Reproducibility Report for ACM SIGMOD 2021 Paper: "Tuplex: Data Science in Python at Native Code Speed"

LUCAS WOLTMANN, AXEL HERTZSCHUCH, WOLFGANG LEHNER, Technische Universität Dresden, Database Research Group, Germany

This paper documents the reproducibility process of the novel data analytics framework *Tuplex* for Python presented in *Tuplex: Data Science in Python at Native Code Speed* [1] at SIGMOD'21.

1 INTRODUCTION

The reproduced code is part of the paper *Tuplex: Data Science in Python at Native Code Speed* by Leonhard Spiegelberg (Brown Univeristy), Rahul Yesantharao (MIT), Malte Schwarzkopf (Brown University), and Tim Kraska (MIT) [1]. The paper introduces a data analytics framework that compiles Python user-defined functions (UDFs) using a just-in-time approach to optimized native code. The main contribution is a *dual-mode execution model* where frequently occurring common function and compiler patterns are reused through a common fast execution path. Code paths, not matching the common structures or producing exceptions, are handled with an exception resolver using slower user-defined code or slower general purpose routines of the Python interpreter. Therefore, the authors claim that any Python UDF will obtain the maximum possible optimization while not compromising any code expressiveness.

2 SUBMISSION

The reproducibility submission is very extensive by providing a Docker container, a Python control script for running and plotting the benchmarks, as well as a data repository. The docker container can be obtained via the official docker repositories (tuplex/benchmark) or can be built from scratch. The Python control script delivers a full command-line toolbox to control the execution of the benchmarks and plotting of their results. The data itself is stored on Google Drive. For data privacy reasons, the data is password protected. The password was provided to the reviewers and chairs via CMT. Additionally, the authors provide a very detailed `README`. The extensive comments and instructions within the `README` cover the complete process for reproducing the paper's results. This includes, but is not limited to, the download of the data/code/container, the setup of the framework, the invocation of the benchmarks, and the generation of the plots and tables in the paper.

Therefore, this work comes close to an ideal reproducibility submission, by providing all the necessary materials for the reviewers and giving very detailed instructions. Unfortunately, the reproducibility process was very time-consuming and drawn-out due to some errors in the code, erroneous test environment configurations, and long communication round-trip times. We will elaborate on this in Section 4.1.

A summary of all submission sources is presented as:

- code repository: https://github.com/LeonhardFS/tuplex-public/tree/sigmod-repro/benchmarks/sigmod21-reproducibility
- data sources (password protected): https://drive.google.com/uc?id=1chJncLpuSOPUvlWwODg_a7A-sEbEORL1
- `README`: https://github.com/LeonhardFS/tuplex-public/blob/sigmod-repro/benchmarks/sigmod21-reproducibility/README.md

3 HARDWARE AND SOFTWARE ENVIRONMENT

All experiments were conducted under the following conditions.

Table 1. Hardware & Software environment

	Paper	Reprod. Review
CPU	Intel Xeon Platinum 8259CL	AMD EPYC 7513
cores	32 (no hyperthreading, 16 physical cores)	16
GHz	2.5	2.6
RAM	256GB	256GB
Storage	180GB Network Storage	200GB Network Storage
OS	Ubuntu 20.04 LTS	Ubuntu 20.04 LTS
GCC	10	10
Python	3.6	3.6

4 REPRODUCIBILITY EVALUATION

4.1 Process

Even though the submission was well prepared, the actual reproducibility process required a lot of additional work from both parties. In the first step, the environment and docker container had to be set up on the target hardware. This worked without any major problems, even though the authors recommended an AWS instance that was not available to the reviewers thus we had to rely on similar compute instances of TU Dresden's local cloud infrastructure. Minor errors were showing up causing blockings and interruptions of the reproducibility process, mostly consisting of erroneous path settings in the Python code. Those were identified by the reviewers and quickly and comprehensively addressed in repository updates by the authors. After the successful setup of hardware and data, the actual benchmarks were executed. There, a whole range of general problems with reproducibility was discovered. Major errors, like segmentation faults, could not be interpreted by the reviewers and stalled the process. This was accompanied by the partial lack of descriptive logs produced by the test environment. Therefore, the reviewers had to request a bug fix for the logging and then re-run all benchmarks once more. Re-running was especially time-consuming because the whole benchmark setup took close to 48h per run. Therefore, every single run to test if there were still errors left in the execution took hours or days. However, in the end, all major errors were finally addressed in code updates provided by the authors.

Lastly, we want to highlight two aspects that caused a general problem when reproducing the results of the paper. We would like to stress that this occurred in spite of the eager cooperation of the authors. Firstly, communication had long round-trip times, partially due to time zone restrictions and extensive code reworks that required some time to be implemented. Surprisingly, the most complex part in our case was to get the plotting and thus the generation of the figures to work. However, fixing all the other errors in the code not concerning the figures took some time as well. Secondly, reproducibility was and is a complex and iterative process, which became clear in this project due to the very long run times of the experiments. Again, the duration of the experiments was no fault of the authors but reflects the general complexity of the benchmarks. Relying on AWS, the many (partial) runs of the experiments, which were - thankfully - executed on the TU Dresden cloud infrastructure, would have caused significant cost.

4.2 Results

The paper includes nine figures and two tables in its evaluation. We were able to reproduce the run times for ten out of these eleven elements. All experiments have slightly faster run times on our hardware than in the original paper but the multiplicative factors (speed ups) between the

different measurements are approximately the same and thus confirm the paper's main message. Interestingly, for Figure 5, the run times are ten times faster on our hardware; again the ratios between the different run times are the same and thus underpinning the general statement of the original paper.

The lambda experiment with Spark and its corresponding statistics (Table 4 in the paper) could not be reproduced because of a major version upgrade of Spark. We do not consider this a problem because major code-breaking updates are always a source for trouble and this limitation was communicated by the authors before the reproducibility process started.

In summary, our evaluation shows the full reproducibility of the paper and its claim to provide the fastest data analytic framework with just-in-time compiled Python UDFs.

Verdict: All in all, we highly recommend the acceptance of the paper *Tuplex: Data Science in Python at Native Code Speed* for the SIGMOD reproducibility track.

5 SUMMARY

We would like to thank the authors for their continuous precise work on the reproducibility of their code. We enjoyed the cooperation and open communication. For the chairs, we want to point out that a time zone-aware matching between authors and reviewers would drastically improve the communication between the parties and thus help on the practical side to minimize the additional effort of reproducibility.

REFERENCES

[1] Leonhard Spiegelberg, Rahul Yesantharao, Malte Schwarzkopf, and Tim Kraska. 2021. *Tuplex: Data Science in Python at Native Code Speed.* Association for Computing Machinery, New York, NY, USA, 1718–1731. https://doi.org/10.1145/3448016.3457244