

# OOPSLA 2019 Artifact

On the Design, Implementation and Use of Laziness in R

---

## Getting Started

The artifact is a [Docker](#) image. The artifact performs a dynamic analysis on packages written in the R language, and analyzes the generated data. The artifact uses this data to generate an HTML report containing the graphs and data appearing in our paper.

The report is served by the Docker container on localhost : 8000.

*The Docker image has been created and tested on Linux using Docker version 18.09.6-ce.*

Docker documentation can be found at: <https://docs.docker.com/>

Instructions to get started with docker can be found at: <https://docs.docker.com/get-started/>

## 1. Download

Instructions to setup Docker on Windows: <https://docs.docker.com/docker-for-windows/>

Instructions to setup Docker on Mac: <https://docs.docker.com/docker-for-mac/>

On Linux, Docker can be installed using the package manager. After installation:

- Start Docker daemon (sudo systemctl start docker)
- Add user to docker group to run Docker commands (sudo addgroup \$USER docker)

It is recommended to *restart the machine* for the settings to take effect.

We used Docker version 18.09.6-ce to build our artifact image.

Pull the artifact Docker image from <https://hub.docker.com/r/aviralgoel/oopsla-2019-r-laziness> with the following command:

```
docker pull aviralgoel/oopsla-2019-r-laziness:latest
```

The artifact is a 2 GB compressed Docker image. After pulling, Docker should show the following sha256 digest:

```
b7a9becec9f3db51b1b0e9dc20bfdbb12c49c9ce74b8b8e5a0c61447511097e8
```

## 2. Start

To start the docker image, run the following command:

```
docker run --cpus 8 --rm -p 8000:8080 -ti \  
aviralgoel/oopsla-2019-r-laziness:latest --name oopsla-2019-r-laziness
```

This command creates a container (instantiates the artifact image), a minimal Linux distribution customized to run the artifact, and opens a bash terminal.

- The `--cpus` option specifies how much of your system CPU resources are available for the container to use.  
Details are here: [https://docs.docker.com/config/containers/resource\\_constraints/](https://docs.docker.com/config/containers/resource_constraints/)  
Our artifact is CPU and Disk intensive. For scalability, we use parallelism heavily. We recommend setting it to as high a value as possible. The (only) downside of a lower value is slower execution.
- The `-p` option is used to describe port mapping.  
Details are here: <https://docs.docker.com/config/containers/container-networking/>  
The container starts an `nginx` web server on port `8080` serving files from `/var/www/`. The container port `8080` is mapped to the system port `8000`. You can navigate to `localhost:8000` in your browser to see an index of the container's `/var/www/` directory. It already contains three files: `paper.pdf`, `small.html` and `large.html` which will be described later on.
- The `-ti` options allocates a `tty` for the container.  
Details are here: <https://docs.docker.com/engine/reference/run/>
- The `--name` option assigns a name to the container.  
Details are here: <https://docs.docker.com/engine/reference/run/#name---name>

The Docker container runs [Debian Buster](#). Precise details of the setup can be found in the Dockerfile: <https://hub.docker.com/r/aviralgoel/oops1a-2019-r-laziness/dockerfile>  
The username and password are `aviral`. For convenience, user `aviral` has passwordless `sudo` privileges.

`Emacs` and `Vim` are included for viewing code. `zsh` and `fish` are installed as `bash` alternatives.

### 3. Run

To run the artifact, type this in a terminal:

```
cd ~/promise-dyntracing-experiment
make pipeline TRACE_DIRPATH=small CORPUS_FILEPATH=corpus/small.txt \
    PARALLEL_JOB_COUNT=8
```

`PARALLEL_JOB_COUNT` should mirror `--cpus` option when starting the docker container.

