

Transfer Learning for Performance Modeling of Configurable Systems: An Exploratory Analysis

Pooyan Jamshidi
Carnegie Mellon University, USA

Norbert Siegmund
Bauhaus-University Weimar, Germany

Miguel Velez, Christian Kästner
Akshay Patel, Yuvraj Agarwal
Carnegie Mellon University, USA

Abstract—Modern software systems provide many configuration options which significantly influence their non-functional properties. To understand and predict the effect of configuration options, several sampling and learning strategies have been proposed, albeit often with significant cost to cover the highly dimensional configuration space. Recently, transfer learning has been applied to reduce the effort of constructing performance models by transferring knowledge about performance behavior across environments. While this line of research is promising to learn more accurate models at a lower cost, it is unclear why and when transfer learning works for performance modeling. To shed light on when it is beneficial to apply transfer learning, we conducted an empirical study on four popular software systems, varying software configurations and environmental conditions, such as hardware, workload, and software versions, to identify the key knowledge pieces that can be exploited for transfer learning. Our results show that in small environmental changes (e.g., homogeneous workload change), by applying a linear transformation to the performance model, we can understand the performance behavior of the target environment, while for severe environmental changes (e.g., drastic workload change) we can transfer only knowledge that makes sampling more efficient, e.g., by reducing the dimensionality of the configuration space.

I. INTRODUCTION

Highly configurable software systems, such as mobile apps, compilers, and big data engines are increasingly exposed to end users and developers on a daily basis for varying use cases. Users are interested not only in the fastest configuration, but also in whether the fastest configuration for their applications also remains the fastest when the environmental situation has been changed. For instance, a mobile developer might be interested to know if the software that she has configured to consume minimal energy on a testing platform will also remain energy efficient on the users’ mobile platform; or, in general, whether the configuration will remain optimal when the software is used in a different environment (e.g., with a different *workload*, on different *hardware*).

Performance models have been extensively used to learn and describe the performance behavior of configurable systems [10], [11], [13], [15], [17], [19], [27], [35]–[37], [44], [47], [48]. However, the exponentially growing configuration space, complex interactions, and unknown constraints among configuration options often make it *costly* and difficult to learn an accurate and reliable performance model. Even worse, existing techniques usually consider only a single environment, since the exponential number of configurations is already a hard problem to encounter. That is, a learned performance model is accurate only for the learned environment. Recently, the use of transfer learning has been suggested to decrease the cost of learning by transferring knowledge about performance behavior across environments [7], [21], [42]. Similar to human

beings that learn from previous experience and transfer the learning to accomplish new tasks easier, here, knowledge about performance behavior gained with high effort in one environment can be reused effectively to learn models for changed environments with much lower cost. While this line of research is promising to learn accurate models at low cost, it is unclear *why* and *when* transfer learning works for performance modeling and analysis in highly configurable systems.

To understand the *why* and *when*, in this paper, we conduct an exploratory empirical study, comparing performance behavior of highly-configurable systems across environmental conditions (changing workload, hardware, and software versions), to explore what forms of knowledge can commonly be exploited for performance modeling and analysis. Specifically, we explore how performance models across the source and target of an environmental change are *related*. This relatedness across environments gives us insights to consolidate common knowledge that is shared implicitly between the two environments, from knowing entire performance distributions, to knowing about the best or invalid configurations, to knowing influential configuration options, to knowing about important interactions. This common knowledge enables *transfer learning* from the source to the target environment [32], [41].

More specifically, we explore several hypotheses about the notion of common knowledge across environments including (i) workload, (ii) hardware, and (iii) version changes. Our hypotheses start with very obvious relationships that can be easily exploited, but range toward more subtle relationships (e.g., influential options or invalid regions remain stable) that can be explored with more advanced transfer learning techniques yet to be developed. We tested our hypotheses across *36 environmental changes in 4 configurable systems* that have been selected purposefully covering different severities and varieties. For instance, we selected simple hardware changes (by changing computing capacity) to more severe changes (by changing hardware from an isolated machine to an unstable cloud machine as well as changes in workload and version).

Our results indicate that some knowledge about performance of highly-configurable systems can be transferred in even the most severe environmental changes we explored, and that transfer learning is actually easy for many environmental changes. We observed that in small environmental changes, we can frequently transfer performance models linearly across environments, while for severe environmental changes, we can still transfer partial knowledge, e.g., information about influential options or regions with invalid configurations, that can still be exploited in transfer learning, for example to avoid certain regions when exploring a configuration space. Overall, our

results are encouraging to explore transfer learning further for building performance models, showing broad possibilities of applying transfer learning beyond the relatively small changes explored in existing work (e.g., small hardware changes [42], low fidelity simulations [21], similar systems [7]).

In summary, our contributions are the following:

- We formulate a series of hypotheses to explore the presence and nature of common, transferable knowledge between a source and a target environment, ranging from easily exploitable relationships to more subtle ones.
- We empirically investigate performance models of 4 configurable systems before and after 36 environmental changes. We performed a thorough exploratory analysis to understand why and when transfer learning works.
- We discuss general implications of our results for performance modeling of configurable software systems.

II. INTUITION

Understanding the performance behavior of configurable software systems can enable (i) performance debugging [12], [36], (ii) performance tuning [14], [16], [17], [26], [27], [30], [39], [42], [44], (iii) design-time evolution [2], [20], or (iv) runtime adaptation [9]–[11], [15], [21], [22]. A common strategy to build performance models is to use some form of sensitivity analysis [34] in which the system is executed repeatedly in different configurations and machine learning techniques are used to generalize a model that explains the influence of individual options or interactions [13], [36], [42].

In this paper, we are interested in how a performance model for a configurable system changes when we deploy the system in a different environment. To this end, we distinguish between *configuration options* – parameters that users can tweak inside the system to select functionality or make tradeoffs among performance, quality, and other attributes – and *environment changes* – differences in how the system is deployed and used in terms of workload, hardware, and version. If a performance model remains relatively stable across environments (e.g., the top configurations remain the top configurations, the most influential options and interactions remain most influential), we can exploit this stability when learning performance models for new environments. Instead of learning the model from scratch, we can reuse knowledge gathered previously for other environments in a form of *transfer learning* [7], [32], [41]. That is, we can develop cheaper, faster and more accurate learning techniques for performance models that allow us to make predictions and optimizations of performance for highly configurable systems *in changing environments* [21].

For example, consider an update to faster hardware. We would often expect that the system will get faster, but will do so in a nearly uniform fashion. However, we may expect that options that cause a lot of I/O operations (e.g., a backup feature) may benefit less from a faster CPU than other options; so not all environment changes will cause uniform changes. If transfer across hardware is indeed usually easy, this encourages, for example, scenarios in which we learn performance models offline on cheap hardware and transfer it to the real system with few expensive measurements for adjustment.

The question we ask in this paper is what kind of knowledge can frequently be exploited across environment changes in practice, with simple or more advanced forms of transfer

learning. Specifically, we ask whether there exists common information (i.e., *transferable/reusable knowledge*) that applies to both source and target environments of configurable systems and therefore can be carried over across environments.

A. Environmental changes

Let us first introduce what do we mean by *environment*, the key concept that is used throughout this paper. An environmental condition for a configurable system, in this work, is determined by its hardware, workload, and software version. (i) *Hardware*: The deployment configuration in which the software system is running. The performance behavior of the system under study can differ when it is deployed on a different hardware with different resource constraints. (ii) *Workload*: The input of the system on which it operates on. The performance behavior of the system can vary under different workload conditions. (iii) *Version*: The state of the code base at a certain point in time. When part of the system undergoes some updates (e.g., when a library that is used in the system boosts its performance in a recent version update), the overall performance of the system will change. Of course, other environmental changes might be possible as well (e.g., changes to the programming language by which a system is implemented). But, we limit this study to this selection as we consider the most important and common environmental changes in practice.

B. Preliminary concepts

In this section, we provide formal definitions of four concepts that we use throughout this study. The formal notations enable us to concisely convey concepts throughout the paper.

