Beware of the Interactions of Variability Layers When Reasoning about Evolution of MongoDB

Luc Lesoil
Univ Rennes, Inria, IRISA
France

Mathieu Acher
Univ Rennes, Inria, IRISA, IUF
France

Arnaud Blouin
INSA Rennes, IRISA
France

Jean-Marc Jézéquel
Univ Rennes, IRISA
France

ABSTRACT

With commits and releases, hundreds of tests are run on varying conditions (e.g., over different hardware and workloads) that can help to understand evolution and ensure non-regression of software performance. We hypothesize that performance is not only sensitive to evolution of software, but also to different variability layers of its execution environment, spanning the hardware, the operating system, the build, or the workload processed by the software. Leveraging the MongoDB dataset, our results show that changes in hardware and workload can drastically impact performance evolution and thus should be taken into account when reasoning about evolution. An open problem resulting from this study is how to manage the variability layers in order to efficiently test the performance evolution of a software.

CCS CONCEPTS

• **Hardware** → Robustness; • **Software and its engineering** → *Empirical software validation*; Software product lines.

KEYWORDS

Deep Software Variability, Software Evolution

ACM Reference Format:
Luc Lesoil, Mathieu Acher, Arnaud Blouin, and Jean-Marc Jézéquel. 2022. Beware of the Interactions of Variability Layers When Reasoning about Evolution of MongoDB. In *Companion of the 2022 ACM/SPEC International Conference on Performance Engineering (ICPE '22 Companion), April 9–13, 2022, Bejing, China.* ACM, New York, NY, USA, 5 pages. https://doi.org/10.1145/3491204.3527489

1 INTRODUCTION

Non-regression testing is supposed to ensure that performance does not decrease drastically when adding new features, but the reality is more complex and needs to be seen from different perspectives; the impact of a commit can change according to the executing environment and the benchmark fed to the software system[1]. So, in a situation where the performance changes brought by a code modification are beneficial for half of users but decrease the quality of service for the other half, should we push this code? Modify it? Ask for more benchmarking[2]? Praise or blame? Most of the time, a commit will optimize performance properties for a vast majority of users. But software performance is sometimes sensitive to different variability layers of its execution environment, such as the hardware, the operating system or the workload processed by the software. We rely on the concept of deep variability: existing interactions between different variability layers can modify or disturb our understanding of software variability and thus changing its underlying performance [16, 25]. Since the performance decreases in a heterogeneous way across all tests, tasks and projects[3], it sometimes complexifies the developer life when reviewing code or pull requests. In this short paper, we consider MongoDB, a well-known DBMS, and ask the following questions: How do the different factors of the executing environment affect MongoDB performance distribution? Do these interactions have an impact on the performance of the software over time? Is it possible to reason about performance evolution independently of *e.g.*, workloads and hardware variability used for measuring MongoDB?

Previous work has shown that thanks to testing infrastructures like Evergreen [7], it is yet possible to identify change points [8] *i.e.*, changes made by developers that significantly alter software performance and measure their impact on the daily usage of the software. We are also interested in studying the impact of these change points, but in relation to the executing environments of MongoDB and the possible impact of variability layers. In short: is MongoDB performance evolution sensitive to deep variability?

Figure 1 typifies the problem we are addressing[4]. On the top graph, we display the change points detected by the test infrastructure over time, as well as their effect and impact on the performance (y-axis, percentage change). Bubble sizes represent the Z-scores [1] related to the change points. On the bottom, we show the evolution of performances *i.e.*, Time Series (TS) for six different executing environments but considering the same project, task, variant of hardware and test for each. Like that, we are able to observe the

ICPE '22 Companion, April 9–13, 2022, Bejing, China

ACM ISBN 978-1-4503-9159-7/22/04…$15.00
https://doi.org/10.1145/3491204.3527489

[1] See https://github.com/mongodb/mongo/pull/1068#discussion_r56706401 or https://github.com/mongodb/mongo/pull/1118#discussion_r88797998
[2] See https://github.com/mongodb/mongo/pull/472#issuecomment-22801074
[3] See also https://github.com/mongodb/mongo/pull/23
[4] Dataset: *Expanded Metrics*, Project: *sys-perf*, Task: *industry benchmark wmajority*, Hardware: *Linux 3 node replSet*, Test: *csb 50 read 50 update w majority*