This `Make` command initiates the pipeline that traces R packages, analyzes the data and generates an HTML report with the figures and data of the paper in a directory named `small`. This pipeline is tracing 5 packages listed in `corpus/small.txt`. You can view the packages using:

```
cat ~/promise-dyntracing-experiment/corpus/small.txt
```

On our machine this takes 15 minutes. The command prints copious amounts of progress information on the terminal. Once done, an HTML report is created at the following location:

```
~/promise-dyntracing-experiment/small/analysis/report/report.html
```

This is copied to `/var/www/`. You can view this report in your browser by navigating to:

localhost:8000/report.html

The figures generated measure the same things as the ones in the paper (but for fewer packages), the paper is here:

localhost:8000/paper.pdf

You can check the correctness of the run by comparing with our results for the same packages:

localhost:8000/small.html

Minor variations in the result are possible due to non-deterministic nature of some programs.

## Step by Step Instructions

*Our artifact is CPU and IO intensive. The data in the paper was generated from 14,875 R packages over 5 days on two servers with 256 GB RAM, 72 cores and 4 TB Hard Disks. The Docker image has 132 R packages to keep the size and build time manageable. A complete Docker image containing all R packages and their dependencies occupies over 140 GB of disk space and takes over 5 hours to build on a 256 GB RAM and 72 core server.*

To run the larger version of the artifact (on 25 R packages), run the following commands in the container:

```
cd ~/promise-dyntracing-experiment
make pipeline TRACE_DIRPATH=large CORPUS_FILEPATH=corpus/large.txt \
    PARALLEL_JOB_COUNT=8
```

PARALLEL\_JOB\_COUNT should mirror --cpus option when starting the docker container.

You can check the list of packages using:

```
cat ~/promise-dyntracing-experiment/corpus/large.txt
```

Once done, an HTML report will be created at the following location.

```
~/promise-dyntracing-experiment/large/analysis/report/report.html
```

This is copied to /var/www/. You can view this report in your browser by navigating to:

localhost:8000/report.html

You can verify that the report contains all the figures of the paper and compare this with a report we generated using the same command:

localhost:8000/large.html

Minor variations in the result are possible due to non-deterministic nature of some programs. The results from 25 packages do not match exactly those of the paper because the number of packages is too small. The data is generated in a directory named `large`.

To analyze other packages, specify them in a text file, one package per line, same as the files in `~/promise-dyntracing-experiment/corpus` directory. This directory has 5 corpus files. The `small.txt` and `large.txt` files were created for this artifact. The `fast.txt` and `slow.txt` files were used in our paper. The `test.txt` file was used for testing and development of the dynamic analysis.

If you create a corpus file `custom.txt` in the `corpus` directory, you can run the dynamic analysis on it using the following commands:

```
cd ~/promise-dyntracing-experiment
make pipeline TRACE_DIRPATH=custom CORPUS_FILEPATH=corpus/custom.txt\
  PARALLEL_JOB_COUNT=4
```

As expected, this will run the analysis and upon completion, copy the generated report `.html` to `/var/www/` which you can view by navigating to `localhost:8000/report.html`

It is important to ensure that the custom packages specified in the corpus file are installed. The list of currently installed packages can be found in the directory:

`~/library`

We have preinstalled 132 packages. You can install more packages by specifying the package names, one package per line, in the file:

`~/promise-dyntracing-experiment/scripts/dependencies.txt`

Then, run the following commands:

```
cd ~/promise-dyntracing-experiment
make install-dependencies
```

This make command runs the following script to install the packages.

`~/promise-dyntracing-experiment/scripts/install-packages.R`

If the installation succeeds without errors (it can fail if dependencies are not met) the installed packages will reside in `~/library` directory.

For our study, we installed over 15,000 R packages from CRAN and BIOCONDUCTOR, the official R package repositories. The R program to install all BIOCONDUCTOR and CRAN packages (~ 16,000 R packages) with our modified R VM (R-dyntrace) is:

`~/promise-dyntracing-experiment/scripts/setup-package-repositories.R`

This script is invoked by the make rule `setup-package-repositories`. However, this program will take more than a day to download all the packages and install them on a standard desktop or laptop. We only use this command for setting up the environment on our servers.

