A. Artifact Description

A.1 Abstract

The artifact proposed for the evaluation is the kernel of the application presented in Sect. 5.1, used for the evaluation of the latency- and energy-aware scaling strategies for data stream processing partitioned-stateful operators. It represents the computational kernel of a *High Frequency Trading* application. The *source* thread generates a stream of financial quotes, while the application (named `elastic-hft` in the following) processes the incoming quotes grouped by stock symbols. A sliding window is maintained for each group (quotes belonging to the same stock symbol) and the computation returns a fitting polynomial for the last received quotes. For the moment being, the *consumer* functionality is executed by the *merger* thread and it is in charge of saving the computation results (in a human readable format) into a file.

For evaluating the artifact, it is required an Intel multicore machine with at least 8 (physical) cores (no virtual machines). Processor(s) has to be of the Sandy Bridge generation (or newer). A Linux-based Operating System and root privileges are required. The artifact uses external libraries: all them are free and publicly available. The artifact should be evaluated without any concurrent application executed in the meanwhile, i.e. the machine must be dedicated to the execution of the artifact. The evaluation is aimed at *reproducing* qualitatively the results reported in the paper, since the application performance varies depending on the used hardware. In particular, it is possible to execute different strategies (`Lat-Node` and `Lat-Power`) with different configurations (`NoSw`, `Sw h=1`, `Sw h=2`, `Sw h=3`) and the results will show that by increasing the horizon length, with the switching cost enabled, we will be able to achieve a better tradeoff between the SASO properties with respect to the configuration of the strategy without switching cost (`NoSw`). A comparison with other two existing strategies can also be evaluated, as described in the paper. Also in this case, the strategies with the switching cost enabled and a sufficiently long horizon ($h = 2, 3$) achieve a better SASO trade-off than the existing solutions.

A.2 Description

A.2.1 Check-list (artifact meta information)

- **Compilation:** it is required the `gcc` compiler, version 4.8 (or newer), and glibc version 2.17 (or newer).
- **Dataset:** the artifact has been evaluated using a synthetic dataset (included in the repository) and a real one available through an external link (size $\sim$ 3GB).
- **Runtime environment:** the artifact requires a Linux based operating systems (kernel version 2.6 or newer). It relies on external libraries: all of them are free and publicly available. It is required to have root access to the used machine.
- **Hardware:** it is required a multicore machine with at least 8 (physical) cores. Processor(s) has to belong to the Intel Sandy-Bridge family or newer (required for energy measurements and DVFS support). The CPU frequency driver used by the OS must be `acpi-cpufreq`.
- **Execution:** the artifact has to be executed over a physical (not virtualized) machine without any concurrent application executed (apart from OS processes). The threads of the artifact are pinned on the physical cores.
- **Output:** the output is shown in the standard output and saved in files. It shows all the required metrics (e.g., measured average latency, number of reconfigurations, and so forth) that are required to evaluate the various scaling strategies proposed in the paper;
- **Experiment workflow:** to obtain results qualitatively similar to the ones in the paper, it is required to manually start the artifact with various configuration files. Scripts are provided to perform multiple runs and derive average metrics.
- **Publicly available?:** Yes.

A.2.2 How delivered

The artifact is released through the public GitHub code repository available at: `https://github.com/tizianodem/elastic-hft`

A.2.3 Hardware dependencies

To run the artifact it is required a multicore machine with at least 8 (physical) cores. Processor(s) has to belong to the Intel Sandy-Bridge family or newer (required for energy measurements and DVFS support). The artifact does not have strict memory requirements, though having at least 16GB of RAM is a suggested configuration. The machine that was used for the experiment reported in the paper is a dual-CPU Intel Sandy Bridge (Xeon E5-2650) with 16 physical cores (32 thread contexts) and 32GB RAM.

