Artifact - Collecting cyclic garbage across foreign function interfaces

1 Overview

This document mentions how to reproduce our experiments described in our paper "Collecting cyclic garbage across foreign function interfaces". Our experiment aims to measure performance overhead caused by *Refgraph GC*, our proposing garbage collection for reclaiming garbage objects that are connected through cyclic remote references. This artifact allows you to reproduce all the graphs presented in the experiments section of our paper. Those graphs illustrate that our garbage collection reclaims cyclic garbage across a language boundary.

Figure 1 shows a sketch of the system structure for the experiment. The experiment runs benchmark programs on a customized Ruby VM, ruby 3.1.2-clang-refgraph. It is an instance of the Ruby 3.1.2 VM supporting our Refgraph GC. The output of the experiment is a number of pdf files for the graphs shown in our paper. We below present how to build this VM and run experiments. We assume that you use Ubuntu 22.04 (or macOS 13 Ventura) or a Docker container.

2 Installation

First, you must download `ruby-refgraph-artifact.zip` from our Zenodo repository. This archive contains following files.

- Source programs of ruby 3.1.2-clang-refgraph
- Benchmark programs for our experiment
- A Python Script to run benchmarks and generate figures
- A Dockerfile

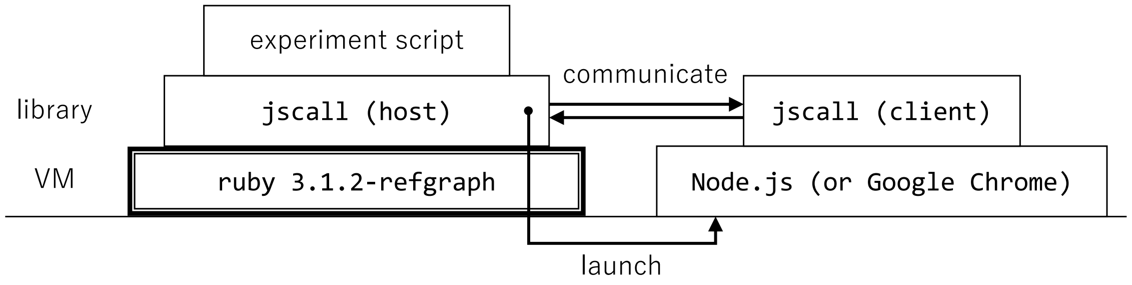

Figure 1: System overview.

Then, we build our customized version of the Ruby VM, ruby 3.1.2-clang-refgraph. We also install node.js as a JavaScript engine and the jscall FFI (foreign function interface) library, which is the third-party Ruby library used by our study. Furthermore, we install Python venv for generating figures (`.pdf` files) to illustrate the results of our experiment.

We show how those components are built and installed on two platforms: Ubuntu 22.04 LTS (or macOS 13 Ventura) and Docker. Note that our Python script for the experiment does not run a pdfjs benchmark in a Docker container since it needs a web browser.

After downloading `ruby-refgraph-artifact.zip` from our Zenodo repository, unzip it. A directory `ruby-refgraph-artifact` will be created. We below assume that `/PATH/TO/RUBY-REFGRAPH` refers to this directory. Next, we build and install necessary components.

Ubuntu 22.04 (or macOS 13 Ventura)

- Install the dependencies for ruby 3.1.2-clang-refgraph

```
$ sudo apt update
$ sudo apt install autoconf bison clang libffi-dev libreadline-dev \
                   libssl-dev rbenv ruby
$ echo 'eval "$(rbenv init -)"' >> $HOME/.bashrc
```

- Install npm and Node.js

```
$ sudo apt update
$ sudo apt install npm wget
$ sudo npm install --global n
$ sudo n lts
```

- Build ruby 3.1.2-clang-refgraph

```
$ cd /PATH/TO/RUBY-REFGRAPH
$ autoconf
$ autoreconf --install
$ mkdir build
$ cd build
$ CC=clang ../configure --prefix $(rbenv root)/versions/3.1.2-clang-refgraph \
                        --disable-install-doc
$ make install
$ cd ../refgraph
$ eval "$(rbenv init -)"
$ rbenv shell 3.1.2-clang-refgraph
$ gem install bundler
$ bin/setup
$ bundle exec rake build
$ gem install pkg/refgraph-0.1.0.gem
```

- Install the dependencies for benchmark programs

```
$ cd /PATH/TO/RUBY-REFGRAPH/refgraph/benchmarks/babel
$ npm install
```

- Install the dependencies for a Python script that generates figures.

```
$ sudo apt update
$ sudo apt install python3-venv
$ cd /PATH/TO/RUBY-REFGRAPH/refgraph/experiment
$ python3 -m venv venv --upgrade-deps
$ venv/bin/pip install matplotlib==3.6.1
```

- Install Google Chrome
  - Access `https://www.google.com/chrome/`
  - Click “Download Chrome”
  - Follow the instruction and do the initial setup.
- *Optional.* When you do not run the pdfjs benchmark, set the environment variable `RUBY_REFGRAPH_BENCHMARK_NO_PDFJS` to `TRUE`. You do not need to install Google Chrome, on which only the pdfjs benchmark depend.

