


Runtime Instrumentation for Reactive Components (Artifact)

Luca Aceto ✉ 


Reykjavik University, Iceland
Gran Sasso Science Institute, L'Aquila, Italy

Duncan Paul Attard ✉ 

University of Glasgow, UK

Adrian Francalanza ✉ 

University of Malta, Msida, Malta

Anna Ingólfssdóttir ✉ 

Reykjavik University, Iceland

Abstract

Reactive software calls for instrumentation methods that uphold the reactive attributes of systems. Runtime verification sets another demand on the instrumentation, namely that the trace event sequences it reports to monitors are *sound*, *i.e.*, they reflect actual executions of the system under scrutiny. Our companion paper, “Runtime Instrumentation for Reactive Components”, presents RIARC, a novel decentralised instrumentation algorithm for outline monitors that meets these two demands. RIARC uses a next-hop IP routing approach to rearrange and report events soundly to monitors despite the potential trace event loss or reordering

stemming from the asynchrony of reactive systems.

The companion paper shows our corresponding RIARC Erlang implementation to be correct through rigorous systematic testing. We also assess RIARC via extensive empirical experiments, subjecting it to large realistic workloads in order to ascertain its reactivity. This artefact packages the RIARC Erlang implementation, systematic tests that demonstrate its correctness, data sets obtained from our original empirical experiments detailed in the companion paper, and the scripts to rerun and replicate these results under lower workloads.

2012 ACM Subject Classification Software and its engineering → Software verification and validation

Keywords and phrases Runtime instrumentation, decentralised monitoring, reactive systems

Digital Object Identifier 10.4230/DARTS.10.2.1

Funding This work is supported by the Reykjavik University Research Fund, the Doctoral Student Grant (No: 207055) and the MoVeMnt project (No: 217987) under the IRF, and the STARDUST project (No: EP/T014628/1) under the EPSRC.

Acknowledgements We thank our reviewers and the Artefact Evaluation Committee for their feedback. Thanks also to Keith Bugeja, Simon Fowler, Simon Gay, and Phil Trinder for their input.

Related Article Luca Aceto, Duncan Paul Attard, Adrian Francalanza, and Anna Ingólfssdóttir, “Runtime Instrumentation for Reactive Components”, in 38th European Conference on Object-Oriented Programming (ECOOP 2024), LIPIcs, Vol. 313, pp. 2:1–2:33, 2024.

<https://doi.org/10.4230/LIPIcs.ECOOP.2024.2>

Related Conference 38th European Conference on Object-Oriented Programming (ECOOP 2024), September 16–20, 2024, Vienna, Austria

Evaluation Policy The artifact has been evaluated as described in the ECOOP 2024 Call for Artifacts and the ACM Artifact Review and Badging Policy.



© Luca Aceto, Duncan Paul Attard, Adrian Francalanza, and Anna Ingólfssdóttir; licensed under Creative Commons License CC-BY 4.0

Dagstuhl Artifacts Series, Vol. 10, Issue 2, Artifact No. 1, pp. 1:1–1:4



DAGSTUHL
ARTIFACTS SERIES

Dagstuhl Artifacts Series
Schloss Dagstuhl – Leibniz-Zentrum für Informatik,
Dagstuhl Publishing, Germany



1 Scope

Our artefact packages the Erlang implementation of inline and RIARC monitoring, together with the experiment set-up and scripts required to launch them. We omit the centralised monitoring experiments discussed in our companion paper since it was shown that they are liable to failure, making them hard to automate. Interested readers are referred to the extended version of our companion paper [1, fig. 14 in app. C.6.1] for details on centralised monitoring.

Distribution. The artefact is packaged as two Docker images: `ecoop-77:1.0.tar`, built for Apple (M1) machines, and `ecoop-77-amd64:1.0.tar`, built for (Linux/amd64) machines. Both images are preloaded with experiments that reproduce our inline and RIARC monitoring results discussed in Secs. 5.4.2, 5.4.3, and 5.5 in the companion paper. The images can be downloaded from Zenodo, including the instructions guiding users to launch the Docker container. These instructions are also found in app. A.

Experiment set-up. Both the `ecoop-77:1.0.tar` and `ecoop-77-amd64:1.0.tar` Docker images provide two experiment configurations:

High concurrency scenarios perform short-lived tasks to induce maximum stress in experiments (emulate systems such as web apps). These use 100k workers and 100 work requests/worker.

Moderate concurrency scenarios engage in long-running, computationally-intensive tasks (emulate systems such as Big Data stream processing). These use 1k workers and 10k work requests/worker.

The high and moderate concurrency scenarios each generate $\approx 20\text{M}$ send and receive messages, producing $\approx 40\text{M}$ analysable trace events. Every experiment consists of 10 benchmarks, which are performed in steps of progressively-increasing workload. Each benchmark generates 100 seconds of workload on the system under these three workload shapes:

Steady models the system under stable workload (Poisson-shaped)

Pulse models the system under gradually rising and falling workload (Normal distribution-shaped)

Burst models the system under instant workload spikes (Log-normal distribution-shaped)

Experiment execution. The instructions included in our distribution focus on the *high concurrency scenarios*, as these results capture the major part of the plots in Sec. 5 of the companion paper. Our experiments induce the worst-case workload scenarios on the reactive system models generated by attaching one dedicated monitor to every system component. We include optional instructions to run the empirical experiments for the moderate concurrency scenario. The scripts that run the systematic tests can be launched by typing `make test`.

