

Project Report

Improvement of the Benchmarking Tool BenRun

Tim van den Berg

MatrNr: xxxxxxxxxxxxxxxx

Email: tim.vandenberg@stud.uni-goettingen.de

Supervisor: Marcus Boden, Jonathan Decker

Georg-August-Universität Göttingen
Institute of Computer Science

Contents

1	Introduction	1
2	BenRun: A Problem Analysis	1
2.1	Architecture of BenRun	1
2.1.1	GitLab Pages	1
2.1.2	GitLab Runner	3
2.2	Problems with BenRun and Scope of the Project	4
3	Implementation	4
3.1	Installation of BenRun	4
3.2	Convert Plotting to Python	5
3.3	Integration of a new Benchmark in BenRun	6
3.3.1	Development of a small "Toy Benchmark"	6
3.3.2	Implementation of the Benchmark	7
4	Summary	7
	References	8
A	Appendix	A1
A.1	Source Code	A1

1 Introduction

For Data Center operators, it is essential to know the performance of different systems in the Data Center. In order to obtain such measurements, benchmarks are used. A benchmark uses a performance indicator to produce a representative, comparable value that describes the performance of a system in a certain performance dimension. These values can then be used to make business decisions, e.g. the values can be used to compare the system to a competitive system to decide if it is worth the investment, or compare the system to itself to detect when the performance of the system declines.

When monitoring a system like this, it would of course be nice, to do this in an automated way. Therefore, the benchmarking automation tool BenRun was developed. BenRun can run benchmarks on a system and represent the results in a visually pleasing website, that is hosted via GitLab pages. From this website, new benchmarks can be triggered with a mouse click. The results are displayed on the website which makes the usage convenient. In this project, the BenRun tool was installed, and adding a new benchmark was simplified a lot with pulling the plotting logic from the JavaScript code of the website to the preprocessing script in python. Lastly, as a proof of concept, an example "toy" benchmark was developed and added to BenRun.

This report will describe the general architecture and functionality of BenRun and its different components in chapter 2. Moreover, possible areas of improvement will be described and the scope of the project will be defined. In chapter 3 a description of the implementation is given and lastly a summary and outlook is given in the last chapter.

2 BenRun: A Problem Analysis

2.1 Architecture of BenRun

BenRun consists of two git repositories. The benchmark repository holds the benchmarks, scripts to run them, the documentation of BenRun and configuration files for the CI/CD pipeline. The repository is called "bench-dlr" for the DLR benchmarks and the forked repository is called BenRun-fork.

The benchmark repository contains a link to the results repository. Here, one can find the results of the benchmarks, a config file for the CI/CD pipeline and scripts to process the benchmark results into an html file.

From the results repository, a GitLab page is hosted, that can be used to interact with the system. How this works in detail will be explained in the next sections.

2.1.1 GitLab Pages

GitLab pages are static websites that can be published directly from a GitLab repository[Git22a]. In the case of BenRun, GitLab pages is used to provide a convenient place to see all benchmark runs (see Figure 3), visualize the results of the individual runs (see Figure 4) and submit new benchmark runs (see Figure 2) while needing relatively little work to set up (see Figure 1). In theory, even somebody who has no idea how BenRun works internally should be able to use the GitLab page, as it is essentially self-explanatory. The GitLab page is regenerated every time a new result is pushed to the results repository. This is done by running the python script mangle.py. This script builds an index.html

file, that is then provided to the GitLab pages instance. When submitting a benchmark (see Figure 2), an http request is send to a GitLab runner (see next section), that will run the respective shell script. This script can, for example, start a slurm session in which the benchmark is run.

