conduct the experiments and the latter is used to compile all the results into a CSV file.

```
├── root/
│   ├── table1/
│   │   ├── dirk/
│   │   ├── spdoffline/
│   │   ├── trace_conversion/
│   │   ├── compile_results.py
│   │   ├── convert.py
│   │   ├── expected_results.csv
│   │   ├── run.py
│   ├── table2/
│   │   ├── calfuzzer/
│   │   │   ├── ...
│   │   │   ├── compile_results.py
│   │   │   ├── run.py
│   ├── README.md
│   ├── record_trace.sh
```

List of claims

The main evaluation claims of the paper are as follows.

- **Section 6.1, Offline Experiments (Table 1).** SPDOffline outperforms Dirk and SeqC in terms of runtime performance and displays a comparable performance with SeqC in terms of the total number of bugs found on the given benchmark traces.
- **Section 6.2, Online Experiments (Table 2).** SPDOnline outperforms DeadlockFuzzer in terms of total number of bug hits and unique bugs found.

Evaluation and expected result

Reproducing Table 1

The following command will run Dirk and SPDOffline on all the benchmarks in Table 1 with 1 hour timeout per run.

```
python3 /root/table1/run.py
```

Note that the above procedure is expected to take ~15 hours on default settings. We refer the interested reader to the experiment customization section for instructions on reducing the timeout duration and/or performing the analysis on a select set of benchmarks. Once the experiments are finished, all the raw output logs will be located under the directory `/root/table1/outfiles`. Running the following script will process these outputs and produce a CSV file.

```
python3 /root/table1/compile_results.py
```

The default path of the generated CSV file is `/root/table1/out.csv`. The expected results (without the running times) can be found in the file `/root/table1/expected_results.csv`. There are minor inconsistencies between these results as compared to our paper. These will be fixed in the final version of our paper.

Reproducing Table 2

The following command will run DeadlockFuzzer and SPDOnline on all the benchmarks in Table 2 with 50 iterations.

```
cd /root/table2/calfuzzer/ ; python3 /root/table2/calfuzzer/run.py
```

Note that the above procedure is expected to take ~15 hours on default settings. We refer the interested reader to the experiment customization section for instructions on reducing the number of iterations and/or performing the analysis on a select set of benchmarks. Once the experiments are finished, all the raw output logs will be located under the directory `/root/table2/calfuzzer/outfiles`. Running the following script will process these outputs and produce a CSV file.

```
python3 /root/table2/calfuzzer/compile_results.py
```

The default path of the generated CSV file is `/root/table2/calfuzzer/out.csv`.

Experiment customization

Running SPDOnline on new benchmarks

Integrating a new benchmark into our experimental workflow for SPDOnline requires modifying the file `/root/table2/calfuzzer/run_stable.xml`. This file must be extended with the following information:

```
 <target name="BENCHMARK-NAME_analysis">
    <property name="javato.work.dir" value="..."/>
    <property name="javato.app.class.path" value="..."/>
    <property name="javato.app.main.class" value="..."/>
    <antcall target="analysis-run"/>
</target>

<target name="BENCHMARK-NAME_instr">
    <property name="javato.work.dir" value="..."/>
    <property name="javato.app.class.path" value="..."/>
    <property name="javato.app.main.class" value="..."/>
    <antcall target="instr-run"/>
</target>
```

Here, `BENCHMARK-NAME` is used as an alias for the name of the benchmark and `javato.work.dir`, `javato.app.class.path`, and `javato.app.main.class` correspond to the working directory, class path and main class of the new benchmark, respectively. The value fields of these properties must be updated accordingly. Here, working directory should be set to the directory where the new benchmark is located. Certain auxiliary files will be created under this directory. The fields `javato.app.class.path` and `javato.app.main.class` should be set in the same way as Java VM classpath and main class. In addition, `javato.app.args` field may be set to pass arguments to the underlying program. After this step, the only remaining expected modification is on the `get_benchmarks` function in the script `/root/table/calfuzzer/run.py`. This function returns a list of tuples of shape `(BENCHMARK-NAME, TIMEOUT (in seconds))`. Users may extend this list with the new benchmark. Note that the `BENCHMARK-NAME` must match with the name supplied in the `/root/table2/calfuzzer/run_stable.xml` file. After this step, the script `/root/table2/calfuzzer/run.py` can be utilized the same way as before.

Running SPDOffline on new benchmarks

SPDOffline takes as input traces in RapidBin format. Recall that a trace is a sequence of events, each of which can be seen as a record `e = <t, op(decor), loc>`, where

- `t` is the identifier of the thread that performs `e`
- `op(decor)` represents the operation along with the operand. `op` can be one of `r` / `w` (read from/ write to a memory location), `acq` / `req` / `rel` (acquire/request/release of a lock object), `fork` / `join` (fork or join of a thread). The field `decor` is the corresponding memory location when `op` is `r` / `w`, lock identifier when `op` is `acq` / `req` / `rel` and thread identifier when `op` is `fork` / `join`.
- `loc` represents the program line number that resulted in the generation of the event `e`.