1) *Configuration and environment space*: Let C_i indicate the i -th configuration option of a system \mathcal{A} , which is either enabled or disabled (the definitions easily generalize to non-boolean options with finite domains). The configuration space is mathematically a Cartesian product of all configuration options $\mathcal{C} = \text{Dom}(C_1) \times \dots \times \text{Dom}(C_d)$, where $\text{Dom}(C_i) = \{0, 1\}$ and d is the number of options. A *configuration* of a system is then a member of the configuration space where all the options are assigned to a specific value in their domain.

We describe an environmental condition e by 3 variables $e = [h, w, v]$ drawn from a given environment space $\mathcal{E} = H \times W \times V$, where each member represents a set of possible values for the hardware h , workload w , and system version v . We use notation $ec : [h, w_1 \rightarrow w_2, v]$ as shorthand for an environment change from workload w_1 to workload w_2 where hardware and version remain stable.

2) *Performance model*: Given a software system \mathcal{A} with configuration space \mathcal{C} and environment space \mathcal{E} , a *performance model* is a black-box function $f : \mathcal{C} \times \mathcal{E} \rightarrow \mathbb{R}$ that maps each configuration $c \in \mathcal{C}$ in an environment $e \in \mathcal{E}$ to a real number corresponding to the performance of the system. To construct a performance model, we run \mathcal{A} in a fixed environmental condition $e \in \mathcal{E}$ on various configurations $c_i \in \mathcal{C}$, and record the resulting performance values $y_i = f(c_i, e) + \epsilon_i$ where $\epsilon_i \sim \mathcal{N}(0, \sigma_i)$ is the measurement noise corresponding to a normal distribution with zero mean and variance σ_i^2 . The training data for learning a performance model for system \mathcal{A} in environment e is then $\mathcal{D}_{tr} = \{(c_i, y_i)\}_{i=1}^n$, where n is the number of measurements.

3) *Performance distribution*: Let us now define another concept that helps us to investigate the relatedness across environments. We can and will compare the performance models, but a more relaxed representation that allows us to assess the potentials for transfer learning is the empirical *performance distribution*. The performance distribution is a stochastic process, $pd : \mathcal{E} \rightarrow \Delta(\mathbb{R})$, that defines a probability distribution over performance measures for each environmental conditions. To construct a performance distribution for a system \mathcal{A} with configuration space \mathcal{C} , similarly to the process of deriving the performance models, we run \mathcal{A} on various configurations $e_i \in \mathcal{C}$, for a specific environmental condition $e \in \mathcal{E}$ and record the resulting performance values y_i . We then fit a probability distribution to the set of measured performance values $\mathcal{D}_e = \{y_i\}$ using kernel density estimation [4] (in the same way as histograms are constructed in statistics).

4) *Influential option*: At the level of individual configuration options, we will be interested in exploring whether options have an influence on the performance of the system in either environment; not all options will have an impact on performance in all environments. We introduce the notion of a *influential option* to describe a configuration option that has a statistically significant influence on performance. That is, when comparing the pairs of configuration in which this option is enabled and disabled respectively, an influential option has a consistent effect to speed up or slow down the program, beyond random chance. Specifically, we test whether the influence of a options is statistically significant.

5) *Options interaction*: The performance influence of individual configuration options may not compose linearly. For example, while encryption will slow down the system due to extra computations and compression can speed up transfer over a network, combining both options may lead to surprising effects because encrypted data is less compressible. In this work, we will look for *interactions of configuration options* as nonlinear effects where the influence of two options combined is different from the sum of their individual influences [36], [37], where this difference is statistically significant across all configurations that differ only in these two options.

6) *Invalid configuration*: We consider a configuration as invalid if it causes a failure in the system or produce an unacceptable performance.

C. Transferable knowledge

As depicted in Figure 1, any sort of knowledge that can be *extracted* from the source environment and can contribute to the learning of a better model (i.e., faster, cheaper, more accurate, or more reliable) in the target environment is considered as a “transferable knowledge” (or reusable knowledge [1]). There are several things we can transfer: (i) classification or regression models, (ii) dependency graphs that represent the dependencies among configurations, and (iii) option interactions in order to prioritize certain regions in the configuration space. For transferring the extracted knowledge, we need a *transfer function* that transforms the source model to the target model: $tf : f(\cdot, e_s) \rightarrow f(\cdot, e_t)$. In its simplest form, it can be a linear mapping that transforms the source model to the target: $f(\cdot, e_t) = \alpha \times f(\cdot, e_s) + \beta$, where α, β need to be learned using few observations from both environments.

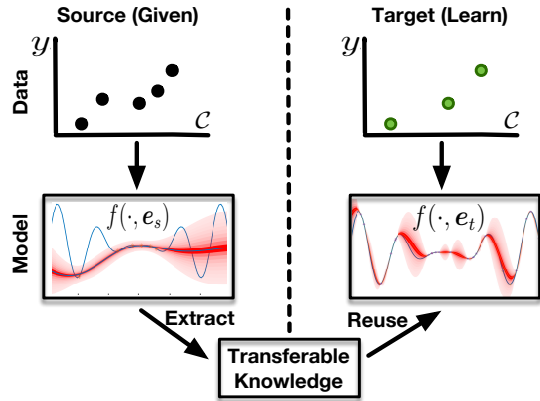


Fig. 1: Transfer learning from source to target environment.

III. RESEARCH QUESTIONS AND METHODOLOGY

A. Research questions

The overall question that we answer in this paper is “*why and when does transfer learning work for configurable software systems?*” Our hypothesis is that performance models in source and target environments are usually somehow “related.” To understand the notion of relatedness that we commonly find for environmental changes in practice, we explore several research questions (each later with several hypotheses), starting with explorations of very strong notions of relatedness (e.g., linear relationships) toward weaker ones (e.g., stability of influential options):

RQ1: *Does the performance behavior stay consistent across environments? (Section IV)*

If we can establish with RQ1 that linear changes across environments are common, this would be promising for transfer learning because even simple linear transformations can be applied. Even if not all environment changes may be amendable to this *easy transfer learning*, we explore what kind of environment changes are more amendable than others.

RQ2: *Is the influence of configuration options on performance consistent across environments? (Section V)*

For cases in which easy changes explored in RQ1 are not possible, RQ2 explores information that can be exploited for transfer learning at the level of individual configuration options. Specifically, we explore how commonly the influential options remain stable across environment changes.

RQ3: *Are the interactions among configuration options preserved across environments? (Section VI)*

In addition to individual options in RQ2, RQ3 explores interactions among options, that, as described above, can often be important for explaining the effect of performance variations across configurations. Again, we explore how commonly interactions are related across environment changes.

RQ4: *Are the configurations that are invalid in the source environment with respect to non-functional constraints also invalid in the target environment? (Section VII)*

Finally, RQ4 explores an important facet of invalid configurations: How commonly can we transfer knowledge about invalid configurations across environments? Even if we cannot transfer much structure for the performance model otherwise, transferring knowledge about invalid parts of the configuration

space can focus learning in the target environment on the relevant parts of the configuration space.

B. Methodology

Design: We investigate changes of performance models across environments. Therefore, we need to establish the performance of a system and how it is affected by configuration options in multiple environments. To this end, we measure the performance of each system using standard benchmarks and repeated the measurements across a large number of configurations. We then repeat this process for several changes to the environment: using different hardware, using different workloads, and using different versions of the system. Finally, we perform the analysis of relatedness by comparing the performance and how it is affected by configuration options in one environment against the performance measured in other environments. We perform a total of 36 comparisons of environment changes.

Analysis: For answering the research questions, we formulate different assumptions about the relatedness of the source and target environments as hypotheses – from more stronger to more relaxed assumptions. For each hypothesis, we define one or more metrics and analyze 36 environment changes in four subject systems described below. For each hypothesis, we discuss how commonly we identify this kind of relatedness and whether we can identify classes of environment changes for which this relatedness is characteristic. If we find out that for an environmental change a stronger assumption holds, it means that a more informative knowledge is available to transfer.

Severity of environment changes: We purposefully select environment changes for each subject system with the goal of exploring many different kinds of changes with different expected *severity* of change. With a diverse set of changes, we hope to detect patterns of environment changes that have similar characteristics with regard to relatedness of performance models. We expect that less severe changes lead to more related performance models that are easier to exploit in transfer learning than more severe ones. For transparency, we recorded the *expected* severity of the change when selecting environments, as listed in Table II, on a scale from small change to very large change. For example, we expect a small variation where we change the processor of the hardware to a slightly faster version, but expect a large change when we replace a local desktop computer by an unstable cloud virtual machine. Since we are neither domain experts nor developers of our subject systems, recording the expected severity allows us to estimate how well intuitive judgments can eventually be made about suitability for transfer learning and it allows us to focus our discussion on surprising observations.