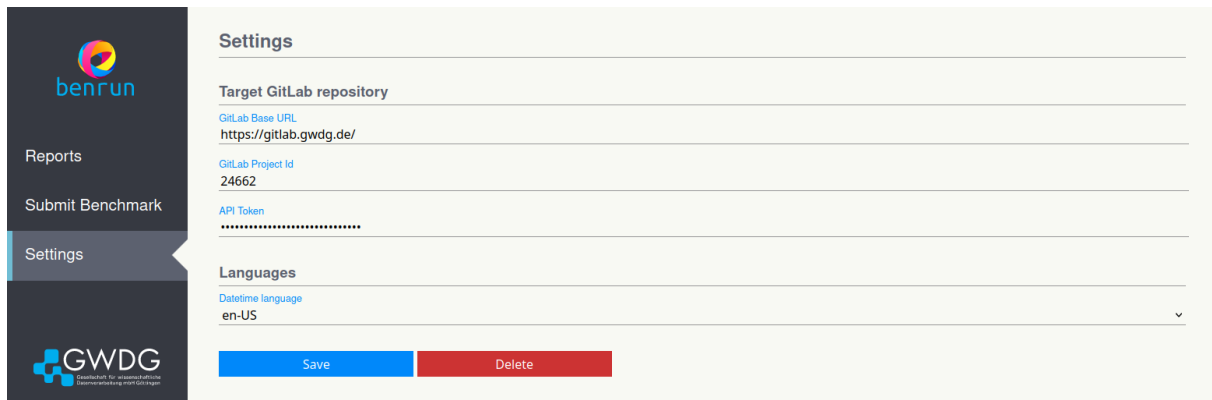
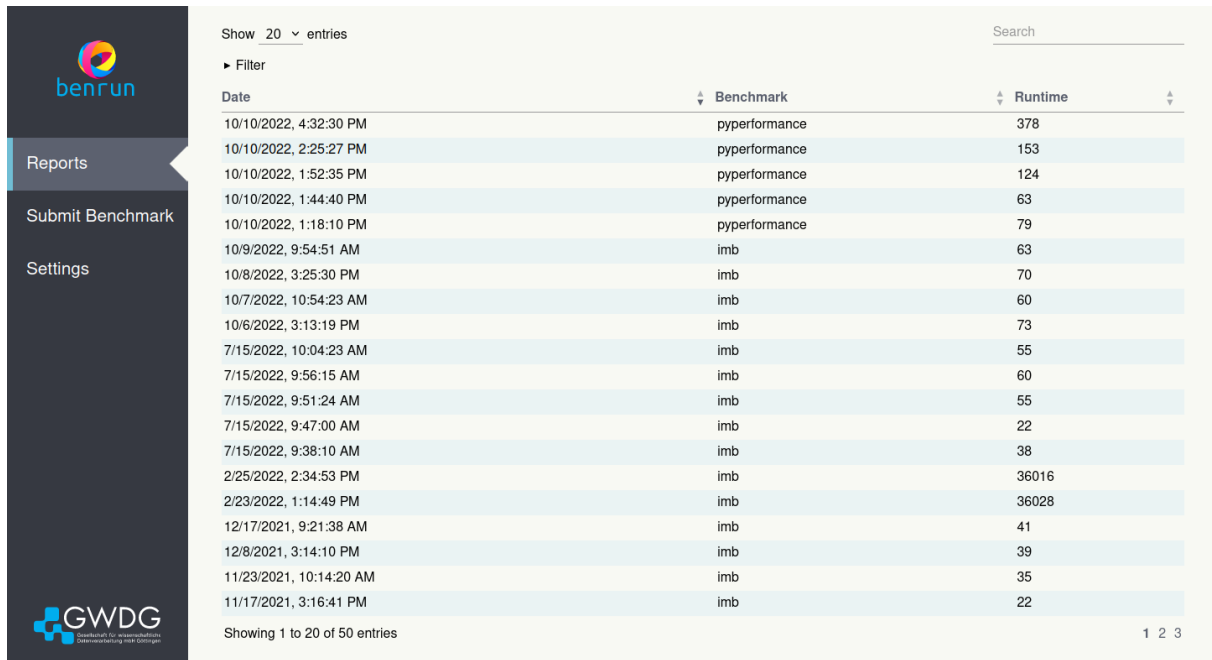


Figure 1: Settings page of the BenRun GitLab page.