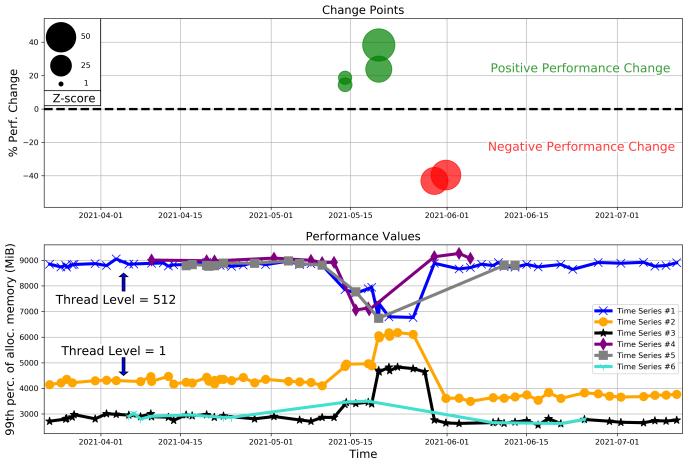


Figure 1: Joint evolution of mongoDB change points (top) and performance values (bottom) - Different thread levels change the evolution of performance values

effect of a change point on the different Time Series, each TS having its own evolution. The interesting part starts in the middle of May and ends in June 2021 ; around the 15^{th} of may, the performance property suddenly drops for TS #1, TS #4 and TS #5 while it increases for TS #2, TS #3 and TS #6. This suggests that two groups of TS react differently to the same commit. This result can be explained by looking at the thread level set when executing MongoDB. With the thread level fixed at 1, the performance property[5] increases. With a thread level of 512, the performance drops. Is it a performance bug though? Based on their thread level value when configuring the software, users will either agree or disagree. In this case, this is an example showing how an external factor (like the thread level here) threatens the work of MongoDB developers. This issue was fixed in the end of May. After this addition -or deletion, the performance matches its former value for all time series. In this paper, we aim at identifying different variability layers impacting the evolution of performance of MongoDB. We first study the difference of performance evolution according to hardware in Section 2 and then the impact of workloads in Section 3.

Technical details. This paper is part of the ICPE 2022 Data Challenge[6]. Code and results are publicly available[7].

2 IMPACT OF HARDWARE

According to the hardware platform of the final user, the performance of MongoDB may evolve differently. In this section, we quantify and study these differences of evolution, by answering the following research question: **RQ1 - What is the impact of hardware variability on the evolution of MongoDB?**

