Availability and Reproducibility

AutoMon: Automatic Distributed Monitoring for Arbitrary Multivariate Functions
https://doi.org/10.1145/3514221.3517866

Hadar Sivan (✉) Moshe Gabel Assaf Schuster

AutoMon is a library for evaluating a mathematical function over the average of multiple vectors that are distributed over multiple nodes of a system. Given Python code that computes a function $f(x)$ from vector x, as well as the desired approximation error, AutoMon automatically provides communication-efficient distributed monitoring of the function approximation over the aggregate (average) of distributed vectors, without requiring any manual mathematical analysis by the developer.

This document explains in detail how to reproduce our paper's experimental results and how one could easily use AutoMon for additional monitoring tasks.

For assistance with the evaluation or with any questions, please feel free to contact hadarsivan@cs.technion.ac.il.

1 Availability

AutoMon is available as an open-source project on GitHub: https://github.com/hsivan/automon under a BSD 3-clause license. The repository is well-documented to make it accessible for new users, as well as to developers who wish to contribute to the project. The README file includes usage example, which shows how one could easily use AutoMon to evaluate any user-defined function on any data. It also includes a script to reproduce our experiments and to generate the paper's figures from the experiments results.

2 Reproducibility

We provide full reproducibility of all our experiments, by simply running a single script and with minimal installation effort. AutoMon's repository includes the script `reproduce_experiments.py`.[1] This script does the following:

1. Fetches AutoMon's source code from its repository, or (if run more than once) merges updates from the repository to existing local AutoMon project.
2. Downloads external datasets, ∼150MB.
3. Re-runs the main group of experiments locally, which takes about 2.5 days (measured on an Intel i9-7900X at 3.3GHz with 64GB RAM, running Ubuntu 18.04). Experiments that have previously finished are skipped (see Error Handling below).
4. Optionally, re-runs the AWS subset of experiments (see below), which takes about 2.5 more days.
5. Generates all figures from the experiment results.
6. Recompiles the paper from Latex source code and produces a new PDF that contains the new figures.

Machine Configuration The system requirements for running the script are:

1. An x86-64 machine running Ubuntu 18.04 or later.

[1] https://github.com/hsivan/automon/blob/main/reproduce_experiments.py

2. At least 16GB of RAM.
3. At least 8 logical cores.
4. Python version ≥ 3.8.
   $\Longrightarrow$ Make sure `python` command is aliased to this python version.
5. Root access (`sudo` permissions).
6. Docker engine (see https://docs.docker.com/engine/install/ubuntu.
   $\Longrightarrow$ Make sure running `sudo docker run hello-world` works as expected).
7. At least 15GB free storage space before running the `reproduce_experiments.py` script.

We tested the script on local machines running Intel i9-7900X at 3.3GHz with 64GB RAM and 20 logical cores and Intel Xeon E5-2420 at 2.20GHz with 16GB RAM and 24 logical cores, as well as on an c5.4xlarge Amazon EC2 instances, all running Ubuntu 18.04.

Running the Script Download the `reproduce_experiments.py` script file; there is no need to clone the entire project as the script works as a standalone. We will refer to the local folder where the script is located as `./` below. From withing the `./` folder run: `python reproduce_experiments.py [--aws]` (see below regarding the optional `--aws` flag). The script runs all the experiments and all the reviewer needs to do is to wait for it to finish and then verify the results (explained below).

The Log File The `reproduce_experiments.py` script logs important information to the file `./reproduce_experiments.log`. The log file indicates the current running experiment and provides runtime estimation for each experiment. Note that estimated runtimes for the local experiments are based on measurements on an Intel i9-7900X at 3.3GHz with 64GB RAM, running Ubuntu 18.04. In addition, at the end of the run, the script writes to the log the list of reproduced figures, and explicitly indicates which figures were replaced by the reproduced ones and which were not. For example, if the reviewer did not use the `--aws` flag, this will be reflected in this list of figures.