A.2.4 Software dependencies

The artifact uses external libraries. In particular:

- `Fastflow`: a C++ parallel programming framework targeting shared-memory architectures. Website: `http://calvados.di.unipi.it/`
- `Lmfit`: a C library for *Levenberg-Marquardt* least-squares minimization and curve fitting. It is used to produce a fitting polynomial of the aggregated quotes (per window). Website: `http://apps.jcns.fz-juelich.de/doku/sc/lmfit`
- `Mammut`: a C++ library providing mechanisms for collecting energy statistics and for changing the CPU frequency. Website: `https://github.com/DanieleDeSensi/Mammut`

In the sequel we will provide a brief description of the installation procedure. Further information are available on the previously cited websites.

A.2.5 Datasets

The artifact takes in input a stream of financial quotes (i.e. *bid* and *ask*), which are generated by an external source (in the following `generator`) provided in the code repository. In the Experiment section of the paper, two different datasets (see Sect. 5.1) have been used to drive the generation of the quotes stream:

- a *synthetic trace*: everything you need to use it is already in the GIT repository;
- a *real trace* that contains the quotes of a trading day in the NASDAQ stock exchange (daily TaQ of 30 Oct. 2014). The original dataset (that contains other additional info) is available at `http://www.nyxdata.com/Data-Products/Daily-TAQ`. You can find a smaller binary version ready to be ingested by the `generator` at `www.di.unipi.it/~dematteis/taq_nasdaq_2014_49544800`. It contains all the quotes for the training day, for a total of 49,544,800 quotes related to 2,836 different stock symbols.

A.3 Installation

A.3.1 Required libraries

In the following we will provide the basic information to install locally (e.g., in your own home directory) all the required libraries.

FastFlow: it is an header-only library. Therefore, it is only required to download it from the website or the SVN. To download the latest version and save it into the `fastflow` directory, run the following command in the shell:

```
$ svn checkout svn://svn.code.sf.net/p/mc-fastflow/\
    code/ fastflow
```

Lmfit: it can be downloaded from the SVN linked in the website. To download the version used in the artifact execute the following command:

```
$ wget http://sourceforge.net/projects/lmfit/files/\
    lmfit/lmfit-6.1.tgz
```

Then, you have to extract and install it as usual. The local installation can be performed using the classical `configure/make/make install` chain, in which you can specify the installation path (`INSTALLATION-PATH` in the following):

```
$ cd lmit-6.1
$ ./configure --prefix=INSTALLATION-PATH
$ make
$ make install
```

N.B.: the artifact has been tested with version 5.1 and 6.1.

Mammut: the library is released via a public GIT repository. The artifact uses the version 0.1. To download it, execute the following command:

```
$ git clone https://github.com/DanieleDeSensi/Mammut.\
    git
$ cd Mammut
$ git checkout tags/v0.1.0
```

To perform a local installation, you have to manually edit the `Makefile` and specify at the line '`export MAMMUT_PATH`' your installation path. After that you can compile and install as usual:

```
$ make -j
$ make install
```

To properly use the library, in some kernels the `msr` module must be loaded[**] by the operating system and the used CPU frequency driver must be `acpi-cpufreq` (default in many kernels up to 3.9). If the module is not loaded, the Mammut library will thrown an exception. You can load it using the root privileges:

```
# modprobe msr
```

The latter condition can be verified using command line utilities like `cpupower` or `cpufreq-info`:

```
$ cpupower frequency-info
```

The `driver` should be `acpi-cpufreq`. If this is not the case, the Mammut library will thrown an exception at program start. Starting from kernel 3.9 the `intel-pstate` is used as frequency driver. To force the kernel to use the `acpi` one you can follow the instruction available at `https://wiki.archlinux.org/index.php/CPU_frequency_scaling`, section "CPU Frequency Driver".