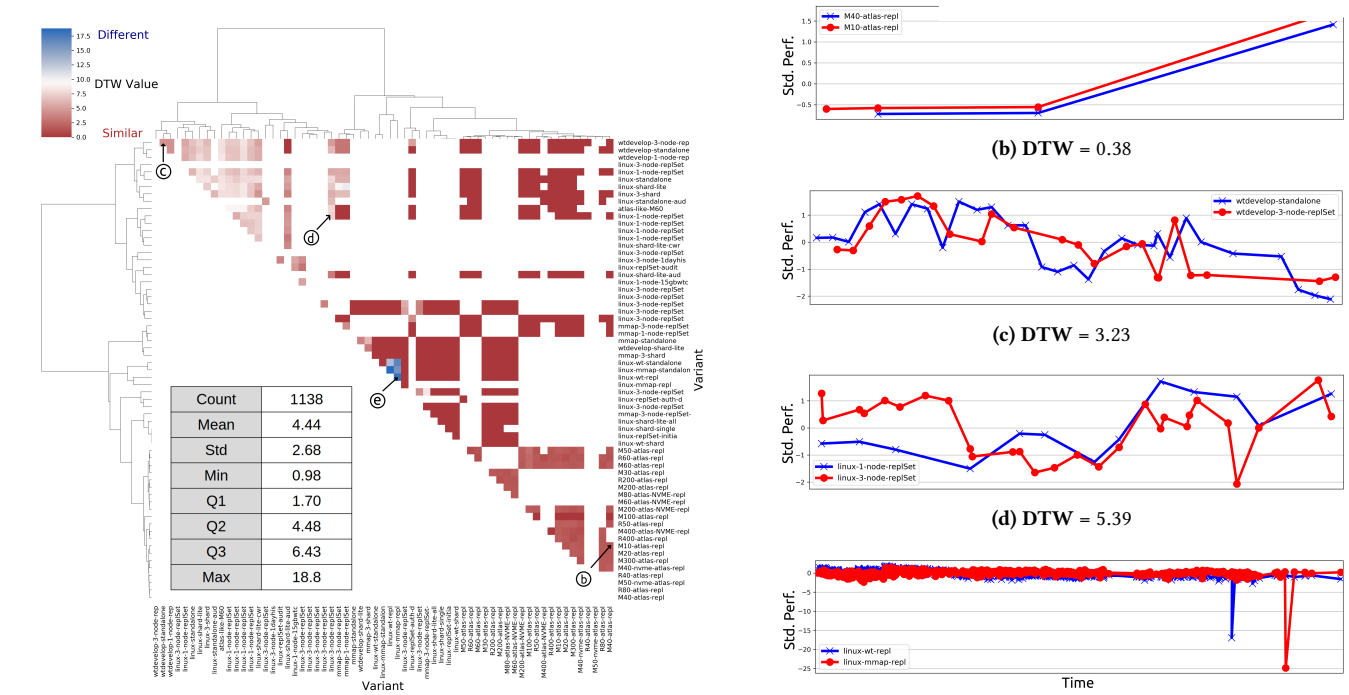
2.1 Protocol

Metric. We first need a measure to quantify the similarities between time series. We rely on Dynamic Time Warping [21] (DTW). Unlike Euclidian distance - a point by point comparison, it is able to detect a pattern common to two time series even if this pattern does not appear at the same time for both series. *Time series pre-processing.* We remove all the time series having less than two measurements. Since two time series do not necessarily have the same time stamps (*e.g.*, TS #3 and TS #6 in Figure 1), we only compare them during their common period of definition. For instance, to compare TS #3 and TS #6, we would remove the values of TS #3 before the starting time of TS #6 and after the last value of TS #6. When there exists a point in one time series that does not have a corresponding point in the other one, we interpolate the value with a linear function based on the two values closest to the missing one. To avoid biasing the results with different scales, we standardise [10] the performance values. *Implementation.* For each project, each test and each workload, we compute the DTW between the time series related to different hardware platforms. Then, we average the DTW values for each pair of variants of hardware. We consider the resulting value as a measure of similarity between the time series of two different hardware platforms. In Figure 2a, each column and each line represents a variant of hardware, on which MongoDB is executed; the intersection of a line and a column shows this resulting DTW value between the variant of the line and the variant of the column. If there is no time series with common project, test and workload between two variants, we leave it blank. Due to space issue, names of variants are cut down to 20 characters[8]. *Interpretation.* When two times series have exactly the same evolution, their DTW is equal to zero. Then, the greater the differences between time series, the greater the DTW value. We show four pairs of time series with their measure of DTW values in Figures 2b to 2e[9].

