Reproducibility Report for ACM SIGMOD 2020 Paper: “Functional-Style SQL UDFs With a Capital ‘F’ ”

DMYTRO BOGATOV, Boston University, United States

The work on reproducibility is exemplary. All software components are packaged in a platform-independent Docker image. A single command-line entry point allows to interactively recreate all plots and tables. The absolute values measured in the reproduction are close enough to the values reported in the paper, all relationships are maintained.

1 INTRODUCTION

This is a reproducibility report for a paper by Duta and Grust [1]. To summarize, all figures and tables have been reproduced accurately enough. The reproducibility scripts is easy to use, intuitive and interactive.

2 SUBMISSION

Reproducibility submission consists of the detailed instructions on how to rerun the experiments and a `Makefile` that acts as a command-line entry point for the reviewer. `Makefile` runs a single script inside the Docker image. The script interactively walks the reviewer through the figures and tables that can be generated. A single experiment results in producing a PDF of a plot or a text file with a row of a table.

The submission contains:

- Docker image: `christianduta/functional-style-udf-experiments`
- A `Makefile` is included in the submission, but not available publicly
- A PDF with detailed instructions is included in the submission, but not available publicly

3 HARDWARE AND SOFTWARE ENVIRONMENT

Table 1 describes the resources the original paper and the reproducibility reviewer used to run the experiments.

Table 1. Hardware & Software environment

	Paper	Repro Review
Platform	64-bit Linux x86	macOS 11.1 Docker 3.1.0
Machine	Unknown	MacBook Pro (15-inch, 2016)
CPU	Intel Core i7	Intel Core i7
Cores	8	4
GHz	3.66	2.7
RAM	64GB	16GB

4 REPRODUCIBILITY EVALUATION

4.1 Process

All experiments have been run using the same method — executing `make run` followed by the interactive choice of the particular scenario. For verifying the empirical results, only the longer versions of the experiments were run. The option for faster and less accurate runs was nevertheless

tested (and is appreciated). I would note that the actual running time appeared to be significantly longer than the displayed estimate.

It was possible to follow the reproducibility instructions without authors' help.

4.2 Results

See reproduced artifacts in Appendix A.

Figures. The following figures have been reproduced: `Figure 3`, `Figure 7`, `Figure 20a`, `Figure 20b`, `Figure 22` and `Figure 23`. The obtained numbers and the visual plots appear to be close enough to the reported values in the paper [1]. The deviation is attributed to the differences in hardware and a platform. Most importantly, the relationships between performances of different algorithms matches the ones reported and discussed in the paper. See Figure 1.

Tables. The reproduced absolute values for `Table 2` experiments expectedly differ from the ones reported in the paper, but the relations are consistent. The `Average Call time: UDF` values are bigger (i.e., execution is slower) due to using different platform, but the gain (the percentage in `Table 2`) is close to the reported value. In fact, the reproduced gain is on average even better (i.e., compiled version is relatively even faster) than reported, which fully supports authors' claims. See Table 2.

Interactive prompt. The examples from the reproducibility submission have been successfully run in the supplied interactive prompt. After changing a few numbers in the snippet, the code still executes correctly.

5 SUMMARY

All major (and minor) figures and tables have been reproduced. The ideas, claims and findings supported by these figures are therefore reproduced as well.

The work on reproducibility is exemplary: the code is trivial to run, the interactive console UI guides the user and the figures are generated as vector PDFs.

REFERENCES

[1] Christian Duta and Torsten Grust. 2020. Functional-Style SQL UDFs With a Capital 'F'. In *Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data* (Portland, OR, USA) *(SIGMOD '20)*. Association for Computing Machinery, New York, NY, USA, 1273–1287. https://doi.org/10.1145/3318464.3389707

A REPRODUCED ARTIFACTS

Fig. 1. Reproduced figures

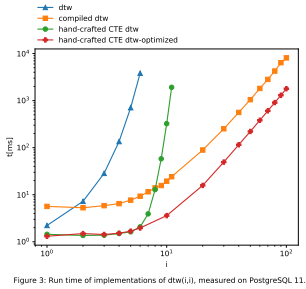

Figure 3: Run time of implementations of dtw(i,i), measured on PostgreSQL 11.3.

(a) Figure 3

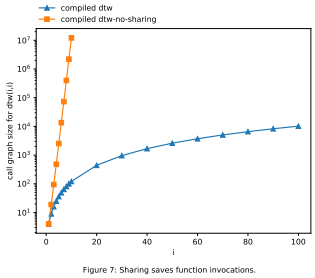

Figure 7: Sharing saves function invocations.

(b) Figure 7

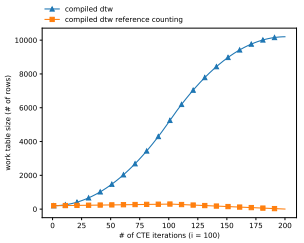

Figure 20 a): Evaluating dtw(i,i): Impact of reference counting on work table size.

(c) Figure 20a

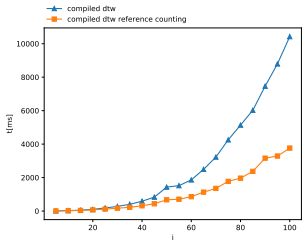

Figure 20 b): Evaluating dtw(i,i): Impact of reference counting on CTE run time.

(d) Figure 20b

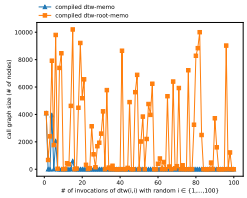

Figure 22: Series of dtw(i,i) invocations: Re-using memotable entries effectively cuts down call graph size.

(e) Figure 22

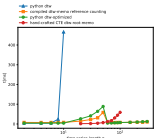

Figure 23: In-database vs. external processing: average evaluation time of dtw(i,i) with random i in {1, ...,n} (HDBMS: PostgreSQL 11.3, ext. processor: Python 3.7).

(f) Figure 23

Table 2. Reproduced Table 2

UDF	Call Time				Gain	
	Reproduced		*Original [1]*			
	UDF	Compiled	*UDF*	*Compiled*	Reproduced	*Original [1]*
comps	1142 ms	20 ms	*357 ms*	*26 ms*	1.7 %	*7.2 %*
eval	668 ms	17 ms	*216 ms*	*20 ms*	2.6 %	*9.2 %*
floyd	>8000 ms	8 ms	*>8000 ms*	*14 ms*	>0.1 %	*>0.18 %*
fsm	1593 ms	201 ms	*659 ms*	*102 ms*	12.6 %	*15.4 %*
lcs	2596 ms	13 ms	*756 ms*	*30 ms*	0.5 %	*3.9 %*
mandel	874 ms	64 ms	*280 ms*	*27 ms*	7.3 %	*9.6 %*
march	1839 ms	12 ms	*742 ms*	*28 ms*	0.6 %	*3.7 %*
paths	673 ms	70 ms	*474 ms*	*46 ms*	10.4 %	*9.7 %*
sizes	212 ms	82 ms	*144 ms*	*67 ms*	38.5 %	*46.5 %*
vm	472 ms	7 ms	*207 ms*	*9 ms*	1.5 %	*4.3 %*