# The Raspberry Pi: A Platform for Replicable Performance Benchmarks?

Holger Knoche
hkn@informatik.uni-kiel.de
University of Kiel

Holger Eichelberger
eichelberger@sse.uni-hildesheim.de
University of Hildesheim

## Abstract

Replicating results of performance benchmarks can be difficult. A common problem is that researchers often do not have access to identical hardware and software setups.

Modern single-board computers like the Raspberry Pi are standardized, cheap, and powerful enough to run many benchmarks, although probably not at the same performance level as desktop or server hardware. In this paper, we use the MooBench micro-benchmark to investigate to what extent Raspberry Pi is suited as a platform for replicable performance benchmarks. We report on our approach to set up and run the experiments as well as the experience that we made.

## 1 Introduction

Replicability, i.e. the ability to be repeated by other researchers with consistent results, is of fundamental importance for every scientific experiment [4]. However, replicating an experiment can be very difficult. In the case of software performance benchmarks, recreating an identical execution environment poses great challenges. In particular, obtaining identical hardware often proves to be impracticable.

In the previous work of one of the authors [6], an attempt was made to replicate benchmark results of SPASS-meter [2], a flexible resource monitoring framework. The goal of the replication was to investigate performance issues in SPASS-meter identified by the MooBench benchmark [5]. Although several opportunities for improvements were identified during the experiments, the goal of replicating the original results was not achieved despite significant efforts. Our hypothesis is that this is due to the researchers being unable to identically recreate the original execution environment, as they neither had access to identical hardware nor to the same software environment.

In this paper, we investigate to what extent the use of modern single-board computers, here the Raspberry Pi[1] platform, can improve replicability. As the term implies, these computers consist only of a single board of circuitry, containing the CPU, memory, GPU, network interface, and controllers for peripherals including USB devices. Except for the storage card, the hardware cannot be changed, and only specific models are available. This high degree of standardization, combined with its low price, makes the Raspberry Pi an interesting option for replicable performance experiments. Since a full-featured Linux distribution ("Raspbian") and important infrastructure components like a Java VM are also available, many experiments should be easily portable.

The remainder of this paper is structured as follows. In Section 2, we describe our experimental approach for investigating the suitability of the Raspberry Pi as a platform for replicable performance benchmarks. Experimental results are presented and discussed in Section 3. Section 4 concludes the paper and sketches opportunities for future work.

## 2 Experimental Setup

In order to assess the viability of the Raspberry Pi platform, we designed an experiment based on MooBench similar to [6] for several Raspberry devices. The devices, the setup of a master installation image, and the experimental procedure are described below.

We obtained two Raspberry Pi 3 devices with same specification (referred to as D1 and D2) as a set with an 8 GB SanDisk SD card supplied by the same vendor (element14) within a time frame of two weeks. The intention was to have two devices that are as identical as possible. To evaluate whether a probably different production lot affects replication, we bought a third device (D3) with same specifications a few months later from a different vendor (Allied Electronics).

We set up a single master installation image[2] suitable for the three devices including all required software. As operating system, we used Raspbian Jessie Lite, which does not contain potentially influencing services such as a graphical user interface, a virus scanner or automated updates. For preparing the master image, we installed the original image on one of the SD cards, enabled SSH for controlling the benchmarks, set up the network and updated the packages to the most recent state as of August 2017. Furthermore, we installed the required software packages, in particular, Oracle JDK 1.8.0u144 for the `armhf` plat-

[1] `https://www.raspberrypi.org/`

[2] Installation image, experimental results and analysis scripts are available at `https://doi.org/10.5281/zenodo.1003075`

form, which provides a just-in-time compiler. For the benchmarks, we installed two copies of MooBench with 512MB JVM heap, one for Kieker [1] and one for SPASS-meter (including an ARM version of its native library). Finally, we tested and archived the image so that running it on the remaining devices just required changing the static IP address.

First tests revealed storage space problems when running Kieker with MooBench. By default, MooBench executes 2,000,000 calls of a test method with a recursion depth of 10, iterates the test 10 times and collects the response time in log files. As baseline, MooBench executes the test without any instrumentation and then applies the respective monitoring framework in given configurations. However, the trace files produced by Kieker using the default MooBench setup exceeded the storage capacity of the SD cards. Therefore, we used a setup for Kieker with a recursion depth of 5 and 1,000,000 calls in 10 iterations.