2.2 Results

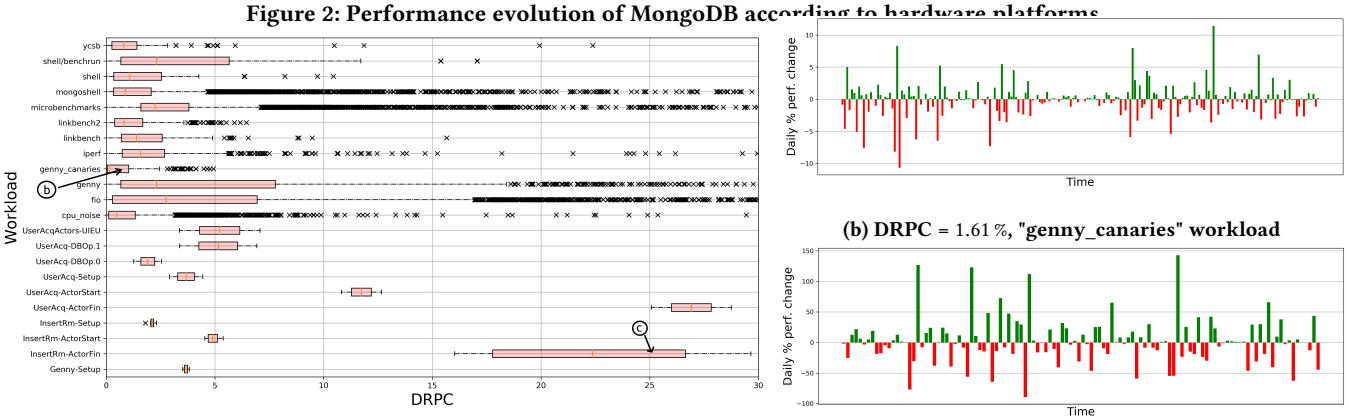
The first result is a good news w.r.t. stability: 25 % of couples of hardware have a very low average DTW value *i.e.*, inferior to 1.70 and half of them have a low DTW value *i.e.*, inferior to 4.48. This is represented by clusters of dark red cells in Figure 2a. Knowing the evolution of MongoDB for one of these hardware platforms is enough to predict or estimate the evolution for all the other hardware platforms, as shown in Figure 2b or Figure 2c. In other words, we can reduce the benchmarking cost for all these variants of hardware, because they have similar performance evolutions. As a MongoDB developer, it shows consistency between these hardware platforms, which is reassuring; optimizations made to MongoDB over time will generalize to most users' executing environments. Then, we highlight of set of hardware platforms for which the evolution of performance differs over time. In Figure 2a, they are mostly located on the top-left corner. A good example that shows how their evolution differ is depicted in Figure 2d. It seems that there is no common pattern of evolution between the different variants of hardware. As a result, the hardware layer deserves to be carefully benchmarked in such a way we understand how they react to code changes. Finally, we isolate few hardware platforms with higher DTW values. They are blue cells in Figure 2a and their differences of evolution are typified by Figure 2e. High DTW values can be explained by the presence of outliers in their performance

[5]Here, the performance property is the 99^{th} percentile of allocated memory (Mb)
[6]See https://icpe2022.spec.org/tracks-and-submissions/data-challenge-track/ and https://www.daviddaly.me/2021/10/questions-on-icpe-2022-data-challenge.html for additional explanations
[7]Companion repository : https://github.com/llesoil/icpe2022
[8]See https://github.com/llesoil/icpe2022/blob/main/results/fig2.png for Figure 2a with full names
[9]Details about Figures 2b to 2e environments properties can be consulted at https://github.com/llesoil/icpe2022/blob/main/Data%20Challenge.ipynb



(a) Heatmap of average DTW between times series related to different hardware

Figure 2: Performance evolution of MongoDB according to hardware platforms



(a) DRPC distributions per workload

Figure 3: Performance evolution of MongoDB according to different workloads.

distributions. We suspect two potential factors at the origin of these outliers: i) either we incidentally create outliers when standardising the performance or ii) this is a problem related to the performance infrastructure. Further experiments are needed to conclude.

3 IMPACT OF WORKLOAD

We now study the stability of performance evolution for different workloads fed to MongoDB. How many percents do we gain or lose between each commit? Does this value vary with the workloads? Do the workloads processed by MongoDB change its performance evolution? To address this, we answer the following research question: **RQ2 - What is the impact of workload variability on the evolution of MongoDB?**

3.1 Protocol

Metric. We define the Daily Relative Percentage Change (DRPC), a metric designed to measure the relative difference of performance (in percentage) per day :

$$DRPC(t) = \frac{100}{d(t,t+1)} * \frac{p(t+1)-p(t)}{p(t)}$$ where $p(t)$ is the performance value at the time t and $d(t, t+1)$ is the number of days between t and $t+1$. We divide by the number of days between t and $t+1$ to avoid artificially increasing the results when there are few measurements separated by long time periods. We then average the results in absolute value for each date of measurement t, which provides the average DRPC related to a time series. *Time series pre-processing.* We remove all the time series having less than two measurements,

because the metric is impossible to compute in this case. *Implementation.* We consider the 22 workloads of the dataset; for each, we gather the time series including performance measurement on this workload and compute the average DRPC. In Figure 3a, we display the boxplots of DRPC distributions per workload. We also display some examples of evolution for two workloads with the detail of percentage change values in Figures 3b and 3c[10]. *Interpretation.* The DRPC measures the stability of the evolution of MongoDB performance and quantifies the daily average percent change between two measurements. For a constant performance distribution, the DRPC is equal to zero. The greater the DRPC, the greater (and the more unstable) the evolution.