A.3.2 Artifact building

The source code of the artifact is available in the GIT repository:

```
$ git clone https://github.com/tizianodem/elastic-hft.\
    git
$ cd elastic-hft
$ git checkout tags/v1.0
```

For the compilation it is required to set proper environment variables to indicate the libraries installation path. The variables are `FASTFLOW_DIR` for the FastFlow library, `LMFIT_DIR` for the Lmfit library, and `MAMMUT_DIR` for the Mammut library. Furthermore, for the execution it could be required that the path to the Lmfit library is in your `LD_LIBRARY_PATH` environment variable. After that, the code can be compiled. The set of command is the following:

```
$ export FASTFLOW_DIR=<...path to fastflow...>
$ export LMFIT_DIR=<... path to lmfit...>
$ export MAMMUT_DIR=<...path to mammut...>
$ export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:$LMFIT_DIR/\
    lib
$ make -j
```

This will produce a set of binaries:

- `elastic-hft` is the High Frequency Trading application used for the experimental evaluation;
- `synthetic-` and `real-generator`, respectively the data generator for the synthetic dataset and the real one;
- `derive-voltage-table` a utility program, whose scope is cleared in the following section.

A.4 Experiment Workflow

In this section we will describe the principal steps required to evaluate the artifact and reproduce (qualitatively) the results obtained in Sect. 5.3 (*Control Strategies Evaluation*) and Sect.5.5(*Comparison with similar approach*).

Please note that from now on the commands must be launched with root privileges (or sudoer user). This is required by the Mammut library to access to the CPU energy counter and change the CPU frequency. If you launch the commands with `sudo` it could be necessary to export your `LD_LIBRARY_PATH`:

```
$ sudo LD_LIBRARY_PATH=$LD_LIBRARY_PATH <command>
```

A.4.1 Preliminaries

In order to estimate the energy consumption (Sect. 3.1.3), it is required to have the information about the voltage levels of the CPU. To derive this values you can use an utility (`derive-voltage-table`) present in the code repository. It must be run before starting the program for the first time:

```
# ./derive-voltage-table
```

It produces a file *voltages.txt* that contains the approximate values of the voltages and it is used by the artifact (it must reside in the same folder of the binary). More accurate measurements of these values can be obtained using the proper demo in the Mammut Library (under `demo/energy/voltageTable`). However, the approximate version provided by the utility in this artifact is good enough for our scope and requires just a couple of minutes to be executed.

A.4.2 Workflow for the control strategies evaluation

The typical evaluation workflow will require to firstly start the `elastic-hft` application and then the `generator`:

1. start the `elastic-hft` program. It takes in input various parameters:
   (a) the number of stock symbols (keys) that the stream will convey. This is used mainly for statistical purposes. For the datasets that are included in this artifact this value must be equal to 2,836;
   (b) the starting number of replicas. This will be automatically adjusted during the execution according to the scaling strategy used. If this number exceeds the maximum allowed (see below how to derive it) for the used machine, an error is returned at program start up and the execution aborts;
   (c) the TCP port (e.g, `8080`) over which the program will wait for a connection from the `generator`;
   (d) the window size expressed in number of quotes. In the experiments it was set to 1,000;

[**] `http://man7.org/linux/man-pages/man4/msr.4.html`

(e) the window slide expressed in number of quotes. In the experiments it was 25 (it can be changed provided that it is a divider of the window size);

(f) a configuration file that provides a set of configuration parameters of the used scaling strategy (see below for its description).

For example, the following command will launch the program with an initial number of replicas equal to 5:

```
# ./elastic-hft 2836 5 8080 1000 25 \
    path_name_of_the_configuration_file.cfg
```

The program output is described in Sect. A.5.

2. start a `generator`: depending on the chosen dataset you can launch the two different generators provided with the artifact:
   - the `synthetic-generator` for the synthetic dataset. It requires the following command line parameters:
     (a) the hostname (or IP address) of the machine in which `elastic-hft` is executed;
     (b) the TCP port number over which `elastic-hft` is waiting for the connection;
     (c) the number of stock symbols of the dataset: 2,836 in our case;
     (d) a file containing the probability distribution of the various stock symbols;
     (e) the initial rate, expressed in quotes (messages) per seconds;
     (f) the total number of quotes to generate. This determines the execution length;
     (g) optionally, a file that contains the different data generation rate during the execution. If not specified the generator will produce quotes with a fixed rate.