Docker container (from Dockerfile)

- Build a docker image

  ```
  $ cd /PATH/TO/RUBY-REFGRAPH
  $ docker build -t ruby-refgraph .
  ```

- Run the built image

  ```
  $ docker run -it ruby-refgraph
  ```

- *Optional.* Since all generated figures are saved into `ruby-refgraph/refgraph/experiment/out`, you might want to mount a local directory `./out` there. In this case, do the following for running the image.

  ```
  $ mkdir ./out
  $ chmod 0777 ./out
  $ docker run -v $(pwd)/out:/root/ruby-refgraph/refgraph/experiment/out \
               -it ruby-refgraph
  ```

Docker container (from Docker image)

- Download a Docker image `ruby-refgraph.image.tar.gz` from our Zenodo repository
- Load the docker image

  ```
  $ docker load < ruby-refgraph.image.tar.gz
  ```

- Run a container

  ```
  $ docker run -it ruby-refgraph
  ```

- *Optional.* Since all generated figures are saved into `ruby-refgraph/refgraph/experiment/out`, you might want to mount a local directory `./out` there. In this case, do the following for running the image.

  ```
  $ mkdir ./out
  $ chmod 0777 ./out
  $ docker run -v $(pwd)/out:/root/ruby-refgraph/refgraph/experiment/out \
               -it ruby-refgraph
  ```

3 Testing

After building and installing necessary components as instructed in the previous section, run test cases in the *same* shell.

- Run tests for ruby 3.1.2-clang-refgraph

  ```
  $ cd /PATH/TO/RUBY-REFGRAPH/refgraph
  $ bundle exec rake test
  ```

All tests are supposed to pass except `test_w_shape_long_chain_looped()` in `test_refgraph.rb`. It may fail when only a limited amount of computing resource is available.

4 How to use ruby 3.1.2-clang-refgraph

- Switch the Ruby VM to `ruby 3.1.2-clang-refgraph`.

  ```
  $ rbenv shell 3.1.2-clang-refgraph
  ```

  This command sets a shell-specific Ruby version so that the Ruby command `ruby` will be bound to that version. For further details, please refer to `https://github.com/rbenv/rbenv`.

- Run the Ruby shell.

  ```
  $ irb
  irb> puts 'Hello, world!!'
  Hello, world!!
  => nil
  irb> require 'refgraph'
  => true
  irb> Jscall.console.log('Hi!')  # call a JavaScript function
  Hi!
  => nil
  irb> Refgraph.gc                # explicitly run Refgraph GC
  => [[0, 0]]
  irb> exit
  ```

  Refgraph GC is an extension to the jscall library. jscall is the third party FFI library for running a JavaScript program from Ruby. The document of this library is available from their github repo `https://github.com/csg-tokyo/jscall`.

5 Reproduce Figures

This section mentions how to reproduce the figures presented in our paper. The outputs are pdf files under `/PATH/TO/RUBY-REFGRAPH/refgraph/experiment/out`. Note that the figures for the pdfjs benchmark are not generated in a Docker container because it needs a web browser.

Since this section runs all the benchmark programs for generating figures, it takes long time, probably, one day. To reduce the execution time, you can decrease the number of iterating the execution of the benchmark programs. By default, it iterates 15 times. The number of iteration is specified in `experiment/run-benchmarks.sh`:

```
# number of iterations
repeat=15
```

Change this line and set `repeat` to, for example, 2. Iterating 2 tiems will take about 2 hours. If the iteration count is not 15, the shell script will produce a warning message *"data may not be sufficient"*, but ignore this message.

To reproduce figures, execute the following commands.

- Move to the benchmark directory. On Ubuntu 22.04,

  ```
  $ cd /PATH/TO/RUBY-REFGRAPH/refgraph
  ```

  In a Docker container,

  ```
  $ cd $HOME/ruby-refgraph/refgraph
  ```

- Run benchmarks to obtain raw results.

  ```
  $ RUBY_REFGRAPH_BENCHMARK_DIR="$(pwd)/benchmarks" experiment/run-benchmarks.sh
  ```

- Generate figures from raw data.

  ```
  $ cd ./experiment
  $ ./generate-figures.sh
  $ ls out
  ```

The output figures will be found under `./out`.