3.2 Results

Figure 3a illustrates how the variability of workloads affects the evolution of MongoDB. The daily percentage change of performance fluctuates with workloads; it can be very low or really high *e.g.*, median at 0.15 % for the "genny_canary" workload or at 26.9 % for the "UserAcq-ActorFin" workload. In Figures 3b and 3c, we plot the detail of percentage changes for these two workloads to show the difference of scale between their evolution of performance - up to a factor of 10 between their percentage changes.

Beyond the median values of the percentage changes, the results also show a difference in stability in the evolution of workload performance; while few stable workloads have a low InterQuartile Range *e.g.*, "Genny-Setup" (IQR = 0.14 %), others can have various ranges of percentage changes *e.g.*, "shell/benchurn" (IQR = 4.97 %) or "UserAcq-ActorFin" (IQR = 8.87 %). These stable workloads are reliable. For them, it is quite easy to detect an outlier, since their performance value rarely exceeds a given threshold. We guess they can be used as reference to detect a regression in the code. If such workloads observe a big decrease of performance, it is the sign that an error occurred in one of the last code modifications.

4 DISCUSSION

In this paper, we have considered two variability layers, namely hardware and workload, and studied their impacts on software evolution. Other variability layers can well be considered in the future, for example i) operating system: the Linux kernel is highly configurable and may have an impact on performance evolution of MongoDB; ii) compile-time options and flags: the way MongoDB has been built[11] can change the performance distributions. For instance, Lesoil *et al.* [18] report on preliminary evidence of complex interactions between run-time options (*e.g.*, command line parameters) and compile-time options (*e.g.*, using `./configure`) with different effects of non-functional properties of software. In the case of MongoDB we can wonder whether and to what extent these layers impact software evolution, *e.g.*, does using a different Linux variant (distribution and kernel) change the conclusions and insights about evolution? Answering these questions requires varying different layers and gather specific measurements' data. It is left as future work. Another limitation of our work is that we have not studied the interactions *between* variability layers. That is, we have focused on individual effects of each variability layer on evolution.

[10]Details about the environments used in Figures 3b and 3c can be consulted at https://github.com/llesoil/icpe2022/blob/main/Data%20Challenge.ipynb
[11]See https://github.com/mongodb/mongo/blob/master/docs/building.md

5 THREATS TO VALIDITY

Our results may vary with other choices of metrics. For instance, the DRPC used in Section 3 may amplify the noise of measurements by construction, thus slightly overestimating the real percentage of performance change. When standardising the performance in Section 2, there exists a risk to artificially create outliers if the standard deviation of the performance is too low. It could partly explain Section 2.2 results. Yet it is the only solution to efficiently compare DTW related to distributions having different scales.

6 RELATED WORK

Software evolution and performance. There exist research works studying the impact of software evolution on performance [2, 5, 8, 19, 20, 22, 24]. Martin *et al.* [19] presents a solution, based on transfer learning, to deal with the evolution of the Linux kernel when predicting kernel sizes of configurations. Mühlbauer *et al.* [22] investigated the history of software performances to isolate when a performance shift happens over time and variants (software configurations). Our study pursues a similar objective, but pays attention to the impact of hardware and workload on software evolution. Our results show that hardware and workload variability cannot be ignored when reasoning about performance history.

Hardware variability. Resarch work has been investigating the effect of hardware on software performance [4, 9, 11, 14, 27, 28]. In particular, Valov *et al.* [27, 28] suggest that changing the hardware has reasonable impacts on software configurations since linear functions are highly accurate when reusing prediction models. Unlike our work, the evolution of software is not taken into account in their studies and dataset, nor are changes in workloads.

Workload performance analysis. There are numerous works addressing the performance analysis of software systems [3, 6, 12, 13, 15, 17, 17, 23, 26] depending on different input data (also called workloads or benchmarks). In this paper, we specifically study the impact of workloads on software evolution.

Jamshidi *et al.* [14] conducted an empirical study on four configurable systems, varying software configurations and environmental conditions, namely hardware, workload, and software versions. Results show transferring performance can be difficult depending on environment changes. An important feature of the study, and therefore of their dataset, is that all three variability factors are modified together, for example, both version, hardware and workload are modified and compared to a baseline. Without isolating the individual effect of each variability layer, it is challenging to understand whether a performance shift is due to the evolution of software or other factors (hardware and workload). In contrast, our study leverages the dataset to investigate i) the impact between hardware and evolution ii) the impact between workload and evolution.