Figure 2: Submission page for a new benchmark run in BenRun. The "PyPerformance" entry is a drop down menu, where the respective benchmark can be chosen.



benrun

Reports

Submit Benchmark

Settings

GWDG
Geoinformationswissenschaftliches Zentrum für Geodäsie, Luft- und Raumfahrt

Show 20 entries

Search

Filter

Date	Benchmark	Runtime
10/10/2022, 4:32:30 PM	pyperformance	378
10/10/2022, 2:25:27 PM	pyperformance	153
10/10/2022, 1:52:35 PM	pyperformance	124
10/10/2022, 1:44:40 PM	pyperformance	63
10/10/2022, 1:18:10 PM	pyperformance	79
10/9/2022, 9:54:51 AM	imb	63
10/8/2022, 3:25:30 PM	imb	70
10/7/2022, 10:54:23 AM	imb	60
10/6/2022, 3:13:19 PM	imb	73
7/15/2022, 10:04:23 AM	imb	55
7/15/2022, 9:56:15 AM	imb	60
7/15/2022, 9:51:24 AM	imb	55
7/15/2022, 9:47:00 AM	imb	22
7/15/2022, 9:38:10 AM	imb	38
2/25/2022, 2:34:53 PM	imb	36016
2/23/2022, 1:14:49 PM	imb	36028
12/17/2021, 9:21:38 AM	imb	41
12/8/2021, 3:14:10 PM	imb	39
11/23/2021, 10:14:20 AM	imb	35
11/17/2021, 3:16:41 PM	imb	22

Showing 1 to 20 of 50 entries

1 2 3

Figure 3: Table of the reports page in Benrun. One row represents a single benchmark run. One can click on a run and get to the results page (see Figure 4)

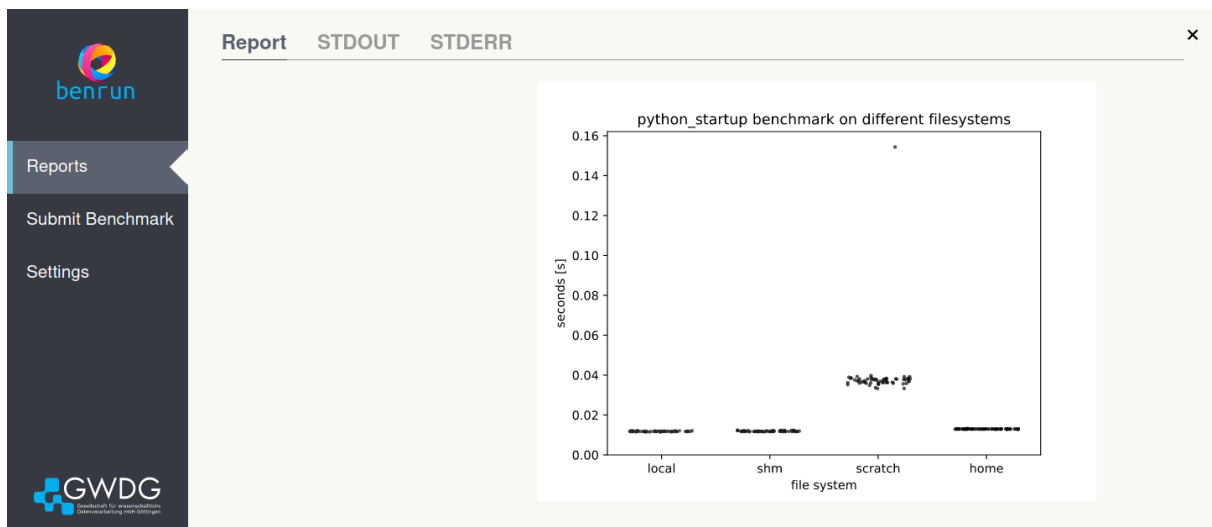


Figure 4: The results page of a PyPerformance benchmark run (see Chapter 3.3). One can click on STDOUT and STDERR to get the respective logs in plain text form.