C. Subject systems

In this study, we selected four different configurable software systems from different domains, with different functionalities, and written in different programming languages (cf. Table I). All systems are configurable with at least 10 configuration options.

SPEAR is an industrial strength bit-vector arithmetic decision procedure and a Boolean satisfiability (SAT) solver. It is designed for proving software verification conditions and

it is used for bug hunting. We considered a configuration space with 14 options that represent heuristics for solving the problems and therefore affect the solving time. We measured how long it takes to solve a SAT problem in all 16,384 configurations in multiple environments: four different SAT problems with different difficulty serve as workload, measured on three hardware system, with two versions of the solver as listed in Table II. The difficulty of the workload is characterized by the SAT problem’s number of variables and clauses.

x264 is a video encoder that compresses video files with a configuration space of 16 options to adjust output quality, encoder types, and encoding heuristics. Due to the size of the configuration space, we measured a subset of 4000 sampled randomly configurations. We measured the time needed to encode three different benchmark videos on two different hardware systems and for three versions as listed in Table II. Each benchmark consists of a raw video with different quality and size and we expect that options related to optimizing encoding affect the encoding time differently. We judged expected severity of environmental changes based on the difference between quality and size of benchmark videos.

SQLite is a lightweight relational database management system, embedded in several browsers and operating systems, with 14 configuration options that change indexing and features for size compression useful in embedded systems, but have performance impact. We expect that some options affect certain kinds of workload (e.g., read-heavy rather than write-heavy workloads) more than others. We have measured 1000 randomly selected configurations on two hardware platforms for two versions of the database system; as workload, we have considered four variations of queries that focus on sequential reads, random reads, sequential write, and batch writes.

SaC is a compiler for high-performance computing based on stateless arrays [33]. The SaC compiler implements a large number of high-level and low-level optimizations to tune programs for efficient parallel executions configurable with 50 options controlling optional optimizer passes such as function inlining, constant folding, and array elimination. We measure the execution time of a program (i.e., the workload in this context) compiled with different 71,267 randomly selected configurations to assess the performance impact of SaC’s options. As workloads, we select 10 different demo programs shipped with SaC, each computationally intensive, but with different characteristics. Workloads include Monte Carlo algorithms such as pfilter with multiple optimizable loops as well as programs heavily based on matrix operations like *srad*. We did not explore hardware or version changes for this system because measuring this large number of configurations required already several weeks despite parallelization over 10 identical machines, and the cost of additional variations would be prohibitive.

To account for measurement noise, we have measured each configuration of each system and environment 3 times and used the mean for the analyses. While many performance and quality measures can be analyzed, our primary performance metric is wall-clock execution time, which is captured differently for each systems in Table I: execution time, encoding time, query time, analysis time.

TABLE I: Overview of the real-world subject systems.

System	Domain	d	$ C $	$ H $	$ W $	$ V $
SPEAR	SAT solver	14	16384	3	4	2
x264	Video encoder	16	4000	2	3	3
SQLite	Database	14	1000	2	14	2
SaC	Compiler	50	71267	1	10	1

d : configuration options; C : configurations; H : hardware environments; W : analyzed workload; V : analyzed versions.

IV. PERFORMANCE BEHAVIOR CONSISTENCY (RQ1)

Here, we investigate the relatedness of the source and target environments in the entire configuration space. We start by testing the strongest assumptions about the relatedness of source and target environments, which would enable an easy transfer learning (H1.1). We expect that the first hypothesis holds only for simple environmental changes. Therefore, we subsequently relax the hypothesis to test whether and when the performance distributions of source and target environments become similar (H1.2), and whether the ranking of configurations (H1.3) and the ranking of the top/bottom configurations (H1.4) stay consistent across environments.

H1.1: The relation of the source response to the target is a constant or proportional shift.

Importance. If the target response is related to the source by a constant or proportional shift, it is trivial to understand the performance behavior for the target environment using the model that has already been learned in the source environment: We need to *linearly transform* the source model to get the target model. We expect a linear shift if a central hardware device affecting the functionality of all configuration options homogeneously, changes such as the CPU, or homogeneous workload change. Previous studies demonstrated the existence of such cases where they trained a linear transformation to derive the target model, but only with hardware changes [42]. **Metric.** Here, we investigate whether $f(c, e_t) = \alpha \times f(c, e_s) + \beta, \forall c \in C$. We use metric **M1**: Pearson linear correlation [4] between $f(c, e_s)$ and $f(c, e_t)$ to evaluate this hypothesis. If the correlation is 1, we can linearly transform performance across environments. However, due to measurement noise, we do not expect perfect correlation. But, we consider correlations higher than 0.9 as strong enough that simple transfer techniques can produce good approximations for estimating the target performance model.

Results. The result in Table II show very high correlations for about a third of all studied environmental changes. In particular, we observe high correlations for hardware changes and for many workload changes of low expected severity.

Hardware change: Hardware changes often result in near-perfect correlations except for severe changes where we have used unstable hardware (e.g., Amazon cloud in ec_2). We investigated why using unstable cloud hardware resulted in weak linear correlations. We analyzed the variance of the measurement noise and we observed that the proportion of the variance of the noise in the source to the target in ec_2 is $\bar{\sigma}_{ec_2^s}^2 / \bar{\sigma}_{ec_2^t}^2 = 33.39$, which is an order of magnitude larger than the corresponding one in ec_1 that is $\bar{\sigma}_{ec_1^s}^2 / \bar{\sigma}_{ec_1^t}^2 = 1.51$. This suggests that we can expect a linear transformation across environments when hardware resources execute in a stable environment. For transfer learning, this means that we can

reuse measurements from cheaper or testing servers in order to predict the performance behavior [6]. Moreover, it also means that virtualization hinders transfer learning such that we might not be able to distribute measurements to cloud environments.

Workload change: For SPEAR, we observed very strong correlations across environments where we have considered SAT problems of different sizes and difficulties. Also, when the difference among the problem size and difficulty is closer across environments (e.g., ec_3 vs. ec_4) the correlation is slightly higher. This observation has also been confirmed for other systems as well. For instance, in environmental instance ec_3 in SQLite, where the workload change is write-heavy from sequential to batch, we have observed an almost perfect correlation, 0.96, while in the read-heavy workload ec_4 (random to sequential read) the correlation is only medium at 0.5. There are two reasons for this medium correlation: First, the underlying hardware contains an SSD, which has different performance properties for reading and writing. Second, a database performs different internal functions when inserting or retrieving data, which affects performance. This implies that some environmental conditions may provide a better means for transfer learning. In environmental change $ec_{12,13}$ in SaC, we observe perfect correlations. In these two workload variations, we measured the runtime of one application (nbody) with two different input values and it is not surprising to see a perfect correlation due to the same application logic. This suggests that when a workload change is not severe, we can expect there exists a linear transformation between source to the target.

Version change: For SPEAR ($ec_{5,6,7}$) and x264 ($ec_{5,6,7,8}$), the correlations are extremely weak or non existence, while for SQLite (ec_5), it is almost perfect. This can be attributed to several reasons. One reason is that the optimization features that are determined by the configuration options for SPEAR and x264 may undergo a substantial revision from version to version because these are lab applications and so algorithmic advancements may significantly improve the way how the optimization features work. Later, when we investigate the interactions between configuration options in RQ3, we will learn that the code updates for SPEAR cause several new interactions between options. The implication for transfer learning is that code changes that substantially influence the internal logic controlled by configuration options may require a non-linear form of transformation or a complete set of new measurements in the target environment for those options only.

Insight. For non-severe hardware changes, we can linearly transfer performance models across environments.

H1.2: The performance distribution of the source is similar to the performance distribution of the target environment.