2 Content

Our distribution packages the following:

1. Erlang implementation of the inline and RIARC instrumentation algorithms;
2. Erlang monitors we use to evaluate these implementations in our companion paper;
3. Python scripts for launching and performing the empirical experiments;
4. Makefile to run the systematic tests that validate our RIARC implementation.

The distribution defaults to the `/ecoop/scripts` directory at startup. The `/ecoop/home` directory is organised as shown in tbl. 1.

■ **Table 1** Organisation of the /ecoop home directory.

Directory	Sub-directory	Description
data/		<i>Data set from experiments in .csv format</i>
	paper/	Original data sets reported in our paper
	hc/	Default data directory for high concurrency scenario experiments
	mc/	Default data directory for moderate concurrency scenario experiments
plots/		<i>Plots from experiments</i>
	paper/	Original plots included in our companion paper
	hc/	Default data directory for high concurrency scenario plots
	mc/	Default data directory for moderate concurrency scenario plots
src/		<i>Erlang code</i>
	benchcrv/	Master-worker model and workload generation supporting libraries
	experiments/	Experiments launcher
	examples/	Examples used in reusability evaluation
	riarc/	Inline and RIARC monitoring libraries
	tools/	Offline tracing supporting libraries
ebin/		<i>Default compiled Erlang code directory</i>
scripts/		<i>Python code</i>
	simulate/ecoop/	Python front-end for executing experiments and generating plots

3 Getting the artefact

The artefact endorsed by the Artefact Evaluation Committee is available free of charge on the Dagstuhl Research Online Publication Server (DROPS). The artefact can also be downloaded from Zenodo, <https://doi.org/10.5281/zenodo.10634182>.

4 Tested platforms

We compiled, tested, and ran our distribution on the following machines:

- Intel Core i7 M620, 8GB of memory, with Ubuntu 18.04 LTS and Erlang/OTP 22.2.1 (4 threads)
- Intel Core i9 9880H, 16GB of memory, with macOS 12.3.1 and Erlang/OTP 25.0.3 (16 threads)
- Apple M1, 16GB of memory, with macOS 12.7 and Erlang/OTP 26.2.1 (10 threads)

5 License

Our distribution is available under the GNU General Public License v3.0 or later.

6 MD5 sum of the artefact

5b2ed7ae10ce4f7c966f9c2aafec00bb

7 Size of the artefact

- 1.3GB `ecoop-77:1.0.tar`
- 1.3GB `ecoop-77-amd64:1.0.tar`

A Installation instructions

A.1 Set up

1. Download and install Docker.
2. Download the `ecoop-77:1.0.tar` [\(M1\)](#) or `ecoop-77-amd64:1.0.tar` [Linux/amd64](#) image from Zenodo.

A.2 Launching the Docker container from the terminal

1. Change your directory to `~/Downloads`.
2. Load the image by typing the following on the terminal, depending on your system:
 - `sudo docker load -i "ecoop-77_1.0.tar"` [\(M1\)](#)
 - `sudo docker load -i "ecoop-77-amd64_1.0.tar"` [Linux/amd64](#)
3. Verify that the image has been loaded by typing:
 - `docker images`

The `ecoop-77` (or `ecoop-77-amd64`) with tag `1.0` should be listed.

4. Run the image in a Docker container using the name `ecoop` by typing:
 - `docker run -name ecoop -p 8080:8080 -it ecoop-77:1.0` ↵
`bash -c "../run.sh; bash"` [\(M1\)](#)
 - `docker run -name ecoop -p 8080:8080 -it ecoop-77-amd64:1.0` ↵
`bash -c "../run.sh; bash"` [Linux/amd64](#)

This launches the Docker container in interactive mode, enabling users to run the evaluation scripts. The name `ecoop` is used to retrieve experiment results from outside the container.

A.3 Running the experiments

1. Ensure that the current home directory is `/ecoop/scripts`.
2. Run experiments:
 - `python -m simulate.ecoop.main run all`

`run all` should take around 3 hours to complete and requires no intervention from users.
3. Plot results:
 - `python -m simulate.ecoop.main plot all`

`plot all` should take around 1 minute to complete.

The artefact evaluation instructions can be accessed locally at `http://localhost:8080` after completing apps. A.1 and A.2. A copy of these instructions is included in the `site.tar` archive on Zenodo. This can be extracted using `tar xvf site.tar` if required.

References

- 1 Luca Aceto, Duncan Paul Attard, Adrian Francalanza, and Anna Ingólfssdóttir. Runtime Instrumentation for Reactive Components. *CoRR*, abs/2406.19904, 2024. [arXiv:2406.19904](#).