The RapidBin file format essentially represents a trace as a sequence of records, where the above three fields are stored as 8-byte values. A human-readable version of a trace in RapidBin format looks as follows:

```
 T0|w(V123)|345
T0|fork(T1)|123
T1|w(V234.23[0])|456
T0|r(V123)|345
T0|fork(T2)|123
T2|req(L34)|120
T2|acq(L34)|120
T2|rel(L34)|130
T0|join(T2)|134
T0|join(T1)|134
```

We refer the interested reader to `/root/table1/spdoffline/src/parse/bin/BinaryFormat.java` for more details in the encoding of RapidBin format.

New benchmark traces can be integrated into our experimental workflow for SPDOffline by simply placing the zipped version of the execution trace in RapidBin format under the folder `/root/table1/spdoffline/benchmarks`. Below we describe two procedures for obtaining new traces in RapidBin format.

Converting existing traces from different formats

We provide the script `/root/table1/convert.py` for converting from common trace formats (e.g., RoadRunner, RVPredict, STD) into the RapidBin format. This script can be utilized as follows:

```
   python3 /root/table1/convert.py -i <input_file> -f <input_format> -o <output_file>
```

For example, the command below converts the trace file `/root/table1/trace_conversion/sample.std` in STD format into RapidBin format and generates the file `/root/table1/trace_conversion/sample.data`

```
python3 /root/table1/convert.py -i /root/table1/trace_conversion/sample.std -f std -o /root/table1/trace_conversion/sample.data
```

After the trace has been converted into the RapidBin format and the zipped file is placed under `/root/table1/spdoffline/benchmarks`, then the script `/root/table1/run.py` can be utilized the same way as before to run SPDOffline.

Generating new traces

Below we describe a procedure for generating new traces by utilizing the DeadlockFuzzer tool. The below command generates a new trace in STD format.

```
cd /root/table2/calfuzzer/ ; python3 /root/table2/calfuzzer/run.py -b <benchmark_name> -r
```

where `benchmark_name` is the name of the benchmark as specified in the `/root/table2/calfuzzer/run_stable.xml` file. Please refer to the `Running SPDOnline on new benchmarks` for adding new benchmarks into this file. The generated trace can be found under the folder `/root/table2/calfuzzer/outfiles/$1/PrintTrace`. After this step, the instructions provided under `Converting existing traces from different formats` can be utilized to convert the trace from STD format into RapidBin format. We provide the script `/root/record_trace.sh` to fully automate this process. Following command can be utilized to generate new traces.

```
 sh /root/record_trace.sh <benchmark_name>
```

The above command generates a trace in RapidBin format, and places it under the folder `/root/table1/spdoffline/benchmarks/`. The name given to the generated trace is of following format: `benchmark_name_dlf`. For example, a new trace for the benchmark `Bensalem` can be generated as follows:

```
sh /root/record_trace.sh Bensalem
```

This generates a trace with the name `Bensalem_dlf`. SPDOffline can be run on the new generated benchmark trace as follows:

```
python3 /root/table1/run.py -b Bensalem_dlf
```

Reducing timeout

In the experiments for reproducing Table 1, the timeout duration for each run can be altered by providing the argument `-t` to the script. For instance, the following sets the timeout to 10 minutes for each run:

```
python3 /root/table1/run.py -t 10m
```

Reducing iterations

In the experiments for reproducing Table 2, the number of iterations for each benchmark can be altered by providing the argument `-i` to the script. For instance, the following sets the number of iterations to 3 for each benchmark:

```
python3 /root/table2/calfuzzer/run.py -i 3
```

Selecting benchmarks

The scripts `/root/table1/run.py` and `/root/table2/calfuzzer/run.py` both support running the corresponding analyses on a select set of benchmarks. This can be achieved by providing the argument `-b` and specifying the names of the focal benchmarks. For instance, the following runs the analysis on the benchmarks `Deadlock`, `StringBuffer`, and `Account`.

```
python3 /root/table1/run.py -b Deadlock,StringBuffer,Account
```

Notes

- We do not have the right to distribute the SeqC tool [Cai et al. 2021], hence it is not contained in our artifact.
- The Swing benchmark in Table 2 cannot be executed in the Docker environment, hence it has been excluded from our artifact.
- A number of benchmark traces of Dirk have been provided outside the container as they are large in size. These traces can be downloaded separately. Simply placing these traces under the folder `/root/table1/dirk/benchmarks` will include them in our experimental workflow.