Importance. In the previous hypothesis, we investigated the situation whether the response functions in the source and target are linearly correlated. In this hypothesis, we consider a relaxed version of H1.1 by investigating if the performance distributions are similar. When the performance distributions are similar, it does not imply that there exists a linear mapping between the two responses, but, there might be a more sophisticated relationship between the two environments that can be captured by a *non-linear transfer function*.

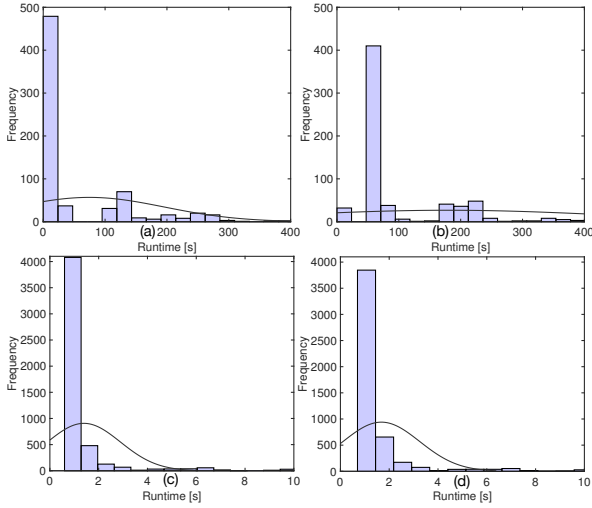


Fig. 2: Performance distributions of ec_1 (a,b) and ec_{13} (c,d) in SaC. (a,c) and (b,d) are the distributions of the source and target.

Metric. We measure **M2:** Kullback-Leibler (KL) divergence [8] to compare the similarity between the performance distributions: $D_{KL}^{ec}(pd_s, pd_t) = \sum_i pd_s(c_i) \log \frac{pd_s(c_i)}{pd_t(c_i)}$, where $pd_{s,t}(\cdot)$ are performance distributions and $c_i \in \mathcal{C}$. As an example, we show the performance distributions of ec_1 and ec_{13} and compare them using KL divergence in Figure 2: The less the value of KL divergence is, the more similar are the distributions. We consider two distributions are similar if $D_{KL}^{ec}(pd_s, pd_t) < 3$ (based on recommendations in the machine learning community [4]) and dissimilar otherwise.

Results. Here, we are interested to find environmental changes where we did not observe a strong correlation between responses, but there might be similarities between the performance distributions of the environment. For the environmental changes $ec_{5,6}$ in SPEAR, ec_{3-7} in x264, $ec_{4,6}$ in SQLite, and $ec_{5,8}$ in SaC, the performance distributions are very similar across source and target. This implies that there exist a possibly non-linear transfer function that we can map performance models across environments. Previous studies demonstrated the feasibility of highly non-linear kernel functions for transfer learning in configurable systems [21].

Insight. Even for some severe environmental changes where there is no linear correlation across performance models, the performance distributions are similar, showing the potential for learning a non-linear transfer function.

H1.3: The ranking of configurations stays stable.

Importance. If the ranking of the configurations stays similar, the response function is then stable across environments. We can use this knowledge to prioritize certain regions in the configuration space for optimizations.

Metric. Here, we use *rank correlation* by measuring the **M3:** Spearman correlation coefficient. Intuitively, the Spearman correlation between two responses will be high when observations have a similar rank. We consider rank correlations higher than 0.9 as strong and suitable for transfer learning.

Results. The results in Table II show that the rank correlations are high across all hardware changes and small workload changes. This metric does not provide additional insights from what we have observed in H1.1. However, in one environmental change, where due to an excessive measurement noise, the linear correlation were low, ec_2 for SPEAR, the rank correlation is high. This might hint that when unstable hardware conditions exist, the overall ranking stay the same.

Insight. The configurations retain their relative performance profile across hardware platforms.

H1.4: The top/bottom performer configurations are similar.

Importance. If the top configurations are similar across environments, we can extract their characteristics and use that in the transfer learning process. For instance, we can identify the top configurations from the source and inform an optimizer [19] to exploit this knowledge in the target while exploring with a lower priority other regions. The bottom configurations can be used to avoid corresponding regions during sampling. Note that this is a more relaxed hypothesis than H1.3 by concentrating only on a subset of configurations. **Metric.** We measure **M4/M5:** the percentage of top/bottom configurations in the source that are also top/bottom performers in the target. Note that we present the results only for the 10th percentile, while other percentile values are calculated and resulted in similar observations.

Results. The results in Table II show that top/bottom configuration are common across hardware and small workload changes, therefore, this metric does not provide additional insights from what we have observed in H1.1.

Insight. Only hardware changes preserve top configurations across environments.

V. SIMILARITY OF INFLUENTIAL OPTIONS (RQ2)

Here, we investigate whether the influence of individual configuration options on performance stays consistent across environments. We investigate two hypotheses about the influence strength (H2.1) and the importance of options (H2.2).

H2.1: The influential options on performance stay consistent.

Importance. In highly dimensional spaces, not all configuration options affect the response significantly. If we observe a high percentage of common influential options across environments, then we need to sample across only a *subset* of all configuration options. because we already know that these are the key options influencing performance.

Metric. In order to investigate the option-specific effects, we use paired t-test analysis [4] to test if an option leads to any significant performance change and whether this change is similar across environments. We measure **M6/M7:** the number of influential options in source and target; We also measure **M8/M9:** the number of options that agree/disagree.

Results. The results in Table II show that slightly more than half of the options, for all subject systems, are influential either in the source or target environments. From the influential options, a very high percentage are common in both. This can lead to a substantial reduction for performance measurements:

we can fix the non-influential options and sample only along options, which we found influential from the source.

Insight. Only a subset of options are influential and a large proportion of influential options are preserved.

H2.2: The importance of options stays consistent.

Importance. In machine learning, each decision variable (here option) has a relative importance to predict the response and importance of the variables play a key role for in the feature selection process [4]. Here, we use this concept to determine the relative importance of configuration options, because in configurable systems, we face many options that if prioritized properly, it can be exploited for performance predictions [21].

Metric. We use regression trees [4] for determining the relative importance of configuration options because (i) they have been used widely for performance prediction of configurable systems [13], [42] and (ii) the tree structure can provide insights into the most essential options for prediction, because a tree splits on those options first that provide the highest information gain [13]. We derive estimates of the *importance of options* for the trained trees on the source and target by examining how the prediction error will change as a result of options. We measure **M10**: correlation between importance of options for comparing the consistency across environments.

Results. From Table II, the correlation coefficient between the importance of options for different environmental changes is high, and the less severe a change the higher the correlation coefficients. This confirms our intuition that small changes in the environment do not affect the influence strength of an option. Let us now consider some environmental changes where the correlation between response functions were low across environments according to M1 and are high according to M10: $ec_{6,7}$ in SPEAR, ec_{3-7} in x264, $ec_{1,2,5,7-11,14}$ in SaC. This observation gives further evidence that even though we did not observe a linear correlation, there might exist a non-linear relationship. For instance, the influence of options on performance stay the same, but interactions might change and are not accounted by a linear model transformation.

Insight. The strength of the influence of configuration options is typically preserved across environments.

VI. PRESERVATION OF OPTION INTERACTIONS (RQ3)

We state two hypotheses about the preservation of option interactions (H3.1) and their importance (H3.2).

H3.1: The interactions between configuration options are preserved across environments.

Importance. In highly dimensional configuration spaces, the possible number of interactions among options is exponential in the number of options and it is computationally infeasible to get measurements aiming at learning an exhaustive number of interactions. Prior work has shown that a very large portion of interactions is non-influential [25], [37].

Metric. The main objective here is to evaluate to what extent the interactions will be preserved from source to target. Here, we learn step-wise linear regression models; a technique that

has been used for creating performance influence model for configurable systems [36]. We learn all pairwise interactions, independently in the source and target environments. We then calculate the percentage of *common pairwise interactions* from the model by comparing the coefficients of the pairwise interaction terms of the regression models. We concentrated on pairwise interactions, as they are the most common form of interactions [25], [37]. We measure: **M11**: The number of interactions in the source; **M12**: The number of interactions in the target; **M13**: The number of interactions that agree on the direction of effects in the source and the target.