     The *probability distribution file* and the *rates file* used for the experiments with the synthetic dataset (random walk workload, Sect. 5.1) are stored in the `distr_and_rates` folder of the code repository. For example, if you want to start the generator with the same characteristic of the one used for the experiments (assuming that `elastic-hft` is in execution on the same machine of the `generator`), type:

     ```
     #./synthetic-generator localhost 8080 2836 \
         distr_and_rates/probability_distribution \
         300000 54000000 distr_and_rates/\
         random_walk_rates
     ```

     This will produce a quotes stream that will last approximately 180 seconds.
   - the `real-generator`, if you are willing to use the real dataset. In this case you have to specify as command line arguments:
     (a) the hostname (or IP address) of the machine in which `elastic-hft` is executed;
     (b) the TCP port number over which the `elastic-hft` is waiting for the connection;
     (c) the path to the provided dataset (see Sect. A.2.5);
     (d) the number of quotes to generate (up to 49,544,800 for the provided dataset);
     (e) `-s <number>` optional parameter, it specifies how many times the original dataset must be accelerated. If not specified it will run at the original speed, resulting in 6 hours and an half of data generation. For this reason we recommend to accelerate it: in our experiments this throttling parameter was set to 100, resulting in a data generation time of 236 seconds.

     An example of execution is the following:

     ```
     # ./real-generator localhost 8080 ./dataset \
         49544800 -s 100
     ```

We recommend to execute the `generator` and the `elastic-hft` on the same machine mainly for two reasons:

- avoid problems due to network interconnection (e.g., slow data transfer);
- avoid problems due to clock drift in one (or both) the used machines, that may results in wrong latency measurements. [††]

Processes (and relative threads) are automatically pinned on the available physical cores (one per core; Hyper-Threading, if present, is not used). For this reason the *maximum number of replicas that can be used* is equal to the number of physical cores minus 4 (a core is used respectively by the generator, splitter, merger and controller). In any case, if you are willing to execute the generator on a different machine, please add the following macro definition: `-DGENERATOR_ON_DIFFERENT_MACHINE` on the `DEFINES` line of the `Makefile` and recompile (`$ make elastic-hft`)

Configuration file description: the application takes as input parameter a configuration file that details the type of scaling strategy to use. The configuration file is a plain text file and has to respect a proper syntax: lines that begin with a dash (#) are treated as comments, while lines that represent parameters are in the form `<attribute>=<value>` (lower case). To be a valid configuration file for a strategy, the file passed must provide the following parameters:

- `control_step=<value>` this attribute specifies the duration of the control step expressed in milliseconds (e.g., we used 1000). It has to be a multiple of 1000;
- `strategy=<value>` this attribute specifies the scaling strategy to use. Possible values are:
  - `none`: no scaling strategy is used. The application will be always executed with the number of replicas specified in command line argument;
  - `latency`: it is a synonym for the `Lat-Node` strategy. It changes only the number of replicas trying to use the minimum one necessary to provide the latency requirement;
  - `latency_energy`: it is a synonym for the `Lat-Power` strategy. It changes the number of replicas and/or the CPU frequency to respect the latency threshold. It tries to minimize the energy consumption;
  - `spl`: it is the strategy defined in the paper "Elastic scaling for data stream processing". Reference [12] in the paper.
  - `latency_rule`: it changes the number of replicas if the relative error between the measured latency and a required one is over a threshold (Rule-Based in the paper, Sect. 5.5)
- depending on the strategy type other parameters are required:
  - `alpha=<value>`, `beta=<value>`, `gamma=<value>`, `horizon=<value>`: are the parameters described in the paper. The first three represent the cost parameters of the optimization problem described in (Sect. 3.2). They are positive