7 CONCLUSION

In this paper, we investigated to what extent two variability layers impact performance of MongoDB evolution, namely the hardware platforms on which MongoDB are executed and the workloads fed to MongoDB. Our analysis over the MongoDB dataset showed that these two variability layers cannot be ignored when reasoning about the performance evolution of MongoDB.

Acknowledgments. This research was funded by the ANR-17-CE25-0010-01 VaryVary project.

REFERENCES

[1] Hervé Abdi. 2007. Z-scores. *Encyclopedia of measurement and statistics* 3 (2007), 1055–1058.

[2] Juan Pablo Sandoval Alcocer and Alexandre Bergel. 2015. Tracking down Performance Variation against Source Code Evolution. *SIGPLAN Not.* 51, 2 (oct 2015), 129–139. https://doi.org/10.1145/2936313.2816718

[3] Juliana Alves Pereira, Mathieu Acher, Hugo Martin, and Jean-Marc Jézéquel. 2020. Sampling Effect on Performance Prediction of Configurable Systems: A Case Study. In *International Conference on Performance Engineering (ICPE 2020)*. https://hal.inria.fr/hal-02356290

[4] Christopher Brink, Erik Kamsties, Martin Peters, and Sabine Sachweh. 2014. On Hardware Variability and the Relation to Software Variability. In *2014 40th EUROMICRO Conference on Software Engineering and Advanced Applications*. 352–355. https://doi.org/10.1109/SEAA.2014.15

[5] Claudia Canali, Michele Colajanni, and Riccardo Lancellotti. 2009. Performance Evolution of Mobile Web-Based Services. *IEEE Internet Computing* 13, 2 (2009), 60–68. https://doi.org/10.1109/MIC.2009.43

[6] Emilio Coppa, Camil Demetrescu, Irene Finocchi, and Romolo Marotta. 2014. Estimating the Empirical Cost Function of Routines with Dynamic Workloads. In *Proc. of CGO'14*. 230:239. https://doi.org/10.1145/2581122.2544143

[7] David Daly. 2021. *Creating a Virtuous Cycle in Performance Testing at MongoDB*. Association for Computing Machinery, New York, NY, USA, 33–41. https://doi.org/10.1145/3427921.3450234

[8] David Daly, William Brown, Henrik Ingo, Jim O'Leary, and David Bradford. 2020. The Use of Change Point Detection to Identify Software Performance Regressions in a Continuous Integration System. In *Proceedings of the ACM/SPEC International Conference on Performance Engineering* (Edmonton AB, Canada) *(ICPE '20)*. Association for Computing Machinery, New York, NY, USA, 67–75. https://doi.org/10.1145/3358960.3375791

[9] Brian Dougherty, Jules White, Chris Thompson, and Douglas C. Schmidt. 2009. Automating Hardware and Software Evolution Analysis. In *2009 16th Annual IEEE International Conference and Workshop on the Engineering of Computer Based Systems*. 265–274. https://doi.org/10.1109/ECBS.2009.22

[10] Shirley M Dowdy and Stanley Wearden. 1983. *Statistics for research*.

[11] Omar Elkeelany and Suman Nimmagadda. 2007. Performance Evaluation of Different Hardware Models of RC5 Algorithm. In *2007 Thirty-Ninth Southeastern Symposium on System Theory*. 124–127. https://doi.org/10.1109/SSST.2007.352331

[12] Hany FathyAtlam, Gamal Attiya, and Nawal El-Fishawy. 2013. Comparative Study on CBIR based on Color Feature. *International Journal of Computer Applications* 78, 16 (Sept. 2013), 9–15. https://doi.org/10.5120/13605-1387

[13] Simon F. Goldsmith, Alex S. Aiken, and Daniel S. Wilkerson. 2007. Measuring Empirical Computational Complexity. In *Proc. of ESEC-FSE'07*. 395–404.

[14] Pooyan Jamshidi, Norbert Siegmund, Miguel Velez, Christian Kästner, Akshay Patel, and Yuvraj Agarwal. 2017. Transfer Learning for Performance Modeling of Configurable Systems: An Exploratory Analysis. In *Proc. of ASE'17*. 497–508.