Distributed Experiments on AWS A subset of our experiments deployed AutoMon on a real-world distributed network using AWS services (Section 4.7 in the paper). The script `reproduce_experiments.py` supports running these experiments as well by adding the optional `--aws` flag:
`python reproduce_experiments.py --aws`.

Running these experiments requires the reviewer to have an AWS account. After opening the account, the reviewer must create an AWS IAM user with AdministratorAccess permissions:

1. In https://console.aws.amazon.com/iam select "Users" on the left and press "Add users".
2. Provide user name, e.g. automon, and mark the "Access key" checkbox and press "Next: Permissions".
3. Expand the "Set permissions boundary" section and select "Use a permissions boundary to control the maximum user permissions". Use the Filter to search for AdministratorAccess and then select it from the list and press "Next: Tags".
4. No need to add tags so press "Next: Review" and in the next page press "Create user".
5. Press "Download.csv" to download the new_user_credentials.csv file.
6. Run `python reproduce_experiments.py --aws` once (if haven't already) to download the automon project. This script will download the project to `./automon_main`. (The script can be called multiple times without overriding existing experiment results.)
7. Put the new_user_credentials.csv file in `./automon_main/aws_experiments/` folder.

After completing these steps, re-run `reproduce_experiments.py` with `--aws` flag. The script will set up the required infrastructure in AWS and deploy the experiments over this infrastructure. The results are collected in an AWS S3 bucket and the script collects these results to the local folder `./automon-main/test_results/`. The reviewer just needs to wait to the script to finish.

Warning: The AWS experiments run for multiple days and would likely cost a few hundred dollars ($\sim$500$)!

AWS experiments use at most 10 c5.4xlarge EC2 instances simultaneously and at most 100 ECS Fargate tasks simultaneously. These instances automatically terminate at the end of each experiment. We also use one S3 bucket to collect the results and one ECR repository to store our Docker image. We include a cleanup script, described below, to help the reviewer release the resources once experiments are done. More details regarding the AWS system can be found in https://github.com/hsivan/automon/blob/main/aws_experiments/README.md, though they are not required to reproduce our experiments.

Output Files and Verification of Reproduced Results After the script `reproduce_experiments.py` ends, assuming with no errors, the reviewer will find the following outputs:

1. `./main.pdf`: the reproduced paper containing the new figures. It should look similar to the original paper.
2. `./automon-main/test_results/*.pdf`: the reproduced figures. See Table 1 for a list of reproduced figures and their mapping to figures/text in the paper. Reproduced figures use the same style and are expected to be similar to those in the paper, perhaps with slight differences due to hardware/software differences or non-deterministic behaviour (e.g., network).
3. `./automon-main/test_results/results_*`: result folder for each local experiment, which includes raw data from which the figures are generated.
4. The `./automon-main/test_results/*_aws_*`: result folder for each AWS experiment.

Error Handling In case the script stopped before completion, whether manually or due to a failure that was later fixed, the reviewer can simply re-run the `reproduce_experiments.py` without losing the results of previously completed experiments. The script simply skips completed experiments (experiments with a result folder that indicates a normal experiment completion) to save time. In case of error, the script stops and the error is reflected in the standard output as well as in `./reproduce_experiments.log`. If the script ended successfully, the last last line in both the standard output and the log file is: `The reproduced paper, containing the new figures based on these experiments, is in ./main.pdf`.

Cleanup All experiments run inside Docker containers; all needed packages are installed inside the container which leaves the local system clean. The only exceptions to this are:

1. TexLive: this tool is needed for the compilation of the paper's Latex source files and is installed by the script. The reviewer may remove it after the script ends by running `sudo apt-get remove texlive-*`.
2. Release AWS resources: after running the AWS experiments with the `--aws` flag, the reviewer should run the script `aws_cleanup.py` to release AWS resources allocated for our distributed experiments (S3 bucket, ECR repository, CloudWatch log groups, ECS clusters, and EC2 instances). This cleanup step is important to avoid unnecessary future charges; however, it should be run only after verifying that the reproduced paper `./main.pdf` is generated and that the reproduced figures are similar to the original paper's figures (See Table 1).
3. AWS CLI: in case `--aws` flag was used, this tool is installed by the script. The reviewers may remove the tool by running `sudo rm /usr/local/bin/aws ; sudo rm /usr/local/bin/aws_completer ; sudo rm -rf /usr/local/aws-cli`. Removing the cli tool should be done only after cleanup step #2.

