FASE 2023 Artifact

NOTE: this README is available also in HTML or PDF format.

Summary

This artifact contains a prototype of a runtime monitoring middleware called VAMOS as discussed in the FASE 2023 paper *VAMOS: Middleware for Best-Effort Third-Party Monitoring* by Marek Chalupa, Stefanie Muroya Lei, Fabian Muehlboeck, and Thomas A. Henzinger. The artifact consist of the software, benchmarks discussed in the paper, and a set of scripts to run experiments and present the results.

Hardware Requirements

For all benchmarks to make sense, the artifact requires a reasonably modern (~past 5? years) x86-64 processor with multiple cores (ideally at least 6 cores).

Using a bundled docker image

You can load the bundled docker image with `docker load` command and run it:

```
docker load < vamos-fase2023.tar.gz
docker run -ti -v "$(pwd)/results":/opt/results vamos:fase
```

Note that the image was build for x86 architectures. If your architecture is different, you either need to specify `--platform` when running/creating the container to emulate your architecture or build the image yourself. Note that emulating the architecture brings non-negligible overhead and so building the image may be a better option.

Building the artifact

To build and run the artifact run this command from the top-level directory:

```
docker build . -t vamos:fase
```

If you are on a new enough Linux system, you may use this command to get faster builds:

```
DOCKER_BUILDKIT=1 docker build . -t vamos:fase
```

The building process can take more than 10 minutes, based on the used machine and the speed of the internet connection.

To run the built image, use:

```
docker run -ti -v "$(pwd)/results":/opt/results vamos:fase
```

This command starts a docker container with experiments ready to run and gives you a shell in this container. It also creates a folder `results/` in the current

directory to the container where will the results of the experiments appear. Feel free to change `$(pwd)/results` to a directory of your choice.

Once in the docker container, continue with the test instructions below.

Test Instructions

We have prepared script to run three versions of experiments: short, mid, and full experiments. Short experiments run just a few minutes, but the results may diverge from the results in the paper. Mid experiments run longer but still in the order of tens of minutes. Full experiments can run for several hours and should reproduce numbers from the paper. Note that running experiments will overwrite results of previous runs of experiments.

You can run short/mid/full experiments with these scripts:

```
./short-experiments.sh
./mid-experiments.sh
./full-experiments.sh
```

A guide on how to run just some experiments is at the end of this document.

Once experiments are finished, plots in the PDF format should be automatically generated and put in the results folder from where the docker was started (or in a folder of your choice if you changed the command). The plots are named after the figures in the paper.

You can also generate the plots manually by running make in the `plots` directory. Moreover, you can generate the plots from the original data that we measured. The data are in `plots/original-data` and you generate the plots by

```
cd plots
make original
```

For more details about plots, check the README.md inside `plots` directory.

Running just some experiments

To run just some experiments, you can comment out lines with experiments in `{short,full}-experiments.sh` scripts and rerun these scripts or you can follow what these script do:

First, pick if you want short of full experiments and according to that, copy `scripts/setup-short.sh` or `scripts/setup-full.sh` into `setup.sh` (we are in the top-level directory now):

```
cp scripts/setup-short.sh setup.sh   # short experiments
cp scripts/setup-full.sh setup.sh   # full experiments
```

Then, go into directories with experiments and run `make experiments` in the directory, for example:

```
cd scalability
make experiments
```

Available experiment directories are `scalability`, `primes`, `bank`, `bank-tessla`, and `dataraces`. The only exception from this pattern are primes-tessla experiments that are run from the `primes` directory using `make experiments-tessla`:

```
cd primes
make experiments-tessla
```