2.1.2 GitLab Runner

A GitLab-runner is "an application that works with GitLab CI/CD to run jobs in a pipeline." [Git22b]. The GitLab Runner is an open-source GO program that has no requirements; it can run as a single binary [Git22c]. GitLab Runners support different operating systems and can run inside of Docker Containers or be deployed in a Kubernetes Cluster. In the case of BenRun, the GitLab runner has to run on a system from where it is able to start slurm jobs on the system that shall be benchmarked.

When a client requests a benchmark run via the BenRun GUI, HTTP requests are sent to

the GitLab runner which is linked to this BenRun instance. It will then start the respective slurm jobs. These slurm jobs are described in the respective sbatch slurm scripts for each different benchmark respectively in the benchmarks folder of the benchmark repository.

Generally, there is a job and a postjob script. The most important task of the postjob script is to commit and push the results of the benchmark to the result repository.

2.2 Problems with BenRun and Scope of the Project

Benruns most obvious problem is its dependence on GitLab, using the GitLab runner and GitLab Pages. This should make it a lot easier to set up in comparison to other architectures, however, if GitLab decides to make the use of GitLab runners a premium feature, this is a problem. An independent solution could be produced, however as I am not a web developer, this is not my area of expertise.

Furthermore, the documentation of BenRun is limited and the setup instructions could be refined. The tool can run in the user space and does not need administrator privileges. To test the installation process by a user, a large part of this project was dedicated to installing the tool using my user account.

Furthermore, including a new benchmark should, in the best case, be effortless. To test this, a PyPerformance benchmark was constructed and included in BenRun. PyPerformance is python library to compare the performance between different python versions against each other[22b]. The benchmark will be shown in detail in the next chapter.

The main problem, that was tackled with this project was the procedure for plotting the results. The mangle.py script extracts the data from the results directory. It then writes the data as a string directly into the JavaScript included in the index.html file. When viewing the data, JavaScript generates the plots on the fly. Therefore, the scripts are currently limited to one specific kind of line plot. Furthermore, when integrating a new benchmark into BenRun, one has to change the Python code in mangle.py and the JavaScript and CSS code in index.html. It is therefore not trivial to integrate a new benchmark, as the user has to bring a broad skill set. Different possibilities to simplify this approach were analyzed and the resulting solution will be presented in the following chapter.

3 Implementation

3.1 Installation of BenRun

In this part of the project, BenRun was successfully installed in the user space. This part of the project took far longer, than expected as a few problems appeared. The installation was necessary for all further tasks and blocked the project for some time, which was frustrating. In the following some problems and possible solutions are discussed.

One major problem was GitLabs recent decision to rename the default branch from master to main[Git21]. When forking the repository, the branch name was no longer master, but main. This led to many problems with BenRun. The solution was to fix BenRun locally. However, it would probably have saved a lot of debugging time to just rename the branch to master. Here, the problem was that the repositories were not forked properly. This was

in part due to the fact, that the original repository is hosted in the "Community Edition" instance of the GWDG GitLab instance (<https://gitlab-ce.gwdg.de/>) whereas the fork was hosted in the "Ultimate" instance <https://gitlab.gwdg.de/>. Instead of using the "fork" button in Gitlab, the repository was cloned, the `.git` directory deleted and then reinitialized in the other gitlab instance. This led to the switch from "master" to "main" branch. In hindsight, it should have been possible to use the community edition as no features of the ultimate version were needed. A lot of work could have been prevented. Furthermore, when forking the repositories, they were renamed to allow differentiation between the forked and the original repositories. This led to further problems, as the original repository name "bench-dlr" was hard coded at several places in the code. It would probably be a good idea to include the repository name in a configuration file. In some cases it is not possible to retrieve the repository path via "`git rev-parse --show-toplevel`" as the results repository is referenced from the benchmark repository. Moreover, some problems regarding Spack occurred[22a]. Spack is used to install the required software to run certain benchmarks. This installation process took several hours and failed in the end. After a few tries, it worked on a different server. Overall, the installation of BenRun only succeeded because Michael Langfermann, the main developer and initializer of BenRun, helped a lot. This was necessary, because the documentation is sparse and does not provide a lot of details. For a professional system administrator the documentation is maybe sufficient, however, it could be more specific, i.e. the documentation just states that the results repository has to be linked to the benchmark repository, it does not provide instructions on how to do this. Furthermore, there are some errors in the documentation. The installation would have been a good opportunity to repair the documentation. However, we were busy fixing bugs and simply forgot. This is something I learned for the future: good documentation is worth every minute, as it will save a lot of time in the future.