3 Replicability

AutoMon's repository was explicitly designed to help future researchers and users to evaluate and use AutoMon. To that end, the repository is well documented and includes different usage examples. In particular, we include two levels of examples.

Basic Usage For researchers and practitioners who wish to use AutoMon to monitor their own defined function over their own data, we provide a simple usage example, which shows how one could easily

use AutoMon to evaluate any user-defined function on any data: https://github.com/hsivan/automon#usage-example. The user can use this example as a template, replacing `func_inner_product` with its own function definition and replacing our synthetic data generator `data = np.random.normal(...)` with other data generator (e.g., with csv file reader), to monitor different functions and data using AutoMon.

Extended Usage For those who wish to dive deeper, we provide additional examples of using AutoMon and comparing it to different state-of-the-art baselines in distributed monitoring. These examples are intended for future researchers who would like to compare AutoMon to other algorithms, or would like to extend and improve it. These extra examples are found in the `example` folder. It contains experiments that compare AutoMon to other baselines for many functions, including functions that we did not include in the paper such as entropy `compare_methods_entropy.py` and variance `compare_methods_variance.py`.

Platform Support Finally, AutoMon is designed as an easy-to-use algorithmic building block, and as such, it was designed to also support different operation systems. AutoMon could be installed on both Linux and Windows, with the limitation that JAX is not fully supported on Windows, and therefore AutoMon uses AutoGrad when installed on Windows. See https://github.com/hsivan/automon#installation for more details.

Table 1: Reproduced figures under `./automon-main/test_results/` and their mapping to the paper's text/figure. Reproduced figures are expected to be similar to the figures in the paper, perhaps with slight differences due to hardware differences or randomness in some experiments. Note: Figures 1 and 2 are not based on experimental results, and therefore are not included in this list.

PDF file	Figure in paper	Description
impact_of_neighborhood_on_violations_three_error_bounds.pdf	Figure 3	See paper. Figure was extracted from Section 4.5 experiment results.
function_values_and_error_bound_around_it.pdf	Figure 4	See paper.
max_error_vs_communication.pdf	Figure 5	See paper.
percent_error_kld_and_dnn.pdf	Figure 6	See paper.
dimension_communication.pdf	Figure 7(a)	See paper.
num_nodes_vs_communication.pdf	Figure 7(b)	See paper.
dimension_coordinator_runtime.pdf	Not in paper	Results are explained in "Coordinator Runtime" in Section 4.4.
dimension_node_runtime.pdf	Not in paper	Results are explained in "Node Runtime" in Section 4.4.
dimension_node_runtime_in_parts.pdf	Not in paper	Results are explained in "Node Runtime" in Section 4.4.
neighborhood_impact_on_communication_error_bound_connection.pdf	Figure 8	See paper.
optimal_and_tuned_neighborhood.pdf	Not in paper	Results are explained in the 3rd paragraph of Section 4.5.
monitoring_stats_quadratic_inverse.pdf	Figure 9(a)	See paper.
monitoring_stats_barchart_mlp_2.pdf	Figure 9(b)	See paper.
func_val_and_approx_error_quadratic_inverse.pdf	Not in paper	$f(x) = -x_1^2 + x_2^2$ value for the data used in Section 4.6.
Figures for AWS Experiments (Section 4.7)		
max_error_vs_transfer_volume.pdf	Figure 10(top)	See paper.
communication_automon_vs_network.pdf	Figure 10(bottom)	See paper.
communication_automon_vs _network_*.pdf	Not in paper	Similar to Figure 10(bottom), but include all tested error bounds per function.
max_error_vs_communication_sim_and_aws_*.pdf	Not in paper	Results are explained in "Number of Messages" in Section 4.7.