Our experimental procedure for a single device includes 1) installing the master image to the respective device, 2) connecting the device via local network to a control computer, 3) starting the benchmark as a background process via SSH from the control computer for SPASS-meter and Kieker with a break of at least 5 minutes, and 4) collecting the raw MooBench results on the control computer. To simulate a replication setup, the authors executed this procedure on the respective devices in their local environment. For analyzing the overall results, we exchanged the collected data and performed an analysis (cf. Section 3). If an analysis of observed effects was needed, we re-executed individual runs with a slightly different hardware setup, e.g., using a different SD card or an external USB harddrive instead of the original SD card.

## 3 Experimental Results

For analyzing the raw results, we rely on the R analysis scripts provided with MooBench. These scripts aggregate 1,000 raw observations of all runs to one average data point and plot illustrative runtime graphs. We modified the original scripts to obtain descriptive statistics for the raw and the aggregated time series for both the complete time series and the second half where the executing JVM is expected to have reached a steady state. We start with the individual results for Kieker and SPASS-meter and then present an overarching analysis of the obtained measurements.

Kieker is a resource monitoring framework, which particularly aims at persisting obtained runtime measures as fast as possible for later offline analyses. Using MooBench, we measured Kieker in three configurations: 1) deactivated probe, 2) activated probe with no logging, and 3) activated probe with file system logging. While the first two configurations have very little overhead, writing the collected traces to the file system causes massive response time fluctuations. An obvious hypothesis is that the bandwidth to the internal SD card is limited. To examine this hypothesis, we equipped the devices with an external USB harddrive, copied MooBench to the external disk and re-executed the benchmark there. Figure 1 shows the aggregated response times on D1. Although the response time is still fluctuating for file system logging, the external hard drive reduced the mean (raw) response time by 79% and the standard deviation even by 96%. Similarly, a run with a faster SD card showed that already this change can cut the response times in half.

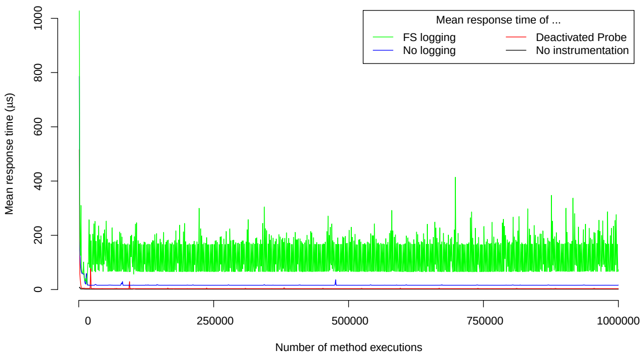

Figure 1: Response times for Kieker on D1.

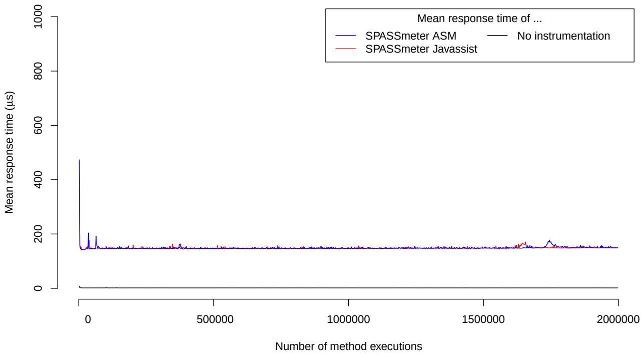

Figure 2: Response times for SPASS-meter on D2.

SPASS-meter is a resource monitoring framework which performs online aggregation of the observed measures according to a user-defined configuration. As in [6], we measured SPASS-meter with all supported resources and instrumentation framework, i.e., Javassist and ASM. Figure 2 illustrates the collected benchmark data for D1. Across the devices, the aggregated time series look rather similar, indicating a mean response time of around 160$\mu$s, a slight increase of the response time during the experiments and two humps between 1,500,000 and 2,000,000 test executions. We also checked the effect of an external USB harddrive: On D1, the mean response time improved around 8% (standard deviation by 87%), which is reasonable as SPASS-meter does not perform extensive file system operations. In the results for all devices even with external harddrive, slope and humps persisted.