[††] Some kernels have this problems, e.g., `http://support.ntp.org/bin/view/Support/KnownOsIssues` Sect.9.2.4.2.7

```
Program terminated, received results: 2158649
#Total number of reconfigurations:      37
#Violations wrt the threshold:          2
#Average Number of used replica:        9.657459
#Average Watt consumed:                 60.133566
#Reconfiguration amplitude:             1.027
```

Figure 16: Screenshot reporting the SASO summary printed at the end of `elastic-hft` execution with `Lat-Node` strategy.

```
Program terminated, received results: 2158649
#Total number of reconfigurations:      31
#Adjustments to the number of replicas:  7
#Adjustements to the CPU frequency:      24
#Violations wrt the threshold:          81
#Average Watt consumed:                 49.397050
#Reconfiguration amplitude:             1.000
```

Figure 17: Screenshot reporting the SASO summary printed at the end of `elastic-hft` execution with `Lat-Power` strategy.

float numbers. The latter specifies the length of the prediction horizon: it must be an integer number greater or equal to one. They are required both for `latency`, `latency_energy` strategies;

- `threshold=<value>`: required for the following strategies: `latency`, `latency_energy` and `latency_rule`. It describes the desired latency threshold in millisecond (positive float number);
- `max_level=<value>`, `change_sensitivity=<value>`, `conf_threshold=<value>` are required by the `spl` strategy; `max_level` is the maximum number of replica to use. The other two parameters should be tuned taking into account the suggestions reported by the authors in [12]. The parameters that we have used in the paper can be a good starting point.

You can find all the configuration files used for the experiments reported in Sect. 5.3 and Sect. 5.5 of the paper in the `config_file` folder, organized according to the used dataset. You can modify them in order to create your own configurations, also taking into account the *tuning phase* described in Sect. A.5.3.

A.5 Evaluation and Expected Results

The results must be validated qualitatively with respect to the ones in the paper. In fact, the exact numerical values depend on the hardware configuration. This artifact allows the reviewers to evaluate the experiments performed in Sect. 5.3 (*Control strategies evaluation*) and Sect. 5.5 (*Comparison with similar approach*). For the first ones we compared different MPC-based strategies, in the latter we compared our results to other known scaling strategy.

A.5.1 Control strategies evaluation

In the experiments section we compared our strategies in terms of the SASO properties (i.e. stability, accuracy, settling time, overshoot). For the two datasets we run the application using the different strategies (`Lat-Node` and `Lat-Power`) and we evaluated the results according to various criteria.

During the `elastic-hft` execution, various information are printed in the standard output typically at each second of the execution. In particular:

- in green, the statistics regarding the last execution second: the number of current replicas used, the number of results produced over the last second and the average latency measured (expressed in microseconds);
- in yellow, the reconfigurations (e.g., the variations in the number of replica and/or CPU frequency).

At the end of the execution the so-called *SASO summary* is printed. These results are needed to evaluate and compare the various strategies. Fig. 16 shows the output of the execution of the `Lat-Node` strategy with any configuration: without switching cost (`NoSw`) and the ones with switching cost and different horizon lengths (`h=1`, `h=2`). In particular:

- the total number of reconfigurations performed. In the case of the `Lat-Power` strategy the SASO summary also differentiates the type of reconfigurations. In particular it shows how many reconfigurations have changed the number of replicas and how many reconfigurations have changed the CPU frequency. There are cases in which a reconfiguration can change both the aspects, so the sum is not necessarily equal to the total number of reconfigurations. An example of this SASO summary is shown in Fig. 17. The total number of reconfigurations is meaningful to evaluate the *stability* property. *Better stability requires fewer reconfigurations*;
- the number of latency violations with respect to the specified threshold. This parameter is fundamental for evaluating the *accuracy* SASO property. *Obtaining few violations is an evidence of a better accuracy of the strategy*.
- the average reconfiguration amplitude. This measurement corresponds to the *settling time* SASO property. *A better settling time is achieved with high reconfiguration amplitude values*.
- the average resource consumption of the strategy. This information corresponds to the *overshoot* SASO property. The `Lat-Node` strategy cannot modify the CPU frequency, so the meaningful parameter to evaluate the overshoot is the average number of replicas used. In constrast, the average watt consumption is the main overshoot measure for the `Lat-Power` strategy. For the sake of completeness, we also report the watts consumed per second by `Lat-Node` strategy. In our artifact we measure the core energy counter, while the overall socket consumption is usually 30 watt higher. Finally, it is worth remembering that *the overshoot property is better with lower resource consumption values*.