3.2 Convert Plotting to Python

The major goal of this project was to change BenRun in such a way, that python plots would be possible to use.

In the results repository, the `mangle.py` script is run to combine the data and write it as a json string into the `index.html` file. Here, JavaScript is used to generate the plots. This is not user friendly, as somebody who wants to insert a new benchmark into BenRun has to understand and adapt the `mangle.py` script and the `index.html` file. This is further complicated, as no documentation exists detailing how these files work. There is not a single comment in the code, leaving the user without help.

The basic idea of this part of the project was to change the `index.html` file in a way, that it does not generate the plots, but instead shows their already generated SVG representation. This way, users can use the programming language of their choice to plot the graphs.

As I had no experience with HTML, CSS or JavaScript, it was a challenge to understand the `index.html`. The solution I came up with was to just use the HTML `` tag, including some padding. There is surely a more elegant solution; however, due to my inexperience this solution has to suffice for the time being. The 118 lines of JavaScript code used to create a line plot were deleted. There is probably some CSS that can be deleted as well, but I did not understand all of the CSS and did not want to break anything.

The `mangle.py` script was refactored and the lineplot creation and storage as an SVG file in the newly created `assets` directory was implemented. Writing data into the `index.html` file could not be avoided, as some pages of the BenRun GUI need metadata on the benchmarks and the plots themselves are not sufficient (see Figure 3).

Overall, the new situation is much more convenient for a user, who wants to include new benchmarks in BenRun. However, one functionality was lost due to the switch to static SVG images. Before, it was possible to hover over the plots and the data points would highlight. Using the new architecture, this is no longer possible. In theory, one could have produced much more interactive plots using JavaScript. However, this comes at the penalty of a lot more work to display static simple plots. To achieve similar possibilities using Python, PyScript could be used. PyScript is a framework, that allows the execution of Python (including libraries!) in the browser[PyS22].

It was not clear to me at the time, but one main reason for the creation of BenRun was to have everything in a single HTML file. This way, the results are portable and archiveable. Nevertheless, pulling the plotting out of the html file is a good idea. Changing the `mangle.py` script to insert the SVG data into the dictionary instead of paths to the files should be an easy change.

3.3 Integration of a new Benchmark in BenRun

3.3.1 Development of a small "Toy Benchmark"

As a proof of concept, a small "toy benchmark" was developed using the PyPerformance library. The benchmark measures the startup times of python using different file systems. This works by using local conda environments on the different file systems. The benchmark does not provide a lot of information. One can see, that in contrast to the other file systems, the scratch file systems are not cached and have therefore longer startup times.

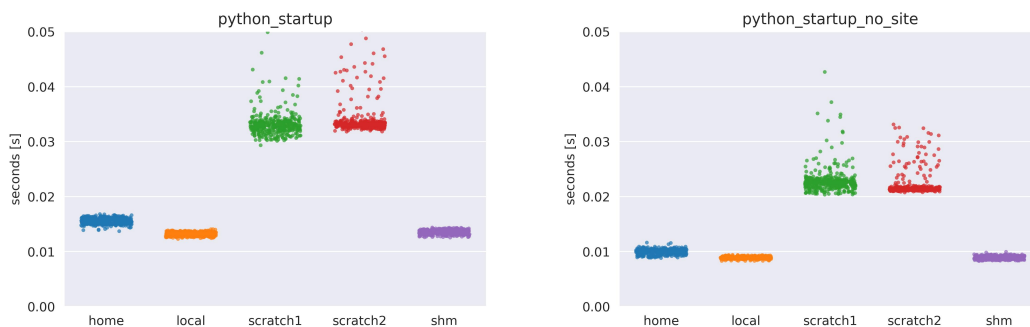


Figure 5: Startup times of python on the scientific compute cluster using different file systems. `python_startup_no_site` does not load site specific files and is therefore faster. One can see that the scratch file systems are slower and the variance of measured startup times is higher. This is due to the fact that scratch is not cached, in contrast to the other file systems.