We now focus on the comparison of the second half (stable state) raw time series for the three analyzed devices. Following [3], we report the time series

| Experiment | D1 | | D2 | | D3 | |
|---|---|---|---|---|---|---|
| | 95% CI | $\sigma$ | 95% CI | $\sigma$ | 95% CI | $\sigma$ |
| Baseline (SD card) | [1.6;1.6] | 0.2 | [1.6;1.6] | 0.8 | [1.6;1.6] | 0.3 |
| SPASS-meter (SD card) | [180.3; 180.4] | 45.8 | [148.8;148.9] | 45.1 | [159.0;159.0] | 39.7 |
| SPASS-meter (USB-HDD) | [164.8; 164.8] | 44.1 | [156.4;156.4] | 46.4 | [164.8;164.8] | 43.9 |
| Kieker (SD card) | [555.0;684.5] | 73,893.1 | [498.7;635.1] | 77,779.6 | [504.8;642.1] | 78,353.5 |
| Kieker (USB-HDD) | [120.8;126.4] | 3,193.7 | [109.6;114.2] | 2,612.1 | [110.8;115.7] | 2,809.4 |

Table 1: Summary of raw stable state response times as confidence intervals (CI) and standard deviation ($\sigma$).

characteristics in terms of (symmetric) confidence intervals and standard deviation. Table 1 summarizes the results for the three devices in terms of baseline (taken from the SPASS-meter experiments, similar for Kieker), SPASS-meter (Javassist instrumentation, internal SD card vs. USB harddrive) and Kieker (file system logging, internal SD card vs. USB harddrive). As the response time is measured by MooBench in terms of nanoseconds, which is typically rather imprecise on Java (some technical reports state fluctuations of about 400ns for Linux), we state the results with one significant decimal place.

The response time measures for the baseline on all three devices is very similar, i.e., we found identical 95% confidence intervals and only minor differences in standard deviation. The confidence intervals of the response times for SPASS-meter are rather narrow (maximum spread of 0.1 $\mu$s) and differ across the devices in a range of only 32 $\mu$s. The experiments with external USB harddrive lead to slightly higher response times (except for D1) and even to overlapping confidence intervals. Mean and median values of the response times differ by at most 3 $\mu$s and the variance is around one third of the mean response time. Compared with the results for standard hardware in [6], the mean response time on a Pi 3 device drops by factor 10 and the standard deviation is up to two magnitudes higher. The file system intensive benchmarks of Kieker lead to significantly higher variance and wider confidence intervals. Running the Kieker benchmarks on the devices with external USB harddrive significantly improves the response time and stabilizes the results: The response time confidence intervals on all devices are similar, tighter (spread of less than 5$\mu$s) and also have lower variance (still about 24 times the mean response time).

## 4 Conclusions and Future Work

Replication of performance experiments is a difficult task. In this paper, we analyzed whether the Raspberry Pi platform as a modern, cheap, single-board computer can be a suitable for replicable technical experiments. Setting up and distributing the installation of performance experiments among different Pi devices is rather straightforward. The obtained response time results indicate good replication capabilities, in particular regarding the CIs of the observed response times and their partial overlap. For SPASS-meter, we even identified similar characteristics in all aggregated time series (slope and humps). However, the results of experiments with intensive file system activities lead to (extremely) high variances. As a mitigation, an external harddrive can be used. Then, in addition to the Pi platform, a specification the storage media is needed for successful replication.

We conclude that the Raspberry Pi platform is a promising candidate to achieve replicable performance experiments. Possibly, the next Raspberry generation will ease such experiments by providing a faster storage interface and more memory. For explaining the identified peculiarities such as the slope or the variances, more detailed experiments are needed. In future work, we aim at investigating how entire experiments can be packaged in a replicable manner, for instance, using the *Docker* infrastructure which is also available for the Raspberry platform.

## References

[1] A. van Hoorn, J. Waller, and W. Hasselbring. “Kieker: A Framework for Application Performance Monitoring and Dynamic Software Analysis”. In: *International Conference on Performance Engineering (ICPE ’12)*. 2012.

[2] H. Eichelberger and K. Schmid. “Flexible Resource Monitoring of Java Programs”. In: *Journal of Systems and Software* 93 (2014).

[3] T. Hoefler and R. Belli. “Scientific Benchmarking of Parallel Computing Systems: Twelve ways to tell the masses when reporting performance results”. In: *International Conference on Supercomputing (SC’15)*. 2015.

[4] J. T. Leek and R. D. Pang. “Opinion: Reproducible research can still be wrong: Adopting a prevention approach”. In: *Proceedings of the National Academy of Sciences* 112.6 (2015).

[5] J. Waller, N. C. Ehmke, and W. Hasselbring. “Including Performance Benchmarks into Continuous Integration to Enable DevOps”. In: *Software Engineering Notes* 40.2 (2015).

[6] H. Eichelberger, A. Sass, and K. Schmid. “From Reproducibility Problems to Improvements: A Journey”. In: *Symposium on Software Performance (SSP ’16)*. 2016.