All the measurements produced by `elastic-hft` (i.e. the number of produced results, average latency, number of replicas used and CPU frequency reported per seconds, and the SASO summary) are saved in a file named `stats.dat`. The generators produce a `generator.dat` file that contains information about the data rate generated at various time instants.

The goal of this artifact is to reproduce the same qualitative behavior of the results shown in the paper. The idea is graphically illustrated in Tab. 4. The best results correspond to three *stars*, whereas the worst ones are denoted by a single star. Two stars represent intermediate results (a good trade-off). Both for the `Lat-Node` and the `Lat-Power` strategy we expect the behavior of the table for each possible strategy configuration (without switching cost or with switching cost and different horizon lengths, e.g., `h=1,2`). The configuration without switching cost is the worst in terms of stability (many reconfigurations are needed), the settling time is the best, the accuracy is the worst (many QoS violations) and, finally, the overshoot is the best, i.e. with this strategy configuration we always use the minimum number of replicas and/or the lowest CPU frequency to meet the latency threshold, therefore violations likely occur if the arrival rate is not accurately predicted. The configuration with switching cost and minimal horizon is the best in terms of stability

(few reconfigurations) with low settling time (small reconfigurations on average). This configuration has a high overshoot, therefore the accuracy is high because we usually oversize the number of replicas/CPU frequency. Finally, the strategy configuration with switching cost and a sufficiently long horizon reaches a good tradeoff between the four SASO properties.

Strategy	*Stability*	*Settling time*	*Accuracy*	*Overshoot*
`NoSw`	★	★★★	★	★★★
`Sw h=1`	★★★	★	★★★	★
`Sw h=2`	★★	★★	★★	★★

Table 4: Qualitative analysis: ★★★ denotes the best results, ★ the worst ones.

This qualitative behavior is expected both for the `Lat-Node` and `Lat-Power` strategy, and makes it possible to reproduce the experiments in Figs. 8, 11, 13 and 14 of the paper, in which each strategy is analyzed by comparing different strategy configurations in terms of the SASO properties.

A.5.2 Comparison with similar approaches

The same qualitative comparison can be achieved by executing the two other strategies (`spl` and `latency_rules`). At the end of the execution the SASO summary is printed. It is possible to achieve results qualitatively similar to the one of Tab.3 of the paper.
Please note that for the *spl* strategy the SASO summary, printed at the end of program execution, does not have information about latency threshold violations. This because this strategy does not take in input this parameter to drive its reconfigurations. For producing the results shown in Table 3 of the paper, we have counted the violations considering the latency values reported in the stats.dat files produced by elastic-hft. To this purpose a simple script is included in the code repository (see Sect. A.5.4).

A.5.3 Tuning phase

The results of the artifact greatly depend on few parameters that are critical and require a certain tuning effort. In particular, for the `Lat-Node` and the `Lat-Power` strategies it is required to choose a proper *threshold* for the latency in the configuration file. If the threshold is too low, this will result in too many QoS violations since the performance constraints are excessive for the underlying hardware. If it is too high, the application is able to sustain the peak rate with few replicas and the difference between different strategy configurations becomes minimal. Therefore, finding a good threshold is fundamental to increase the quality of the results and to make the previously described qualitative behavior more evident. Heuristically, we found that a good threshold must be able to produce few reconfigurations, e.g., $5 \div 7$ with the `Lat-Node` strategy (`Sw h=1`) and the synthetic generator. To facilitate the user to find a good threshold, we propose to use the following two scripts. They should be invoked from the `elastic-hft` folder:

