PROCEEDINGS OF SCIENCE

Porting DIRAC Benchmark to Python3: impact of the discrepancies and solutions

Alexandre F. Boyer,[a,b,*] **Imane Iraoui,**[c] **Federico Stagni,**[b] **Christophe Haen**[b] **and David R.C. Hill**[a]

[a] *Université Clermont Auvergne, Clermont Auvergne INP, CNRS, Mines Saint-Etienne, LIMOS, Clermont-Ferrand, France*

[b] *European Organization for Nuclear Research, Meyrin, Switzerland*

[c] *Al Akhawayn University, Ifrane, Morocco*

E-mail: alexandre.franck.boyer@cern.ch, iraoui.imane@gmail.com, federico.stagni@cern.ch, christophe.haen@cern.ch, david.hill@uca.fr

To progress in their research, scientific communities generally rely on shared computing resources aggregated into clusters. To provide fair use of the computing resources to every user, administrators of these clusters set up Local Resources Management Systems. They orchestrate the scientific workloads and the resources by allowing a given workload to be executed for a certain time on a definite number of CPUs or machines. To maximize the use of the computing resources and avoid running out of time, users may assess their environments by executing fast CPU benchmarking solutions such as DIRAC Benchmark. Developed in Python 2 in 2012, DIRAC Benchmark has been mainly and successfully employed in the context of the LHCb experiment. Now that Python 2 is deprecated, DIRAC Benchmark has to be ported to Python 3. This paper describes this transition, the impact brought by the changes, and the considered solutions. The main contributions of this paper are: (i) various methods to improve and maintain the program, such as unit tests, a Continuous Integration and Delivery pipeline; (ii) a comprehensive analysis of the discrepancies. The problems were addressed by applying a constant factor on the scores depending on the underlying CPU model and Python version used.

41st International Conference on High Energy physics - ICHEP2022
6-13 July, 2022
Bologna, Italy

*Speaker

PoS(ICHEP2022)980

1. Introduction

In this paper, we focus on DIRAC Benchmark, a fast CPU benchmarking tool developed by the LHCb collaboration in 2012[1]. DIRAC Benchmark has been primarily created to estimate the correlation between the power of a given CPU and the execution of Gauss [2], a Monte-Carlo simulation application replicating collisions occurring in the LHCb detector.

The tool, written in Python 2, has worked fine for several years but requires a major update now. Indeed, Python 2 was declared deprecated in January 2020 [3] and, therefore, DIRAC Benchmark has to be ported in Python 3. While most of these portability changes are minor, some optimizations of the code might lead to consequent issues in the benchmark tool that could lead to wrong CPU power estimations. This paper describes the work done to port DIRAC Benchmark to Python 3, evaluates the impact of the changes and provides solutions to correct the CPU power estimations.

2. CPU Benchmarking tools in High Energy Physics

Valassi et al. propose a comprehensive review of CPU benchmarking tools in High Energy Physics [4]. They address their evolution and limitations in this context. Developers from the LHCb experiment have specifically studied DIRAC Benchmark. Stagni and Charpentier developed elastic Gauss jobs depending on DIRAC Benchmark estimations [5]. Charpentier also noticed issues with HS06 and proposed an update of DIRAC Benchmark in 2017 to better fit with the LHCb workload [6].

3. Updating DIRAC Benchmark

We need to make sure that the benchmark will still provide valuable and accurate scores in Python 3 and beyond. Indeed, developers of Python 3 have included numerous optimizations and changes compared to Python 2. We also need to provide a portable package so that users can directly import the benchmark in their applications.

The first step consists in providing tools to maintain a clean and well-tested code before the transition. We reformatted the code using Black, a style guide auto-formatter for Python Code compliant with Python Enhancement Proposals 8 (PEP8) [7]. We also developed unit tests with pytest [8] to cover the program. Finally, we set up a continuous integration (CI) pipeline based on GitHub Actions [9] to better handle future changes in the code. To help developers to produce quality code, we also included a pre-commit [10] module in the repository. It automatically formats the code using Black and performs several checks before pushing the code to the repository.

The second step introduces the transition to Python 3.9. The program only relies on standard Python libraries and contains about 300 lines of code. Changes were minimal but generated discrepancies in the results that are going to be analyzed in Section 4.

Finally, the last step of the transition consists in creating a Python package to embed the benchmark and make it portable and easy to import. We completely revisited the structure of the project by splitting the command line interface options from the code related to the benchmark and adding configuration files to set up a package. We added a continuous delivery (CD) module to the GitHub Actions pipeline of the project. The CD module, triggered only when a GitHub tag

is created, generates a binary wheel as well as a source tarball from the repository and publishes the files to PyPI [11], a recognized repository of Python software. We also added the package to Conda-Forge [12], a repository of Conda recipes, to ease the integration of the DIRAC Benchmark in other projects.