There is a similar make rule, `mirror-package-repositories`, that quickly downloads the source code of all R packages by mirroring CRAN and BIOCONDUCTOR using `rsync`. Note that it does not install the packages. This command takes a few hours to run but over 150 GB disk space.

The artifact is divided into three checked out and pre-installed git repositories in

`/home/aviral/`

The repositories are described below.

<code>R-dyntrace</code>	Modified R VM with probes
<code>promisedyntracer</code>	Dynamic Analyzer
<code>promise-dyntracing-experiment</code>	Dynamic analysis pipeline

`promisedyntracer` is installed as an R package with our modified R VM `R-dyntrace`. It uses the probes exposed by `R-dyntrace` for catching interpreter events and collecting information about laziness. `promise-dyntracing-experiment` is the control center of our study. It extracts executable programs (test, vignettes, examples) from the R packages, executes them against the dynamic analyzer in parallel, analyzes the data and generates reports.

The API exposed by `R-dyntrace` for dynamic analysis can be found in the file:

`~/R-dyntrace/src/include/Rdyntrace.h`

The `dyntrace_t` struct in this file defines an API for callbacks that are executed on specific interpreter events.

These callbacks are defined in:

`~/promisedyntracer/src/probes.cpp`

These callbacks in conjunction with various helper classes in the same directory contain the dynamic analysis logic. The high-level R API for initiating the analysis is defined in:

`~/promisedyntracer/R/promisedyntracer.R`

The last part of our artifact is the analysis pipeline. This is implemented as a collection of R programs that are invoked from a top-level Makefile at:

```
~/promise-dyntracing-experiment/Makefile
```

The R program to extract runnable code from packages for dynamic analysis is:

```
~/promise-dyntracing-experiment/scripts/extract-tests-examples-vignettes.R
```

This script is invoked by the make rule `extract-tests-examples-and-vignettes`. You can see the extracted programs in the `~/promise-dyntracing-experiment/small/corpus` or `~/promise-dyntracing-experiment/large/corpus` directory. It has the following layout:

```
small/
├── corpus
│   ├── <package-name>
│   │   ├── doc
│   │   │   ├── <filename>.Rmd
│   │   │   └── __wrapped__<filename>.R
│   │   └── ...
│   │       ├── examples
│   │       │   ├── <filename>.R
│   │       │   └── __wrapped__<filename>.R
│   │       └── ...
│   └── tests
│       ├── <filename>.R
│       └── __wrapped__<filename>.R
└── ...
```

The `doc`, `examples` and `tests` directory contain vignettes, examples and test scripts respectively, extracted from the corresponding R package. For each script `<filename>.R` or `<filename>.Rmd`, a script with name `__wrapped__<filename>.R` is generated which contains the same code wrapped in a call to the dynamic analyzer.

The data generated by the dynamic analysis is present in the directory `~/promise-dyntracing-experiment/small/analysis/raw` with the following layout:

```
small/
├── analysis
│   ├── raw
│   │   ├── <package-name>
│   │   │   ├── <script-type>
│   │   │   │   ├── <script-name>
│   │   │   │   │   ├── CONFIGURATION
│   │   │   │   │   └── BEGIN
│   │   └── ...
└── ...
```



~/promise-dyntracing-experiment/small/logs

The logs directory mirrors the structure of the analysis subdirectories to make it possible to identify specific logs.