Results. The results in Table II show three important observations: (i) only a small proportion of possible interactions have an effect on performance and so are relevant (confirming prior work); (ii) for the large environmental changes, the difference in the proportion of relevant interactions across environments is more different, while for smaller environmental changes, the proportion is almost equal; (iii) a very large proportion of interactions is common across environments.

The mean percentage of interactions (averaged over all changes) are 25%, 28%, 10%, 6% for SPEAR, x264, SQLite, SaC respectively, where 100% would mean that all pairwise combination of options have a distinct effect on performance. Also, the percentage of common interactions across environments is high, 96%, 81%, 85%, 72% for SPEAR, x264, SQLite, SaC respectively. This result points to an important transferable knowledge: interactions often stay consistent across changes. This insight can substantially reduce measurement efforts to purposefully measure specific configurations.

Insight. A low percentage of potential interactions are relevant for performance.

H3.2: The effects of interacting options stay similar.

Importance. If the effects of interacting options are similar across environments, we can prioritize regions in the configuration space based on the importance of the interactions.

Metric. We measure **M14**: the correlation between the coefficients of the pairwise interaction terms in the linear model learned independently on the source and target environments using step-wise linear regression.

Results: The results in Table II reveal a very high and, in several cases, perfect correlations between interactions across environments. For several environmental changes where we previously could not find a strong evidence of transferable knowledge by previous metrics: ec_8 in x264, $ec_{4,6,7}$ in SQLite and ec_{14} in SaC, we observed very strong correlations for the interactions. The implication for transfer learning is that a linear transfer function (see H1.1) may not be applicable for severe changes, while a complex transfer function may exist.

Insight. The importance of interactions is typically preserved across environments.

VII. INVALID CONFIGURATIONS SIMILARITY (RQ4)

For investigating similarity between invalid configurations across environments, we formulate two hypotheses about percentage of invalid configurations and their commonalities

across environments (H4.1) and the existence of reusable knowledge that can distinguish invalid configurations (H4.2).

H4.1: The percentage of invalid configurations is similar across environments and this percentage is considerable.

Importance. If the percentage of invalid configurations is considerable in the source and target environments, this provides a motivation to carry any information about the invalid configurations across environments to avoid exploration of invalid regions and reduce measurement effort.

Metric. We measure **M15/M16**: percentage of invalid configurations in the source and target, **M17**: percentage of invalid configurations, which are common between environments.

Results. The results in Table II show that for SPEAR and x264, a considerable percentage ($\approx 50\%$) of configurations are invalid and all of them are common across environments. For SaC, approximately 18% of the sampled configurations are invalid. For some workload changes the percentage of common invalid configuration is low ($\leq 10\%$). The reason is that some options in SaC may have severe effects for some programs to be compiled, but have lower effects for others.

Insight. A large percentage of configurations is typically invalid for configurable systems and a large proportion of invalid configurations are common across environments.

H4.2: A classifier for distinguishing invalid from valid configurations is reusable across environments.

Importance. If there are common characteristics among the invalid configurations, we can learn a *classifier* in the source to identify the invalid configurations and transfer the knowledge (classifier model) to the target environment to predict invalid configurations before measuring them.

Metric. We learn a classifier using multinomial logistic regression [4]. It is a model that is used to predict the probabilities of being invalid, given a set of configuration options. We measure **M18**: the correlation between the coefficients (i.e., the probability of the configuration being invalid) of the classification models that has been learned independently.

Results. The results in Table II show that for SPEAR and x264, the correlation between the coefficients are almost perfect. For SaC, in environmental changes where the common invalid configurations are high, the correlation between coefficients is also very high. For two cases, $ec_{6,7}$ in SPEAR, we could not find any reusable knowledge previously with other metrics. Here, we can observe that even when the influence and interactions of all options change, the region of invalid configurations may stay the same. This means that we can avoid measurements (almost half of the space) in the target.

Insight. Information for distinguishing invalid regions can be transferred across environments.

VIII. DISCUSSION

Based on our analyses of 36 environment changes, we can now discuss implications and threats to validity.

A. Implications for transfer learning research

This empirical study showed that transfer learning is not accidental or as a result of over-fit. For the first time, we provide explanations of when and why transfer learning works for performance modeling and analysis of highly configurable systems. While all research questions have positive answers for some environment changes and negative answers for others, as discussed above in Section IV–Section VII, the results align well with our expectations regarding the severity of change and their correspondence to the type of transferable knowledge: (i) For small environmental changes, the overall performance behavior was consistent across environments and a linear transformation of performance models provide a good approximations for the target performance behavior. (ii) For large environmental changes, we found evidence that individual influences of configuration options and interactions may stay consistent providing opportunities for a non-linear mapping between performance behavior across environments. (iii) Even for severe environmental changes, we found evidence of transferable knowledge in terms of reusability of detecting invalid from valid configurations providing opportunities for avoiding a large part of configuration space for sampling.

The fact that we could largely predict the severity of change without deep knowledge about the configuration spaces or implementations of the subject systems is encouraging in the sense that others will likely also be able to make intuitive judgments about transferability of knowledge. For example, a user of a performance analysis approach estimating low severity of an environment change can test this hypothesis quickly with a few measurements and select the right transfer learning strategy. Transfer learning approaches for easy environment changes are readily available [7], [21], [42], [49].

For more severe environment changes, more research is needed to exploit transferable knowledge. Our result show that that even with severe environmental change, there always is some transferable knowledge that can contribute to performance understanding of configurable systems. While some learning strategies can take existing domain knowledge into account and could benefit from knowledge about influential options and interactions [36], [37], it is less obvious how to effectively incorporate such knowledge into sampling strategies and how to build more effective learners based on limited transferable knowledge. While we strongly suspect that suitable transfer learning techniques can provide significant benefits even for severe environment changes, more research is needed to design and evaluate such techniques and compare it to state of the art sampling and learning strategies. Specifically, we expect research opportunities regarding (i) *Sampling strategies* to exploit the relatedness of source and target to provide more informative samples exploiting the importance of specific regions or avoiding invalid regions. (ii) *Learning mechanisms* to exploit the relatedness across environments and learn either a linear or non-linear associations in order to learn faster, reliable and cheaper models. (iii) *Performance testing and debugging* of configurable systems to benefit from our findings by transferring interesting test cases where interactions between options that need to be covered.

B. Threats to validity

1) *External validity*: We selected a diverse set of subject systems and a large number of purposefully selected environment changes, but, as usual, one has to be careful when generalizing to other subject systems and environment changes. We actually performed experiments with more environmental changes and with additional measurements on the same subject systems (e.g., for SaC we also measured the time it takes to compile the program not only its execution), but we excluded those results because they were consistent with the presented data and did not provide additional insights.

2) *Internal and construct validity*: Due to the size of configuration spaces, we could only measure configurations exhaustively in one subject system and had to rely on sampling (with substantial sampling size) for the others, which may miss effects in parts of the configuration space that we did not sample. We did not encounter any surprisingly different observation in our exhaustively measured SPEAR dataset.

We operationalized a large number of different measures through metrics. For each measure, we considered multiple alternative metrics (e.g., different ways to establish influential options) but settled usually on the simplest and most reliable metric we could identify to keep the paper accessible and within reasonable length. In addition, we only partially used statistical tests, as needed, and often compared metrics directly using more informal comparisons and some ad-hoc threshold for detecting common patterns across environments. A different operationalization may lead to different results, but since our results often have a strong effect size and are largely consistent across a large number of questions and measures, we do not expect any changes to the overall big picture.

For building the performance models, calculating importance of configuration options, and classifying the invalid configurations, we elected to use different machine learning models: step-wise linear regression, regression trees, and multinomial logistic regression. We chose these learner mainly because they are successful models that have been used in previous work for performance predictions of configurable systems. However, these are only few learning mechanisms out of many that may provide different accuracy and cost.

Measurement noise in benchmarks can be reduced but not avoided. We performed benchmarks on dedicated systems and repeated each measurement 3 times. We repeated experiments when we encountered unusually large deviations.

IX. RELATED WORK

A. Performance analysis of configurable software

Performance modeling and analysis is a highly researched topic [43]. Researches investigate what models are more suitable for predicting performance of the configurable systems, which sampling and optimization strategies can be used for tuning these models, and how to minimize the amount of measurement efforts for model training.