4. Comparing the Python 3.9 version of DIRAC Benchmark with the Python 2.7 version

We studied the potential impact of the discrepancies on various CPU models by leveraging the computing resources of WLCG. We experimented by submitting workloads containing both versions of the DIRAC Benchmark on 102 grid sites involving 83 different CPU models. Programs, resources, results and figures are publicly available to facilitate the reproducibility of this work [13].

Based on the results of this experiment, we notice that Python 3.9 scores are mostly higher than Python 2.7 scores. The gap seems to be almost constant across the scores compared. Yet we can still distinguish two different cases: scores coming from AMD processors would need a stronger correction than the scores coming from Intel processors. The root mean square error of the Python 3.9 scores with respect to the Python 2.7 scores is about 2.86.

To correct the Python 3.9 version of the tool, we designed a function applying a scaling factor to a score based on the underlying architecture and Python version: 0.86 for Intel processors and 0.71 for AMD processors. The root mean square error of the transformed Python 3.9 scores with respect to the Python 2.7 scores is about 1.19.

The solution was successfully applied to the LHCbDIRAC production environment. Nevertheless, it implies that developers would conduct similar experiments each time a new CPU architecture or Python version is introduced.

5. Comparing the Python 3.9 version of DIRAC Benchmark to the LHCb Monte-Carlo simulations

We also compared DIRAC Benchmark scores to Gauss jobs running on WLCG. It is worth mentioning that this experiment took place after the introduction of Python 3.9 in the LHCb production environment, Python 2.7 being not used anymore. The objective is to ensure the benchmark is still adapted to the current workload or to correct it if need be. We experimented by getting the list of the active Monte-Carlo productions. We extracted the attributes and parameters of 4955 Gauss jobs bound to 75 productions. Jobs ran on 98 distinct CPU models across 55 sites. Programs, resources and results are publicly available to facilitate the reproducibility of this work [14].

Based on the results of this experiment, we notice that DIRAC Benchmark underestimates the capabilities of the AMD processors when dealing with Gauss jobs. Disabling the corrections introduced in section 4 provides better results on AMD CPUs but slightly less accurate results on Intel CPUs. Given the current state of WLCG, which is mainly composed of Intel CPUs, disabling the corrections of the benchmark would not allow LHCbDIRAC to better exploit allocations. Thus, we chose to keep applying the corrections.

6. Conclusion

In the last years, computing infrastructure and funding models have significantly changed, and national science programs have consolidated computing resources and encouraged using cloud systems as well as supercomputers. While supercomputers offer a significant amount of resources, they are highly heterogeneous architectures, pose higher integration challenges and have not been operated continuously for LHC experiments workload processing. HEP communities are adapting their workloads to run on these resources including non-x86 CPUs and accelerators, such as GPUs and FPGAs. DIRAC Benchmark has not been conceived for such use cases and would need an important update. One of the solutions would be to study the development of the HEP-Benchmarks suite [4] and propose a new fast benchmark solution based on it.

References

[1] DiracGrid, *source code of the dirac benchmark 12*, 2017.

[2] M. Clemencic, G. Corti, S. Easo, C.R. Jones, S. Miglioranzi, M. Pappagallo et al., *The LHCb simulation application, gauss: Design, evolution and experience*, *Journal of Physics: Conference Series* **331** (2011) 032023.

[3] T.P.S. Foundation, *Sunsetting python 2*, 2020.

[4] Valassi, Andrea, Alef, Manfred, Barbet, Jean-Michel, Datskova, Olga, De Maria, Riccardo, Fontes Medeiros, Miguel et al., *Using hep experiment workflows for the benchmarking and accounting of wlcg computing resources*, *EPJ Web Conf.* **245** (2020) 07035.

[5] F. Stagni and P. Charpentier, *Jobs masonry in lhcb with elastic grid jobs*, *Journal of Physics: Conference Series* **664** (2015) 062060.

[6] P. Charpentier, *Benchmarking worker nodes using LHCb productions and comparing with HEPSpec06*, *Journal of Physics: Conference Series* **898** (2017) 082011.

[7] P.S. Foundation, *Black*, 2021.

[8] pytest-dev team, *pytest: helps you write better programs*, 2021.

[9] GitHub, *Github actions*, 2021.

[10] pre-commit dev team, *pre-commit*, 2021.

[11] P.S. Foundation, *Pypi*, 2021.

[12] Conda-Forge, *Conda-forge*, 2021.

[13] A.F. Boyer and F. Stagni, *Porting db12 to python3: Analysis of the scores*, Aug., 2022. 10.5281/zenodo.7031029.

[14] A.F. Boyer, *Porting DB12 to Python3: Comparison with LHCb Gauss tasks*, Oct., 2022. 10.5281/zenodo.7199144.