However, one has to point out, that the startup times of the first few startups are ignored, this leads to cached startup times, that do not necessarily represent a real world application, e.g. starting python 16 times before actually using it is not typical behavior.

Nevertheless, since this is just a proof of concept, this can be ignored.

One goal of this proof of concept was to show that even exotic libraries can be used for benchmarking and plotting. Therefore, seaborn was used to plot the graphs shown in figure 5 and pyperformance was used for the benchmark.

3.3.2 Implementation of the Benchmark

A `job.sh` and `postjob.sh` sbatch file were created for the benchmark using the existing benchmarks as a blueprint.

To be able to use the respective plotting libraries, installation commands were added to the `before_script` section in the `.gitlab-ci.yml` file in the results repository.

The logic for generating and saving the plots was added to the `mangle.py` script. The resulting page can be seen in Figure 5.

4 Summary

With this project, a few goals were reached. The benchmarking automation platform BenRun was installed using a user account. Here, some problems were encountered. These problems were mainly caused by bad documentation and hard coded links within the code. This led to problems when changing the name of the repositories and forking them not correctly.

Secondly, the main part of this project was to enable the usage of Python for the plots of BenRun. Here, the plotting logic was moved from the `index.html` (JavaScript) to `mangle.py`. `Mangle.py` was refactored and has now better code quality as before. As I hold no expertise in JavaScript, HTML and CSS, only small changes were made to the `index.html` file. The current solution saves the generated SVG files in an `assets` folder. However, I learned later, that having everything in one file is a key feature of BenRun. This problem should be easily repairable by inlining the SVGs.

Lastly, as a proof of concept, a new benchmark was appended to BenRun.

In future work, one could add an automatic comparison feature to BenRun. As of now, BenRun only starts benchmarks and shows their results. However, one can only see one benchmark run at a time. It would be convenient and a selling point to combine plots of different runs, given the use case of BenRun.

As it is possible to schedule automatic benchmark runs with the GitLab CI/CD functionalities, an automated alarm system for system misbehavior could be implemented using BenRun.

Furthermore, the installation process must be simplified and the documentation updated in order for people to use BenRun.

However, BenRun could become an important tool as it is intuitive to use after the initial setup. This is proven by the usage of BenRun by the GWDG to try to minimize energy usage until November 1st to cut the energy bill. Hopefully some parts of the presented work can be useful in this endeavour, and it succeeds.

References

- [22a] *Spack Documentation*. 2022. URL: <https://spack.readthedocs.io/en/latest/>.
- [22b] *The Python Performance Benchmark suite*. 2022. URL: <https://pyperformance.readthedocs.io/>.
- [Git21] GitLab. *The new git default branch name*. Mar. 2021. URL: <https://about.gitlab.com/blog/2021/03/10/new-git-default-branch-name/>.
- [Git22a] GitLab.org. *Gitlab pages documentation*. 2022. URL: <https://docs.gitlab.com/ee/user/project/pages/>.
- [Git22b] GitLab.org. *Gitlab Runner Docs*. 2022. URL: <https://docs.gitlab.com/runner/>.
- [Git22c] GitLab.org. *Gitlab Runner Repository*. 2022. URL: <https://gitlab.com/gitlab-org/gitlab-runner>.
- [PyS22] Anaconda PyScript. 2022. URL: <https://pyscript.net/>.

A Appendix

A.1 Source Code

The original Source Code of Benrun can be found at <https://gitlab-ce.gwdg.de/hpc-team/bench-dlr> and <https://gitlab-ce.gwdg.de/hpc-team/scc/bench-dlr-results>. The changed fork can be found at <https://gitlab.gwdg.de/tim.vandenberg/benrun-fork> and <https://gitlab.gwdg.de/tim.vandenberg/benrun-fork-results>. Since these repositories are not public, access has to be granted. If you are interested, please ask.