Sampling strategies based on experimental design (such as Plackett-Burman) have been applied in the domain of configurable systems [13], [35], [36]. The aim of these sampling approaches is to ensure that we gain a high level of information from sparse sampling in high dimensional spaces.

Optimization algorithms have also been applied to find optimal configurations for configurable systems: Recursive random sampling [46], hill climbing [45], direct search [49], optimization via guessing [31], Bayesian optimization [19], and multi-objective optimization [11]. The aim of optimization approaches is to find the optimal configuration in a highly dimensional space using only a limited sampling budget.

Machine learning techniques, such as support-vector machines [47], decision trees [27], Fourier sparse functions [48], active learning [36] and search-based optimization and evolutionary algorithms [14], [44] have also been used.

Our work is related to the performance analysis research mentioned above. However, we do not perform a comparison of different models, configuration optimization or sampling strategies. Instead, we concentrate on transferring performance models across hardware, workload and software version. Transfer learning, in general, is orthogonal to these approaches and can contribute to make these approaches more efficient for performance modeling and analysis.

B. Performance analysis across environmental change

Environmental changes have been studied before. For example, in the context of MapReduce applications [47], performance-anomaly detection [38], performance prediction based on micro-benchmark measurements on different hardware [18], consistency-analysis of parameter dependencies [49], and performance prediction of configurable systems based on hardware variants and similarity search [40].

Recently, transfer learning is used in systems and software engineering. For example, in the context of performance predictions in self-adaptive systems [21], configuration dependency transfer across software systems [7], co-design exploration for embedded systems [5], model transfer across hardware [42], and configuration optimization [3]. Although previous work has analyzed transfer learning in the context of select hardware changes [7], [21], [42], we more broadly empirically investigate why and when transfer learning works. That is, we provide evidence why and when other techniques are applicable for which environmental changes.

Transfer learning has also been applied in software engineering in very different contexts, including defect predictions [24], [28], [29] and effort estimation [23].

X. CONCLUSIONS

We investigated when and why transfer learning works for performance modeling and analysis of highly configurable systems. Our results suggest that performance models are frequently related across environments regarding overall performance response, performance distributions, influential configuration options and their interactions, as well as invalid configurations. While some environment changes allow simple linear forms of transfer learning, others have less obvious relationships but can still be exploited by transferring more nuanced aspects of the performance model, e.g., usable for guided sampling. Our empirical study demonstrate the existence of diverse forms of transferable knowledge across environments that can contribute to learning faster, better, reliable, and more important, less costly performance models.

TABLE II: Results.

Environment	ES	RQ1					RQ2					RQ3				RQ4				
		H1.1	H1.2	H1.3	H1.4		H2.1				H2.2		H3.1			H3.2	H4.1			H4.2
		M1	M2	M3	M4	M5	M6	M7	M8	M9	M10	M11	M12	M13	M14	M15	M16	M17	M18	
SPEAR— Workload (#variables/#clauses): $w_1 : 774/5934, w_2 : 1008/7728, w_3 : 1554/11914, w_4 : 978/7498$; Version: $v_1 : 1.2, v_2 : 2.7$																				
$ec_1 : [h_2 \rightarrow h_1, w_1, v_2]$	S	1.00	0.22	0.97	0.92	0.92	9	7	7	0	1	25	25	25	1.00	0.47	0.45	1	1.00	
$ec_2 : [h_4 \rightarrow h_1, w_1, v_2]$	L	0.59	24.88	0.91	0.76	0.86	12	7	4	2	0.51	41	27	21	0.98	0.48	0.45	1	0.98	
$ec_3 : [h_1, w_1 \rightarrow w_2, v_2]$	S	0.96	1.97	0.17	0.44	0.32	9	7	4	3	1	23	23	22	0.99	0.45	0.45	1	1.00	
$ec_4 : [h_1, w_1 \rightarrow w_3, v_2]$	M	0.90	3.36	-0.08	0.30	0.11	7	7	4	3	0.99	22	23	22	0.99	0.45	0.49	1	0.94	
$ec_5 : [h_1, w_1, v_2 \rightarrow v_1]$	L	0.23	0.30	0.35	0.28	0.32	6	5	3	1	0.32	21	7	7	0.33	0.45	0.50	1	0.96	
$ec_6 : [h_1, w_1 \rightarrow w_2, v_1 \rightarrow v_2]$	L	-0.10	0.72	-0.05	0.35	0.04	5	6	1	3	0.68	7	21	7	0.31	0.50	0.45	1	0.96	
$ec_7 : [h_1 \rightarrow h_2, w_1 \rightarrow w_4, v_2 \rightarrow v_1]$	VL	-0.10	6.95	0.14	0.41	0.15	6	4	2	2	0.88	21	7	7	-0.44	0.47	0.50	1	0.97	
x264— Workload (#pictures/size): $w_1 : 8/2, w_2 : 32/11, w_3 : 128/44$; Version: $v_1 : r2389, v_2 : r2744, v_3 : r2744$																				
$ec_1 : [h_2 \rightarrow h_1, w_3, v_3]$	SM	0.97	1.00	0.99	0.97	0.92	9	10	8	0	0.86	21	33	18	1.00	0.49	0.49	1	1	
$ec_2 : [h_2 \rightarrow h_1, w_1, v_3]$	S	0.96	0.02	0.96	0.76	0.79	9	9	8	0	0.94	36	27	24	1.00	0.49	0.49	1	1	
$ec_3 : [h_1, w_1 \rightarrow w_2, v_3]$	S	0.65	0.06	0.63	0.53	0.58	9	11	8	1	0.89	27	33	22	0.96	0.49	0.49	1	1	
$ec_4 : [h_1, w_1 \rightarrow w_3, v_3]$	M	0.67	0.06	0.64	0.53	0.56	9	10	7	1	0.88	27	33	20	0.96	0.49	0.49	1	1	
$ec_5 : [h_1, w_3, v_2 \rightarrow v_3]$	S	0.05	1.64	0.44	0.43	0.42	12	10	10	0	0.83	47	33	29	1.00	0.49	0.49	1	1	
$ec_6 : [h_1, w_3, v_1 \rightarrow v_3]$	L	0.06	1.54	0.43	0.43	0.37	11	10	9	0	0.80	46	33	27	0.99	0.49	0.49	1	1	
$ec_7 : [h_1, w_1 \rightarrow w_3, v_2 \rightarrow v_3]$	L	0.08	1.03	0.26	0.25	0.22	8	10	5	1	0.78	33	33	20	0.94	0.49	0.49	1	1	
$ec_8 : [h_2 \rightarrow h_1, w_1 \rightarrow w_3, v_2 \rightarrow v_3]$	VL	0.09	14.51	0.26	0.23	0.25	8	9	5	2	0.58	33	21	18	0.94	0.49	0.49	1	1	
SQLite— Workload: $w_1 : write - seq, w_2 : write - batch, w_3 : read - rand, w_4 : read - seq$; Version: $v_1 : 3.7.6.3, v_2 : 3.19.0$																				
$ec_1 : [h_3 \rightarrow h_2, w_1, v_1]$	S	0.99	0.37	0.82	0.35	0.31	5	2	2	0	1	13	9	8	1.00	N/A	N/A	N/A	N/A	
$ec_2 : [h_3 \rightarrow h_2, w_2, v_1]$	M	0.97	1.08	0.88	0.40	0.49	5	5	4	0	1	10	11	9	1.00	N/A	N/A	N/A	N/A	
$ec_3 : [h_2, w_1 \rightarrow w_2, v_1]$	S	0.96	1.27	0.83	0.40	0.35	2	3	1	0	1	9	9	7	0.99	N/A	N/A	N/A	N/A	
$ec_4 : [h_2, w_3 \rightarrow w_4, v_1]$	M	0.50	1.24	0.43	0.17	0.43	1	1	0	0	1	4	2	2	1.00	N/A	N/A	N/A	N/A	
$ec_5 : [h_1, w_1, v_1 \rightarrow v_2]$	M	0.95	1.00	0.79	0.24	0.29	2	4	1	0	1	12	11	7	0.99	N/A	N/A	N/A	N/A	
$ec_6 : [h_1, w_2 \rightarrow w_1, v_1 \rightarrow v_2]$	L	0.51	2.80	0.44	0.25	0.30	3	4	1	1	0.31	7	11	6	0.96	N/A	N/A	N/A	N/A	
$ec_7 : [h_2 \rightarrow h_1, w_2 \rightarrow w_1, v_1 \rightarrow v_2]$	VL	0.53	4.91	0.53	0.42	0.47	3	5	2	1	0.31	7	13	6	0.97	N/A	N/A	N/A	N/A	
SaC— Workload: $w_1 : srad, w_2 : pfilter, w_3 : kmeans, w_4 : hotspot, w_5 : nw, w_6 : nbody100, w_7 : nbody150, w_8 : nbody750, w_9 : gc, w_{10} : cg$																				
$ec_1 : [h_1, w_1 \rightarrow w_2, v_1]$	L	0.66	25.02	0.65	0.10	0.79	13	14	8	0	0.88	82	73	52	0.27	0.18	0.17	0.88	0.73	
$ec_2 : [h_1, w_1 \rightarrow w_3, v_1]$	L	0.44	15.77	0.42	0.10	0.65	13	10	8	0	0.91	82	63	50	0.56	0.18	0.12	0.90	0.84	
$ec_3 : [h_1, w_1 \rightarrow w_4, v_1]$	S	0.93	7.88	0.93	0.36	0.90	12	10	9	0	0.96	37	64	34	0.94	0.16	0.15	0.26	0.88	
$ec_4 : [h_1, w_1 \rightarrow w_5, v_1]$	S	0.96	2.82	0.78	0.06	0.81	16	12	10	0	0.94	34	58	25	0.04	0.15	0.22	0.19	-0.29	
$ec_5 : [h_1, w_2 \rightarrow w_3, v_1]$	M	0.76	1.82	0.84	0.67	0.86	17	11	9	1	0.95	79	61	47	0.55	0.27	0.13	0.83	0.88	
$ec_6 : [h_1, w_2 \rightarrow w_4, v_1]$	S	0.91	5.54	0.80	0.00	0.91	14	11	8	0	0.85	64	65	31	-0.40	0.13	0.15	0.12	0.64	
$ec_7 : [h_1, w_2 \rightarrow w_5, v_1]$	L	0.68	25.31	0.57	0.11	0.71	14	14	8	0	0.88	67	59	29	0.05	0.21	0.22	0.09	-0.13	
$ec_8 : [h_1, w_3 \rightarrow w_4, v_1]$	L	0.68	1.70	0.56	0.00	0.91	14	13	9	1	0.88	57	67	36	0.34	0.11	0.14	0.05	0.67	
$ec_9 : [h_1, w_3 \rightarrow w_5, v_1]$	VL	0.06	3.68	0.20	0.00	0.64	16	10	9	0	0.90	51	58	35	-0.52	0.11	0.21	0.06	-0.41	
$ec_{10} : [h_1, w_4 \rightarrow w_5, v_1]$	M	0.70	4.85	0.76	0.00	0.75	12	12	11	0	0.95	58	57	43	0.29	0.14	0.20	0.64	-0.14	
$ec_{11} : [h_1, w_6 \rightarrow w_7, v_1]$	S	0.82	5.79	0.77	0.25	0.88	36	30	28	2	0.89	109	164	102	0.96	N/A	N/A	N/A	N/A	
$ec_{12} : [h_1, w_6 \rightarrow w_8, v_1]$	S	1.00	0.52	0.92	0.80	0.97	38	30	22	6	0.94	51	53	43	0.99	N/A	N/A	N/A	N/A	
$ec_{13} : [h_1, w_8 \rightarrow w_7, v_1]$	S	1.00	0.32	0.92	0.53	0.99	30	33	26	1	0.98	53	89	51	1.00	N/A	N/A	N/A	N/A	
$ec_{14} : [h_1, w_9 \rightarrow w_{10}, v_1]$	L	0.24	4.85	0.56	0.44	0.77	22	21	18	3	0.69	237	226	94	0.86	N/A	N/A	N/A	N/A	