To get a sense of the size of data, you can run the following commands:

```
cd ~/promise-dyntracing-experiment/small
du -sh
```

The data layout for the aforementioned analysis is shown below.

```
small/
├── analysis
│   ├── prescanned
│   │   └── traced_scripts.csv
│   ├── reduced
│   │   ├── <data-table-name>
│   │   │   ├── <package-name>
│   │   │   │   ├── <script-tpe>
│   │   │   │   │   └── <script-name>
│   │   │   │   ├── BEGIN
│   │   │   │   ├── FINISH
│   │   │   │   ├── NOERROR
│   │   │   │   └── <sub-data-table-name>.bin.zst
│   │   └── ...
│   ├── scanned
│   │   ├── all_scripts.csv
│   │   ├── invalid_scripts.csv
│   │   └── valid_scripts.csv
│   ├── combined
│   │   ├── <data-table-name>
│   │   │   ├── <sub-data-table-name>
│   │   │   │   ├── aviral-part-000001.bin.zst
│   │   │   │   ├── BEGIN
│   │   │   │   ├── FINISH
│   │   │   │   └── ERROR
│   │   └── ...
│   └── <script-tpe>
│       └── <script-name>
│           ├── BEGIN
│           ├── FINISH
│           ├── NOERROR
│           └── <sub-data-table-name>.bin.zst
│               ...
└── merged
```



```

├── events
├── scanned
├── log
├── summarized
├── <data-table-name>
├── summary
│   ├── raw
│   └── reduced
│       └── <data-table-name>
└── procfile

```

## Errors and Warnings

Some errors/warnings have no effect on the correctness of the results.

The following warnings generated by R can be safely ignored. These are either deprecation warnings or superfluous for they warn us of ignoring some aspect of the computation that we are not interested in.

---

```

Warning: `recursive` is deprecated, please use `recurse` instead
Warning: `recursive` is deprecated, please use `recurse` instead

```

---

Warning messages:

```

1: In bind_rows_(x, .id) :
  Vectorizing 'fs_path' elements may not preserve their attributes
2: In bind_rows_(x, .id) :
  Vectorizing 'fs_path' elements may not preserve their attributes
3: In bind_rows_(x, .id) :
  Vectorizing 'fs_path' elements may not preserve their attributes
4: In bind_rows_(x, .id) :
  Vectorizing 'fs_path' elements may not preserve their attributes
5: In bind_rows_(x, .id) :
  Vectorizing 'fs_path' elements may not preserve their attributes
6: In bind_rows_(x, .id) :
  Vectorizing 'fs_path' elements may not preserve their attributes
7: In bind_rows_(x, .id) :
  Vectorizing 'fs_path' elements may not preserve their attributes

```

---

The following error generated by xvfb can be safely ignored. Xvfb is X virtual framebuffer. It is an in-memory display server used for running graphical applications on remote servers with no display. R needs an X11 server for making graphs and the entire tracing pipeline spawns processes using xvfb. The following error happens at the end of the tracing and does not affect the artifact or its results.

---

xvfb-run: usage error: need a command to run

Usage: xvfb-run [OPTION ...] COMMAND

Run COMMAND (usually an X client) in a virtual X server environment.

Options:

-a --auto-servernum try to get a free server number, starting at  
--server-num (deprecated, use --auto-display  
instead)

-d --auto-display use the X server to find a display number  
automatically

-e FILE --error-file=FILE file used to store xauth errors and Xvfb  
output (default: /dev/null)

-f FILE --auth-file=FILE file used to store auth cookie  
(default: ../Xauthority)

-h --help display this usage message and exit

-n NUM --server-num=NUM server number to use (default: 99)

-l --listen-tcp enable TCP port listening in the X server

-p PROTO --xauth-protocol=PROTO X authority protocol name to use  
(default: xauth command's default)

-s ARGS --server-args=ARGS arguments (other than server number and  
"-nolisten tcp") to pass to the Xvfb server  
(default: "-screen 0 640x480x24")

-w DELAY --wait=DELAY delay in seconds to wait for Xvfb to start  
before running COMMAND (default: 3)

make[1]: \*\*\* [Makefile:222: trace-ast] Error 2

make[1]: Leaving directory '/home/aviral/promise-dyntracing-experiment'

make: [Makefile:603: pipeline] Error 2 (ignored)

---