- for the synthetic workload:

```
# bash scripts/calibrate_threshold_synthetic.sh
```

- for the real workload:

```
# bash scripts/calibrate_threshold_real.sh <path \
    to dataset>
```

They will run `elastic-hft` several times (with a number of replicas fixed to 4) with different thresholds until a threshold that respects the conditions described before is found (for a maximum of 10 attempts). This value can be used in the configuration files of the strategies. This value is a guess, thus the user has to adjust it a little bit.

In the unfortunate case of not being able to find a good threshold, it is possible to slightly decrease the *alpha* parameter in the configuration file of the strategy that we want to execute. The rationale is that with a higher *alpha* the optimization gives more weight to the performance, thus fewer violations are expected. Slightly smaller *alpha* values lowers the weight of the performance part of the cost function, and more violations are expected.

A.5.4 Scripts

Different runs of the artifact, though executed in the very same conditions, could result in slightly different results. For this reason the results reported in the paper have been derived by running the tests multiple times (25 in our case) and by collecting the average results for the various strategies and workload traces.

In the `scripts` folder of the repository there are two scripts (`conf_interval_synthetic.sh` and `conf_interval_real.sh`, respectively for the synthetic and real workload; the latter takes as command line arguments the path to the dataset) that automatize this procedure. Both of them take in input the set of configuration files in the `configs` folder (you have to create and populate it) and for each of them they launch the artifact multiple times (by default 5) with the proper generator (synthetic or real). The results are collected in the `results` folder according to the strategy configuration file used. In particular, you can find the following information:

- the statistics of each run (numbered from 1 to N where N is the number of run, e.g., 5);
- a file `avg_metrics.dat` that will contain the overall SASO summary, with the average values collected in the various runs. These can be helpful for comparing the various strategies as we did in Figs. 8, 11, 13 and 14;
- two files `conf_interval_latency.dat` and `conf_interval_watt.dat` that contain for each second of the execution the average values of the latency and power consumption, averaged among the various runs. They contain the information for displaying the confidence interval at the 95% with tools like gnuplot or similar. For example, if you want to print it in gnuplot:

```
gnuplot> plot "conf_interval_latency.dat" u \
    1:2:3:4 with yerrorline title "Conf Interval \
    95"
```

  In this way, by comparing the different strategies, it is possible to reproduce the plots in Fig. 10 and 12.

The default values of the scripts (e.g., the number of runs or the directory that will contain the results) can be changed by editing the scripts.

As mentioned before, the *spl* strategy does not produce in output the number of latency threshold violations and he number reported in the overall SASO summary (`avg_metrics.dat` file) is equal to zero. To compute this value (e.g. to reproduce Tab. 3), you can use the `count_violations.sh` script contained in the `scripts` folder. It takes in input a folder containing the `stats.dat` produced by the program execution (e.g. by using the `conf_interval_*` scripts) and the threshold values in micro seconds. For example:

```
$ bash scripts/count_violations.sh results/spl/ 7000
```

In output it produces the average number of violations.

A.6 Final notes

- Thread pinning is automatically applied assuming that the core IDs are numbered starting from zero. Usually this is the case of many machines. To cover all the situations we will provide in the future an easy way to let user specifies its own affinity setting (e.g., via a configuration file);
- If you want to save all the results in text files (i.e. the fitting polynomial and other info useful for producing live statistics), you can add the macro definition `-DSAVE_RESULTS` to the `DEFINES` line in the `Makefile` and recompile the program;
- If you are willing to define another rates distribution to be passed to the `synthetic-generator`, you have to create rate file with the same syntax of `distr_and_rates/random_walk_rates`. In particular in the first row there is the number S of seconds the stream has to last. Then there are S rows, one per seconds, indicating the second number (starting from zero) and the number of tuples to produce in that second.