ES: Expected severity of environmental change (see Sec. III-B): *S*: small change; *SM*: small medium change; *M*: medium change; *L*: large change; *VL*: very large change.

SaC workload descriptions: *srad*: random matrix generator; *pfilter*: particle filtering; *hotspot*: heat transfer differential equations; *k-means*: clustering; *nw*: optimal matching; *nbody*: simulation of dynamic systems; *cg*: conjugate gradient; *gc*: garbage collector. Hardware descriptions (ID: Type/CPU/Clock (GHz)/RAM (GiB)/Disk):

h1: NUC/4/1.30/15/SSD; h2: NUC/2/2.13/7/SCSI; h3: Station/2/2.8/3/SCSI; h4: Amazon/1/2.4/1/SSD; h5: Amazon/1/2.4/0.5/SSD; h6: Azure/1/2.4/3/SCSI

REFERENCES

- [1] A. Ahmad, P. Jamshidi, and C. Pahl. Classification and comparison of architecture evolution reuse knowledge - a systematic review. *Journal of Software: Evolution and Process*, 26(7):654–691, 2014.
- [2] J. P. S. Alcocer, A. Bergel, S. Ducasse, and M. Denker. Performance evolution blueprint: Understanding the impact of software evolution on performance. In *Proc. of Working Conference on Software Visualization (VISSOFT)*, pages 1–9. IEEE, 2013.
- [3] M. Artač, editor. *Deliverable 5.2: DICE delivery tools-Intermediate version*. 2017. <http://www.dice-h2020.eu/>.
- [4] C. M. Bishop. *Pattern recognition and machine learning*. Springer, New York, 2006.
- [5] B. Bodin, L. Nardi, M. Z. Zia, H. Wagstaff, G. Sreekar Shenoy, M. Emani, J. Mawer, C. Kotselidis, A. Nisbet, M. Lujan, B. Franke, P. H. Kelly, and M. O’Boyle. Integrating algorithmic parameters into benchmarking and design space exploration in 3D scene understanding. In *Proceedings of the International Conference on Parallel Architectures and Compilation (PACT)*, pages 57–69. ACM, 2016.
- [6] A. Brunner, A. van Hoorn, F. Willnecker, A. Danciu, W. Hasselbring, C. Heger, N. Herbst, P. Jamshidi, R. Jung, J. von Kistowski, A. Koziolok, J. Kross, S. Spinner, C. Vögele, J. Walter, and A. Wert. Performance-oriented toward: A research agenda. *SPEC-RG-2015-01, RG DevOps Performance*, 2015.
- [7] H. Chen, W. Zhang, and G. Jiang. Experience transfer for the configuration tuning in large-scale computing systems. *IEEE Trans. on Knowledge and Data Eng. (TKDE)*, 23(3):388–401, 2011.
- [8] T. M. Cover and J. A. Thomas. *Elements of information theory*. John Wiley & Sons, 2012.
- [9] A. Elkhodary, N. Esfahani, and S. Malek. Fusion: A framework for engineering self-tuning self-adaptive software systems. In *Proc. Int’l Symp. Foundations of Software Engineering (FSE)*, pages 7–16. ACM, 2010.
- [10] N. Esfahani, A. Elkhodary, and S. Malek. A learning-based framework for engineering feature-oriented self-adaptive software systems. *IEEE Trans. Softw. Eng. (TSE)*, 39(11):1467–1493, 2013.
- [11] A. Filieri, H. Hoffmann, and M. Maggio. Automated multi-objective control for self-adaptive software design. In *Proc. Int’l Symp. Foundations of Software Engineering (FSE)*, pages 13–24. ACM, 2015.
- [12] A. Grebhahn, N. Siegmund, H. Köstler, and S. Apel. Performance prediction of multigrid-solver configurations. In *Software for Exascale Computing-SPPEXA 2013-2015*, pages 69–88. Springer, 2016.
- [13] J. Guo, K. Czarnecki, S. Apel, N. Siegmund, and A. Wasowski. Variability-aware performance prediction: A statistical learning approach. In *Proc. Int’l Conf. Automated Software Engineering (ASE)*, pages 301–311. IEEE, 2013.
- [14] C. Henard, M. Papadakis, M. Harman, and Y. Le Traon. Combining multi-objective search and constraint solving for configuring large software product lines. In *Proc. Int’l Conf. Software Engineering (ICSE)*, volume 1, pages 517–528. IEEE Computer Society, 2015.
- [15] H. Hoffmann, S. Sidiroglou, M. Carbin, S. Misailovic, A. Agarwal, and M. Rinard. Dynamic knobs for responsive power-aware computing. In *ACM SIGPLAN Notices*, volume 46, pages 199–212. ACM, 2011.
- [16] H. H. Hoos. Automated algorithm configuration and parameter tuning. In *Autonomous search*, pages 37–71. Springer, 2011.
- [17] H. H. Hoos. Programming by optimization. *Communications of the ACM*, 55(2):70–80, 2012.
- [18] K. Hoste, A. Phansalkar, L. Eeckhout, A. Georges, L. K. John, and K. De Bosschere. Performance prediction based on inherent program similarity. In *Proc. of the International Conference on Parallel Architectures and Compilation Techniques (PACT)*, pages 114–122. ACM, 2006.
- [19] P. Jamshidi and G. Casale. An uncertainty-aware approach to optimal configuration of stream processing systems. In *Proc. Int’l Symp. on Modeling, Analysis and Simulation of Computer and Telecommunication Systems (MASCOTS)*, pages 39–48, Los Alamitos, CA, USA, September 2016. IEEE Computer Society.
- [20] P. Jamshidi, M. Ghafari, A. Ahmad, and C. Pahl. A framework for classifying and comparing architecture-centric software evolution research. In *Proc. of European Conference on Software Maintenance and Reengineering (CSMR)*, pages 305–314. IEEE, 2013.
- [21] P. Jamshidi, M. Velez, C. Kästner, N. Siegmund, and P. Kawthekar. Transfer learning for improving model predictions in highly configurable software. In *Proc. Int’l Symp. Software Engineering for Adaptive and Self-Managing Systems (SEAMS)*. IEEE Computer Society, May 2017.
- [22] P. Kawthekar and C. Kästner. Sensitivity analysis for building evolving and & adaptive robotic software. In *Proceedings of the IJCAI Workshop on Autonomous Mobile Service Robots (WSR)*, 7 2016.
