#60 Overview of Artifact

Zhiqiang Zuo, Lu Fang, Guoqing (Harry) Xu, Shan Lu
University of California, Irvine
University of Chicago

1 Description

We put the artifact in a virtual machine managed by Oracle VM VirtualBox. Users can download the image file (OOPSLA_2016_60.tgz), untar it by running the following command `tar -xzvf OOPSLA_2016_60.tgz`, and then import OOPSLA_2016_60.vdi file into your VirtualBox client. Note that the operating system of our virtual machine is Linux Fedora (64-bit).

The artifact consists of two parts: namely a single PC implementation and a distributed implementation. In the Getting Started Guide, we only consider the single PC environment. The distributed part will be discussed in the Step-by-Step Instructions.

In the single PC environment, our experiments contain 9 subjects, each of which consists of several faulty versions. All the subjects are located at the following directory: `/home/paper_60/oopsla_artifacts/single/Subjects/`. All our raw data and scripts for the distributed environment are located under the directory: `/home/paper_60/oopsla_artifacts/distributed/`. In addition, we put all our original experimental data under the directory: `/home/paper_60/original_experimental_data/`.

2 Getting Started Guide

In this started guide, we will take one subject (i.e., sed) as example. Run the following instructions for subject **sed** to test and explore the key elements of our artifact. Note that it may take several minutes to finish, especially Step 4 which requires running the instrumented programs multiple times. Moreover, since the program under analysis could be buggy, it may print out error or warning messages during running. Please just ignore it.

1. First run a script which instruments the buggy programs and runs tests on the instrumented programs to obtain traces.
   ```
   cd /home/paper_60/oopsla_artifacts/single/Subjects/C/sed/scripts/
   sh runAll_inst.sh
   ```

2. Analyze the above generated traces to count how many predicates our approach need to instrument, which corresponds to Q1 in the paper.
   ```
   java -jar ~/bin/HI.jar ~/oopsla_artifacts/single/Subjects/C/ sed
   ~/oopsla_artifacts/single/Console/
   ```
   The result will be exported into an excel file (sed_m.xlsx) located at:
   `/home/paper_60/oopsla_artifacts/single/Console/sed/`
   The percentage data shown by Figure 3 in our paper corresponds to Column AR (top-1), BH (top-3), BX (top-5), CN (top-10) in the above output excel file. We calculate the average across multiple versions for each subject.

3. Analyze the traces to obtain the performance results for different types of predicates, which corresponds to Q2 in the paper. In order to consider *branch*, *return*, and *scalar_pair* predicates, run the following commands respectively:

```
java -jar ~/bin/HI_branch.jar ~/oopsla_artifacts/single/Subjects/C/ sed
~/oopsla_artifacts/single/Console/
```

```
java -jar ~/bin/HI_return.jar ~/oopsla_artifacts/single/Subjects/C/ sed
~/oopsla_artifacts/single/Console/
```

```
java -jar ~/bin/HI_scalar.jar ~/oopsla_artifacts/single/Subjects/C/ sed
~/oopsla_artifacts/single/Console/
```

The results will be exported into excel files (sed_m.xlsx) located at
`~/oopsla_artifacts/single/Console/sed_branch/`,
`~/oopsla_artifacts/single/Console/sed_return/`,
`~/oopsla_artifacts/single/Console/sed_scalar/`, respectively.
The data shown by Table 3 in our paper corresponds to Column BX (Percentage of predicates), and BY (Number of Iterations) in the above output excel file. We calculate the average across multiple versions for each subject.

4. In order to assess the user-side overhead, first run the script which runs tests on the respective programs to obtain the overhead data.

```
cd /home/paper_60/oopsla_artifacts/single/Subjects/C/sed/scripts/
sh runAll_overhead.sh
```

Then execute the following command to collect all the overhead data and export them into a single excel file (sed_overhead.xlsx) located at the console folder.

```
java -cp /home/paper_60/bin/HI.jar zuo.util.readfile.IterativeTimeReader
~/oopsla_artifacts/single/Subjects/C/ sed ~/oopsla_artifacts/single/Console/
```

The user-side overhead data shown by Table 4 in our paper is calculated from the respective column divided by Column "outputs" in the above excel file.

3 Step by Step Instructions

In the following, we will explain running instructions step by step to reproduce our entire experiments. In the single PC environment, we list the instructions needed to validate research questions Q1, Q2, Q3 and Q4 in our paper. Later, we discuss the raw data and instructions for Q5 in the distributed environment.

3.1 Single PC

We have 9 subjects in our experiments. Users can run the following instructions for each subject one by one to reproduce our entire experiments. Since we have to run multiple instrumented programs over many test cases multiple times for comparison, it may take hours to finish the whole process especially for large subjects (e.g., bash, grep, ant, derby) or small subjects with a lot of sleeps in the test scripts (e.g., siena). For Q1 and Q2, it requires large memory for analyzing traces of large subjects (i.e., derby, bash, grep). If no enough memory in the virtual machine, users can export the subject folder to a machine with big memory after running the scripts in Step 1 (which must be run on our virtual machine due to the required instrumentation library). Then run Steps 2 and 3 on the new machine by specifying the respective parameters (i.e., subject-root-dir subject-name output-dir).

