1 DDMon artifact evaluation

This document contains instructions on how to evaluate the DDMon artifact running within a Docker container.

As an alternative, similar commands can be executed locally, after performing a local installation as described in README.md.

1.1 Getting started

For this evaluation, we recommend building and running DDMon within a Docker container. On Linux, make sure you have the docker service running. On Mac, make sure you have Docker Desktop open.

To build the docker image, run the following commands:

```
mkdir output              # Creates a directory for the plots and other data
docker build -t ddmon .   # Creates a Docker image called 'ddmon'
```

(Note: the creation of the Docker image might take several minutes.)

1.2 "Kick the tires" instructions

The following instructions explain how to assess whether DDMon is working as intended within the Docker container created above. These steps check two features: the generation of PDF files containing plots, and the execution of test scenarios.

1.2.1 Plot generation

Run the following command:

```
docker run --rm -v "$(pwd)/output:/app/output" ddmon ./bench.sh small
```

While the program runs, you should see several legends and animated diagnostics looking as follows:

```
*** Running experiments on 120 scenarios. Experiment status legend:
 . Waiting | o Preparing | O Working | @ Success | D Deadlock | ! Crash | T Timeout
[ . o D @ o @ @ @ D . D @ D T O @ D D D @ ]
```

The command should terminate after about a minute and print `Done` at the end.

No stack trace (i.e., a wall of red text) should be printed, nor the animation should show a flashing exclamation mark (except the legend) — otherwise, please send us the entire output. If you run into permission issues, please delete the `output` folder and try again.

After the program has finished, the `output` folder should contain PDF files with various plots (e.g. `output/fig_15_a.pdf` or `output/fig_16_b/conditional__1000__2619635.pdf`; the name of the latter may differ).

Please try visualising such PDF files with a PDF viewer: if the files contain some form of plot (the plotted values are not important), then this step of the "Kick the tires" assessment is completed.

1.2.2 Test scenarios

Run the following command:

```
docker run --rm ddmon ./ddmon scenarios/supersimple.conf
```

The output should resemble the following:

```
Node: ddmon@32063e08d1f7
Timeout: 2010ms
Sessions: s
00:000:030  | M0:   I0 ? Q(s)
00:000:058  | M0:   %[ Q(s) ]
00:000:079  | M0:   P0 ! Q(s)
00:010:567  | M0:    ? R(s)
00:010:590  | M0:   %[ ?! R(s) ]
00:010:601  | M0:   I0 ! R(s)
Registered 6 events:
    2   messages sent
    1   queries
    1   replies
    0   probes
```

```
    2   mque picks
Directions:
    1   involving Init
    0   Mon -> Mon
    0   Mon -> Proc
    1   Proc -> Mon
    0   Proc -> Proc
State changes:
    0   unlocks
    0   locks
    0   deadlocks
Time: 10570973

### TERMINATED ###
```

More specifically, the generated output should satisfy these requirements:

1. It should end with the line `### TERMINATED ###`.
2. It should not contain stack traces nor error messages — otherwise, please send us the entire output.
3. The time measurements (e.g. `00:010:601` or `Time: 10570973`) must be present, although they may be different.

If the 3 requirements above are met, then this step of the "Kick the tires" assessment is completed.

1.3 Reproducing the results in the paper: step-by-step instructions

1.3.1 Reproducing the plots in *Figures 15 and 16*

To reproduce *Figures 15 and 16*, run the following commands:

```
rm -fr output  # Remove the 'output' directory, if it already exists
mkdir output
docker run --rm -v "$(pwd)/output:/app/output" ddmon ./bench.sh
```

Note: the `docker` command above takes about **one hour** to complete on a computer with Intel Core i7-1185G7 (8 CPU cores), **32GB of RAM**, running Fedora 42. The command generates detailed statistics from every experiment, which may take up to 20GB of disc space in total. This is what we used to obtain the results in the paper. If the command crashes and displays `/app/bench.sh: line 30: 1193 Killed "$@"` anywhere in the output, it means that the program has likely run out of memory. If that happens, delete the `output` folder and try the smaller variant described below.

For a less resource-intensive variant (which may less accurately align to the results in the paper), you can run the following `docker` command instead: it takes about 10 minutes on the same computer, and needs **10GB of *available* RAM**.

```
docker run --rm -v "$(pwd)/output:/app/output" ddmon ./bench.sh medium
```

After the `docker` command terminates, the following PDF files should be generated:

- *Figure 15a*: `output/fig_15_a.pdf`
- *Figure 15b*: `output/fig_15_b.pdf`
- *Figure 15c*: `output/fig_15_c.pdf`
- *Figure 16a*: `output/fig_16_a/*.pdf` (many files)
- *Figure 16b*: `output/fig_16_b/*.pdf` (many files)
- *Figure 16c*: `output/fig_16_c/*.pdf` (many files)

Note that the produced plots may look different w.r.t. those in the paper. This is because the scripts perform multiple executions of several concurrent systems, and each execution may or may not deadlock at a certain time, depending on (1) intrinsic nondeterminism in their behaviour, and (2) further nondeterminism introduced by scheduling (similarly to the non-deterministic deadlock illustrated in *Example 3.10* in the paper).