6 Package Versions

We tested our artifact with the following packages. If you see a problem, please consider downgrading (or upgrading) packages.

- Ubuntu `22.04.01 LTS`
- `apt 2.4.8`
- `GNU autoconf 2.71`
- `GNU bison 3.8.2`
- `clang 14.0.0-1ubuntu1`
- `libffi-dev 3.4.2-4`
- `libreadline-dev 8.1.2-1`
- `libssl-dev 3.0.2-0ubuntu1.7`
- `rbenv 1.1.2`
- `ruby` (system) `3.0.2p107 (2021-07-07 revision 0db68f0233) [x86_64-linux-gnu]`
- `npm 8.15.0`
- `wget 1.21.2`
- `n v9.0.0`
- `python 3.10.6`
- `matplotlib 3.6.1`
- `jscall 1.4.0`

7 Source code

The implementation of Refgraph GC consists of an extension to CRuby and a Ruby gem library named Refgraph. This Ruby gem library is an extension to another Ruby gem library, Jscall.

- `ruby-refgraph-artifact`
  The source code of CRuby 3.1.2-clang-refgraph. The source code of the extension to the original CRuby 3.1.2 is obtained by the following command:

  ```
  $ cd /PATH/TO/RUBY-REFGRAPH
  $ git diff refgraph_0.0 refgraph_0.0.1
  ```

- `ruby-refgraph-artifact/refgraph`
  The source code of the Refgraph library.

– `refgraph/lib`	Ruby and JavaScript source
– `refgraph/ext`	C source
– `refgraph/benchmarks`	benchmark programs

The main method of Refgraph GC is `Refgraph.gc`, which is in `lib/refgraph/jscall.rb`. It calls `Refgraph.simply_make` to construct a compressed reference graph in the JSON format. The `simply_make` method is implemented in the C language. Its source code is found in `ext/refgraph/simple.c`.

The constructed graph is sent to JavaScript by calling `Jscall.funcall("Refgraph.gc", json)` or `Jscall.funcall("Refgraph.eager_gc", json)`. This executes `Refgraph.gc(json)` or `eager_gc(json)` in JavaScript through the Jscall library. Those JavaScript functions are found in `lib/refgraph/refgraph.mjs`. Note that `Refgraph.gc` calls `override_methods`, which installs the send barrier by redefining several methods on the export table. The original source code for implementing the export table is included in the Jscall library.[1]

Benchmarks

The source files of the benchmark programs are in `refgraph/benchmarks`. Each benchmark program is a pair of a Ruby program and a JavaScript program. To illustrate the common structure of benchmark programs, let's look at the source file of the loop benchmark because it is the simplest benchmark.

The loop benchmark is run by the following command:

```
$ ruby loop.rb true
```

This runs the benchmark with the Refgraph GC.

The main method of the loop benchmark is `run_benchmark` in `loop.rb`. When Refgraph GC is used, this method first runs `install_refgraph.rb`. It redefines several methods in the Jscall library so that the Refgraph GC will be periodically executed.

Then, `run_benchmark` calls `define_js_class` to execute a JavaScript program, which is embedded in `loop.rb` as the source text, on node.js. It defines a class and a function in JavaScript. The JavaScript programs for some benchmarks are stored in a separate source file. For example, the JavaScript program for the deltablue benchmark is in `deltablue.mjs`.

After executing the JavaScript program, `run_benchmark` starts benchmark looping. For every iteration, it prints logged data by calling `Refgraph.logging`. The source code of `logging` is in `inspect.rb`. For example, it prints the following data:

[1] `https://github.com/csg-tokyo/jscall/blob/main/lib/jscall/main.mjs`

```
total time: 1.013 sec.
gc time: 514.266 msec.
reclaimed=21400. Rb=19.22Mb, Js=16.05Mb
Refgraph-gc count: 1, time: 89.933 msec.
refgraph size: [[35, 21436, 21472, 21471, 333211]]
[9802, 0, 9802, "import/import-zombi/export"]
```

The first line presents the total execution time of the iteration. The second line presents the execution time of Ruby's GC. In the third line, `reclaimed` is the number of proxy objects that are reclaimed in Ruby. `Rb` presents the memory usage in Ruby, and `Js` presents the memory usage in JavaScript. The fourth line presents the number of occurrences of the Refgraph GC and its total execution time during the iteration. The fifth line presents an array of the size of a compressed reference graph created during the execution. The size is represented by five numbers: the number of edges from the root, the number of edges from the other nodes, the number of the nodes in the export table in Ruby, the number of the node in the import table in Ruby, and the byte length of the JSON text representing the graph. The last line presents the number of the live proxy-objects, the number of the dead proxy-objects, and the number of the objects in the export table in Ruby when the iteration finishes.