- [23] E. Kocaguneli, T. Menzies, and E. Mendes. Transfer learning in effort estimation. *Empirical Software Engineering*, 20(3):813–843, 2015.
- [24] R. Krishna, T. Menzies, and W. Fu. Too much automation? The bellwether effect and its implications for transfer learning. In *Proc. Int’l Conf. Automated Software Engineering (ASE)*, pages 122–131. ACM, 2016.
- [25] D. R. Kuhn, R. N. Kacker, and Y. Lei. *Introduction to combinatorial testing*. CRC press, 2013.
- [26] A. Murashkin, M. Antkiewicz, D. Rayside, and K. Czarnecki. Visualization and exploration of optimal variants in product line engineering. In *Proc. Int’l Software Product Line Conference (SPLC)*, pages 111–115. ACM, 2013.
- [27] V. Nair, T. Menzies, N. Siegmund, and S. Apel. Faster discovery of faster system configurations with spectral learning. *arXiv preprint arXiv:1701.08106*, 2017.
- [28] J. Nam and S. Kim. Heterogeneous defect prediction. In *Proc. Int’l Symp. Foundations of Software Engineering (FSE)*, pages 508–519. ACM, 2015.
- [29] J. Nam, S. J. Pan, and S. Kim. Transfer defect learning. In *Proc. Int’l Conf. Software Engineering (ICSE)*, pages 382–391. IEEE Press, 2013.
- [30] R. Olacoea, D. Rayside, J. Guo, and K. Czarnecki. Comparison of exact and approximate multi-objective optimization for software product lines. In *Proc. Int’l Software Product Line Conference (SPLC)*, pages 92–101. ACM, 2014.
- [31] T. Osogami and S. Kato. Optimizing system configurations quickly by guessing at the performance. In *Proc. of International Conference on Measurement and Modeling of Computer Systems (SIGMETRICS)*, volume 35, 2007.
- [32] S. J. Pan and Q. Yang. A survey on transfer learning. *IEEE Trans. on Knowledge and Data Eng. (TKDE)*, 22(10):1345–1359, 2010.
- [33] SaC compiler. www.sac-home.org.
- [34] A. Saltelli, M. Ratto, T. Andres, F. Campolongo, J. Cariboni, D. Gatelli, M. Saisana, and S. Tarantola. *Global sensitivity analysis: the primer*. John Wiley & Sons, 2008.
- [35] A. Sarkar, J. Guo, N. Siegmund, S. Apel, and K. Czarnecki. Cost-efficient sampling for performance prediction of configurable systems. In *Proc. Int’l Conf. Automated Software Engineering (ASE)*, pages 342–352, Washington, DC, USA, November 2015. IEEE Computer Society.
- [36] N. Siegmund, A. Grebhahn, S. Apel, and C. Kästner. Performance-influence models for highly configurable systems. In *Proc. Europ. Software Engineering Conf. Foundations of Software Engineering (ESEC/FSE)*, pages 284–294, New York, NY, USA, August 2015. ACM.
- [37] N. Siegmund, S. S. Kolesnikov, C. Kästner, S. Apel, D. Batory, M. Rosenmüller, and G. Saae. Predicting performance via automated feature-interaction detection. In *Proc. Int’l Conf. Software Engineering (ICSE)*, pages 167–177. IEEE, 2012.
- [38] C. Stewart, K. Shen, A. Iyengar, and J. Yin. Entomomodel: Understanding and avoiding performance anomaly manifestations. In *Proc. Int’l Symp. on Modeling, Analysis and Simulation of Computer and Telecommunication Systems (MASCOTS)*, pages 3–13. IEEE Computer Society, 2010.
- [39] J. Styles, H. H. Hoos, and M. Müller. Automatically configuring algorithms for scaling performance. In *Learning and Intelligent Optimization*, pages 205–219. Springer, 2012.
- [40] E. Thereska, B. Doebel, A. X. Zheng, and P. Nobel. Practical performance models for complex, popular applications. In *SIGMETRICS Perform. Eval. Rev.*, volume 38, pages 1–12. ACM, 2010.
- [41] L. Torrey and J. Shavlik. Transfer learning. *Handbook of Research on Machine Learning Applications and Trends: Algorithms, Methods, and Techniques*, 1:242–264, 2009.
- [42] P. Valov, J.-C. Petkovich, J. Guo, S. Fischmeister, and K. Czarnecki. Transferring performance prediction models across different hardware platforms. In *Proc. Int’l Conf. on Performance Engineering (ICPE)*, pages 39–50. ACM, 2017.
- [43] M. Woodside, G. Franks, and D. C. Petriu. The future of software performance engineering. In *Future of Software Engineering (FOSE)*, pages 171–187. IEEE, 2007.
- [44] F. Wu, W. Weimer, M. Harman, Y. Jia, and J. Krinke. Deep parameter optimisation. In *Proc. of the Annual Conference on Genetic and Evolutionary Computation*, pages 1375–1382. ACM, 2015.
- [45] B. Xi, Z. Liu, M. Raghavachari, C. H. Xia, and L. Zhang. A smart hill-climbing algorithm for application server configuration. In *13th International Conference on World Wide Web (WWW)*, pages 287–296. ACM, 2004.
- [46] T. Ye and S. Kalyanaraman. A recursive random search algorithm for large-scale network parameter configuration. *ACM SIGMETRICS International Conference on Measurement and Modeling of Computer Systems*, 31(1):196–205, 2003.

- [47] N. Yigitbasi, T. L. Willke, G. Liao, and D. Epema. Towards machine learning-based auto-tuning of mapreduce. In *Proc. Int'l Symp. on Modeling, Analysis and Simulation of Computer and Telecommunication Systems (MASCOTS)*, pages 11–20. IEEE Computer Society, 2013.
- [48] Y. Zhang, J. Guo, E. Blais, and K. Czarnecki. Performance prediction of configurable software systems by Fourier learning. In *Proc. Int'l Conf. Automated Software Engineering (ASE)*, pages 365–373, 2015.
- [49] W. Zheng, R. Bianchini, and T. D. Nguyen. Automatic configuration of internet services. *ACM SIGOPS Operating Systems Review*, 41(3):219–229, 2007.