The following subsections explain how to compare the plots produced by the artifact scripts above with the plots in the paper.

1.3.1.1 *Figure 15*

The plots produced for *Figure 15* by our script (i.e., `output/fig_15_a.pdf`, `output/fig_15_b.pdf`, and `output/fig_15_c.pdf`) may be slightly different w.r.t. the paper, but the overall trends should be the same:

- In *Figures 15a, 15b and 15c*, the orange line should show values greater or equal to blue and green lines, while red line should be above the orange line.
- In *Figures 15a and 15b*, the blue line should present roughly the same values as the green line.
- In *Figure 15b*, the blue and green lines should be close to zero.
- In *Figure 15c*, the blue line should show values approximately 3 times lower than the other lines.

1.3.1.2 *Figure 16*

Each time series shown in *Figure 16* in the paper visualises one specific execution of a non-deterministic concurrent system, under different probe emission delays (0ms, 1000ms, or 5000ms). To produce *Figure 16* we manually selected 3 executions that clearly show how many queries, responses, and probes may be emitted, and when.

The script in the artifact performs multiple executions of a non-deterministic concurrent system for each probe delay, and plots and saves the time series of each execution. As a consequence, the script produces many candidate plots for Figures *16a*, *16b* and *16c*. The script saves all such candidate plots as PDF files in the directories `output/fig_16_a/`, `output/fig_16_b/`, and `output/fig_16_c/`, respectively.

In each produced plot, there may be any number of deadlocks (usually zero or one) reported as red dashed vertical lines; to create the figures in the paper, we selected two plots without deadlocks (Figures *16a* and *16b*), and one plot with a deadlock (Figure *16c*), and we manually placed the corresponding legends.

When inspecting the candidate plots produced by the benchmarking scripts, you should be able to observe the following trends (also visible in the paper):

- In the candidate plots for *Figure 16a* (PDF files saved in the directory `output/fig_16_a/`), the red solid line rises early, and often rises above the dashed cyan line.
- In the candidate plots for *Figure 16b* (PDF files saved in the directory `output/fig_16_b/`), the orange solid line is generally below both the dashed cyan and dotted blue lines.
- In the candidate plots for *Figure 16c* (PDF files saved in the directory `output/fig_16_c/`), the green solid line is generally below both dashed cyan and dotted blue lines; moreover, the green solid line should be always close to zero, *unless* a red dashed vertical line (indicating a deadlock) is present. (This happens because probe emission increases when a deadlock occurs.)

1.3.2 Reproducing the test scenario execution logs in Appendix A - *Listings 1, 2, 3, 4*

The following instructions describe how to reproduce the results of two deadlock detection test scenarios:

- *Listing 2* in Appendix A.1 (based on the scenario in *Figure 7* implemented in *Listing 1*).
- *Listings 3* and *4* in *Figure 19* in Appendix A.1.

NOTE: the deadlock detection test scenarios described below are nondeterministic: when they are executed, they may or may not deadlock. This is due to (1) intrinsic nondeterminism in the scenario itself, and (2) further nondeterminism introduced by scheduling (similarly to the non-deterministic deadlock illustrated in *Example 3.10* in the paper). Therefore, it may be necessary to **reexecute the test scenarios multiple times** to obtain a deadlock and a corresponding detection log that closely correspond to *Listings 2, 3, and 4* in the companion paper.

NOTE: To correctly visualise the logs generated by the test scenarios below, you may need to run them in a maximised terminal window with a small font: this is because each log line may be very long. If that does not suffice, you can try adding the option `--indent=0` at the very end of each of the commands below: this will "flatten" the output.

NOTE: The output log may contain additional entries and it may be formatted slightly differently compared to listings in the companion paper.

NOTE: the meaning of the scenario execution logs is documented in SCENARIOS.md. The file illustrates a list of further example scenarios, including e.g. replicated services as described in *Appendix B*.

1.3.2.1 Listings 1 and 2

The log in *Listing 2* is obtained with the following command, where the file `scenarios/envelope.conf` corresponds to *Listing 1* in the paper.

```
docker run --rm ddmon ./ddmon scenarios/envelope.conf
```

1.3.2.2 Listings 3 and 4 (in *Figure 19*)

The logs in *Listings 3* and *4* (in *Figure 19*) are obtained from two executions of the following command:

```
docker run --rm ddmon ./ddmon scenarios/envelope-small.conf
```

As mentioned above, several runs might be needed to obtain each of the two possible outcomes ("terminated" and "deadlock").

NOTE: In the paper, the captions of *Listings 3* and *4* are swapped. This is an error which we will fix.

1.3.2.3 Running other deadlock detection scenarios

As mentioned in *Appendix 1* of the paper, DDMon comes with a testing DSL for reproducing various (possibly deadlocking) scenarios of `gen_server`-based systems. Such scenarios are specified in `.conf` files which describe the calls exchanged by services in a simulated network. We provide a number of different scenarios in the `scenarios` directory.

To execute e.g. the scenario called `scenarios/example.conf`, run:

```
docker run --rm ddmon ./ddmon scenarios/example.conf
```

For more information on using the testing DSL, see the documentation in SCENARIOS.md.

2 Using DDMon to monitor a `gen_server`-based application

Please see the documentation in EXAMPLE.md.