Q1 & Q2

1. First execute the following command to run a script which instruments the buggy programs and runs tests on the instrumented programs to obtain traces.

```
cd subject-script-dir
sh runAll_inst.sh
```
Taking nanoxml as an example, run the following command
```
cd /home/paper_60/oopsla_artifacts/single/Subjects/Java/nanoxml/scripts/
sh runAll_inst.sh
```

2. (Q1) Analyze the above generated traces to count how many predicates our approach needs to instrument, which corresponds to Q1 in the paper.
```
java -jar ~/bin/HI.jar subject-root-dir subject-name output-dir
```
For example, like nanoxml, run the following command
```
java -jar ~/bin/HI.jar ~/oopsla_artifacts/single/Subjects/Java/ nanoxml
~/oopsla_artifacts/single/Console/
```
The result will be exported into an excel file (nanoxml_m.xlsx) located at
`~/oopsla_artifacts/single/Console/nanoxml/`

3. (Q2) Analyze the traces to obtain the performance results for different types of predicates, which corresponds to Q2 in the paper.
In order to consider *branch*, *return*, and *scalar_pair* predicates, run the following commands respectively:
```
java -jar ~/bin/HI_branch.jar subject-root-dir subject-name output-dir
```
```
java -jar ~/bin/HI_return.jar subject-root-dir subject-name output-dir
```
```
java -jar ~/bin/HI_scalar.jar subject-root-dir subject-name output-dir
```
For example, run the following command to compute the result for subject nanoxml in terms of *branch* predicates only.
```
java -jar ~/bin/HI_branch.jar ~/oopsla_artifacts/single/Subjects/Java/ nanoxml
~/oopsla_artifacts/single/Console/
```
The results will be exported into excel files (nanoxml_m.xlsx) located at
`~/oopsla_artifacts/single/Console/nanoxml_branch/`.

Q3

1. (Q3) In order to assess the user-side overhead, first execute the following command to run the script running tests on the respective programs (original program without instrumentation, sampling-based instrumented program, iteratively instrumented program) to obtain the overhead data.
```
cd subject-script-dir
sh runAll_overhead.sh
```
For example, like nanoxml, run the following command
```
cd /home/paper_60/oopsla_artifacts/single/Subjects/Java/nanoxml/scripts/
sh runAll_overhead.sh
```
Then execute the following command to collect all the overhead data and export them into a single excel file.
```
java -cp /home/paper_60/bin/HI.jar zuo.util.readfile.IterativeTimeReader subject-root-
dir subject-name output-dir
```
For nanoxml:
```
java -cp /home/paper_60/bin/HI.jar zuo.util.readfile.IterativeTimeReader
~/oopsla_artifacts/single/Subjects/Java/ nanoxml ~/oopsla_artifacts/single/Console/
```
The resulting excel file is nanoxml_overhead.xlsx which is located under the output directory (i.e., `~/oopsla_artifacts/single/Console/`).

Q4

1. (Q4) Run the following command to get the data plotted in Figure 4 of our paper, which actually represents how the *Importance* value of the top predicate changes as more iterations are performed. Note that for Q4, we only consider subject **space**.

```
cd ~/oopsla_artifacts/single/Subjects/C/space/scripts/
sh runAll_inst.sh
java -cp ~/bin/HI.jar zuo.processor.functionentry.client.iterative.Client_Ranking
~/oopsla_artifacts/single/Subjects/C/ space ~/oopsla_artifacts/single/Console/
```

   The data shown in Figure 4 of our paper is from the sheet named "space" of the output excel file (`~/oopsla_artifacts/single/Console/space_correlation.xlsx`). As for the data of ABI approaches appeared in Figure 4, we directly acquired the data from the original authors. Users can check the raw data in the following file:
   `/home/paper_60/original_experimental_data/space_score.dat`.

3.2 Distributed Environment

We have developed a central controller which automates the experiments on a distributed environment, which consists of 1 master node and 12 slave nodes connected via an InfiniBand network. According to our experience, setting up the runnable environment is nontrivial and time-consuming. There are a lot of variables and parameters to set. To reproduce our experimental results, users should have a cluster with similar configurations as our cluster described in the paper. Besides, running the whole experiments takes time (more than a day). Taking into account the feasibility for artifact evaluation, we provide the raw data and scripts to help the reviewers understand and validate our experimental data on the distributed environment at the first place. We are also pleased to provide the central controller and the respective configuration information to users later if anyone is interested in running the distributed version.

Q5

All our raw data and scripts are located under the directory: `/home/paper_60/oopsla_artifacts/distributed/`. Specifically,

- The collected traces (transfered from slaves to master) are included in the "raw_data/trace" folder.
- The distributed binaries (transfered from master to slaves) are included in the "raw_data/binary" folder.
- A script "scripts/analyze_trace.sh", which calculates the trace size and profile number. User can run the following command and the results will be printed in the console.

```
cd /home/paper_60/oopsla_artifacts/distributed/scripts/
sh analyze_trace.sh
```

- A script "scripts/analyze_binary.sh", which calculates the size of the distributed binaries.

Note that when we calcuated the size of the transferred data, we ignored the unnecessary fragments in files, since these fragments would not be transferred through network. That's why our script uses our own 'SizeCalculator' instead of 'du' or other linux commands.