[15] Philipp Leitner and Jürgen Cito. 2016. Patterns in the Chaos—A Study of Performance Variation and Predictability in Public IaaS Clouds . *ACM Trans. Internet Technol.* 16, 3, Article 15 (April 2016), 23 pages. https://doi.org/10.1145/2885497

[16] Luc Lesoil, Mathieu Acher, Arnaud Blouin, and Jean-Marc Jézéquel. 2021. Deep Software Variability: Towards Handling Cross-Layer Configuration. In *15th International Working Conference on Variability Modelling of Software-Intensive Systems* (Krems, Austria) *(VaMoS'21)*. Association for Computing Machinery, New York, NY, USA, Article 10, 8 pages. https://doi.org/10.1145/3442391.3442402

[17] Luc Lesoil, Mathieu Acher, Arnaud Blouin, and Jean-Marc Jézéquel. 2021. The Interaction between Inputs and Configurations fed to Software Systems: an Empirical Study. arXiv:2112.07279 [cs.SE]

[18] Luc Lesoil, Mathieu Acher, Xhevahire Tërnava, Arnaud Blouin, and Jean-Marc Jézéquel. 2021. The Interplay of Compile-time and Run-time Options for Performance Prediction. In *SPLC 2021 - 25th ACM International Systems and Software Product Line Conference - Volume A*. ACM, Leicester, United Kingdom, 1–12. https://doi.org/10.1145/3461001.3471149

[19] Hugo Martin, Mathieu Acher, Juliana Alves Pereira, Luc Lesoil, Jean-Marc Jézéquel, and Djamel Eddine Khelladi. 2021. Transfer Learning Across Variants and Versions: The Case of Linux Kernel Size. *IEEE Transactions on Software Engineering* (2021), 1–17. https://hal.inria.fr/hal-03358817

[20] T. Mens, M. Wermelinger, S. Ducasse, S. Demeyer, R. Hirschfeld, and M. Jazayeri. 2005. Challenges in software evolution. In *Eighth International Workshop on Principles of Software Evolution (IWPSE'05)*. 13–22. https://doi.org/10.1109/IWPSE.2005.7

[21] Meinard Müller. 2007. Dynamic time warping. *Information retrieval for music and motion* (2007), 69–84.

[22] S. Mühlbauer, S. Apel, and N. Siegmund. 2020. Identifying Software Performance Changes Across Variants and Versions. In *2020 35th IEEE/ACM International Conference on Automated Software Engineering (ASE)*. 611–622.

[23] Suporn Pongnumkul, Chaiyaphum Siripanpornchana, and Suttipong Thajchayapong. 2017. Performance Analysis of Private Blockchain Platforms in Varying Workloads. In *Proc. of ICCCN'17*. 1–7. https://doi.org/10.1109/icccn.2017.8038517

[24] Juan Pablo Sandoval Alcocer, Alexandre Bergel, Stéphane Ducasse, and Marcus Denker. 2013. Performance evolution blueprint: Understanding the impact of software evolution on performance. In *2013 First IEEE Working Conference on Software Visualization (VISSOFT)*. 1–9. https://doi.org/10.1109/VISSOFT.2013.6650523

[25] Mohammed Sayagh, Noureddine Kerzazi, and Bram Adams. 2017. On Cross-Stack Configuration Errors. In *Proceedings of the 39th International Conference on Software Engineering* (Buenos Aires, Argentina) *(ICSE '17)*. IEEE Press, 255–265. https://doi.org/10.1109/ICSE.2017.31

[26] Urjoshi Sinha, Mikaela Cashman, and Myra B. Cohen. 2020. Using a Genetic Algorithm to Optimize Configurations in a Data-Driven Application. In *Proc. of SSBSE'20*. 137–152. https://doi.org/10.1007/978-3-030-59762-7_10

[27] Pavel Valov, Jianmei Guo, and Krzysztof Czarnecki. 2020. Transferring Pareto Frontiers across Heterogeneous Hardware Environments. In *Proc. of ICPE'20*. 12–23. https://doi.org/10.1145/3358960.3379127

[28] Pavel Valov, Jean-Christophe Petkovich, Jianmei Guo, Sebastian Fischmeister, and Krzysztof Czarnecki. 2017. Transferring Performance Prediction Models Across Different Hardware Platforms. In *Proc. of ICPE'17*. 39–50.