Transfer Learning for Improving Model Predictions in Highly Configurable Software

Pooyan Jamshidi, Miguel Velez, Christian Kästner
Carnegie Mellon University, USA
{pjamshid,mvelezce,kaestner}@cs.cmu.edu

Norbert Siegmund
Bauhaus-University Weimar, Germany
norbert.siegmund@uni-weimar.de

Prasad Kawthekar
Stanford University, USA
pkawthek@stanford.edu

***Abstract*—Modern software systems are built to be used in dynamic environments using configuration capabilities to adapt to changes and external uncertainties. In a self-adaptation context, we are often interested in reasoning about the performance of the systems under different configurations. Usually, we learn a black-box model based on real measurements to predict the performance of the system given a specific configuration. However, as modern systems become more complex, there are many configuration parameters that may interact and we end up learning an exponentially large configuration space. Naturally, this does not scale when relying on real measurements in the actual changing environment. We propose a different solution: Instead of taking the measurements from the real system, we learn the model using samples from other sources, such as simulators that approximate performance of the real system at low cost. We define a cost model that transform the traditional view of model learning into a multi-objective problem that not only takes into account model accuracy but also measurements effort as well. We evaluate our cost-aware transfer learning solution using real-world configurable software including (i) a robotic system, (ii) 3 different stream processing applications, and (iii) a NoSQL database system. The experimental results demonstrate that our approach can achieve (a) a high prediction accuracy, as well as (b) a high model reliability.**

***Index Terms*—highly configurable software, machine learning, model learning, model prediction, transfer learning**

I. Introduction

Most software systems today are configurable, which gives end users, developers, and administrators the chance to customize the system to achieve a different functionality or tune its performance. In such systems, hundreds or even thousands of configuration parameters can be tweaked, making the system highly configurable [36]. The exponentially growing configuration space, complex interactions, and unknown constraints among configuration options make it difficult to understand the performance of the system. As a consequence, many users rely on default configurations or they change only individual options in an ad-hoc way.

In this work, we deal with the type of configurable systems that operate in dynamic and uncertain environments (*e.g.*, robotic systems). Therefore, it is desirable to react to environmental changes by tuning the configuration of the system when we anticipate that the performance will drop to an undesirable level. To do so, we use black-box performance models that describe how configuration options and their interactions influence the performance of a system (*e.g.*, execution time). Black-box performance models are meant to ease understanding, debugging, and optimization of configurable systems [36]. For example, a reasoning algorithm may use the learned model in order to identify the best performing configuration for a robot when it goes from indoor to an outdoor environment.

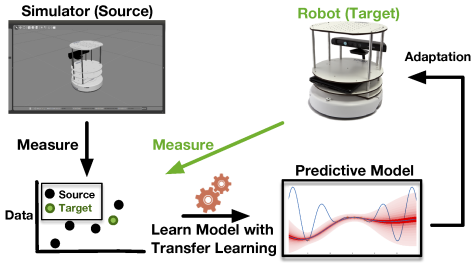


Fig. 1: Transfer learning for performance model learning.

Typically, we learn a performance model for a given configurable system by measuring from a set of configurations selected by some sampling strategy. That is, we measure the performance of a given system multiple times in different configurations and learn how the configuration options and their interactions affect performance. However, such a way of learning from real systems, whether it is a robot or a software application, is a difficult task for several reasons: (i) environmental changes (*e.g.*, people wandering around robots), (ii) high costs or risks of failure (*e.g.*, a crashed robot), (iii) the large amount of time required for measurements (*e.g.*, we have to repeat the measurements several times to get a reliable value), and (iv) changing system dynamics (*e.g.*, robot motion). Moreover, it is often not possible to create potentially important scenarios in the real environment.

In this paper, as depicted in Figure 1, we propose a different solution: instead of taking the measurements from the real system, we reuse prior information (that we can get from other sources at a lower cost) in order to learn a performance model for the real system faster and cheaper. The concept of reusing information from other sources is the idea behind *transfer learning* [38], [32]. Similar to human beings that can learn from previous experience and transfer the learning to accomplish new tasks easier, quicker, and in a better way, in this work, we use other sources to provide cheaper samples for accelerating model learning. Instead of taking the measurements from the real system (we refer to as the *target*), we measure the system performance using a proxy of the system (we refer to as the *source*, *e.g.*, a simulator). We then use a regression model that automatically learns the relationship between source and target to learn an accurate and reliable performance model using only a few samples taken from the real system, leading to much lower cost and faster learning

period. We define a cost model that turns model learning into a multi-objective problem taking into account model accuracy as well as measurement cost. We demonstrate that our cost-aware transfer learning will enable accurate performance predictions by using only a few measurements of the target system. We evaluate our approach on (i) a robotic system, (ii) 3 stream processing applications, and (iii) a NoSQL database system.

In summary, our contributions are the following:

- A *cost-aware transfer learning* method that learns accurate performance models for configurable software.
- An implementation, analysis and *experimental evaluation* of our cost-aware transfer learning compared to a no transfer learning method.

II. Model Learning for Performance Reasoning in Highly Configurable Software

We motivate the model learning problem by reviewing specific challenges in the robotics domain. However, our method can be applied in other configurable systems as well.

A. A motivating example

The software embedded in a robot implements algorithms that enable the robot to accomplish the missions assigned to it, such as path planning [24]. Often, robotics software exposes many different configuration parameters that can be tuned and typically affect the robot's performance in accomplishing its missions. For instance, we would like to tune the localization parameters of an autonomous robot while performing a navigation task as *fast* and *safe* as possible while consuming *minimal energy* (corresponding to longer battery life).

Navigation is the most common activity that a robot does in order to get from point A to point B in an environment as efficient as possible without hitting obstacles, walls, or people. Let us concentrate on the *localization* algorithm that enables navigation tasks and previous studies shown that configuration setting is influential to its performance [24]. Localization is the technique that enables the robot to find (i) its current position and (ii) its current orientation at any point in its navigation task [37]. Configurations that work well for one robot in one environment may not work well or at all in another environment for multiple reasons: (i) **The environment in which robots operate may change at runtime**. For instance, the robot may start in a dark environment and then reaches a much brighter area. As a result, the quality of localization may be affected as the range sensors for estimating the distance to the wall provide measurements with a higher error rate and the false readings increase then. (ii) **The robot itself may move to an environment where the motion of the robot will change**. For example, when a wheeled robot moves to a slippery floor, then the motion dynamics of the robot will change and the robot cannot accelerate the speed with the same power level. (iii) **The physics of the environment may change**. For example, imagine the situation where some items are put on a service robot while doing a mission. As a result, the robot has to sustain more weight and consumes more energy for doing the same mission. Therefore, it may be desirable to sacrifice the accuracy by adjusting localization parameters in order to perform the mission before running out of battery. (iv) **A new mission may be assigned to the robot**. While a service robot is doing a mission, a new mission may be assigned, for example, a new delivery spot may be defined. In cases where the battery level is too low to finish the new mission, we would change to a configuration that sacrifices the localization accuracy in favor of energy usage.

In all of these situations, it is desirable to change the configuration regarding the localization of the robot, in an automated fashion. For doing so, we need a model to reason about the system performance especially when we face contradicting aspects that require a trade-off at runtime. We can learn a performance model empirically using observations which are taken from the robot performing missions in a real environment under different configurations. But, the observations in real-world are typically costly, error-prone, and sometimes infeasible. There are, however, several simulation platforms that provide a simulated environment, in which we can learn about the performance of the robot in different conditions given a specific configuration (*e.g.*, Gazebo [34]). Although the observations from a simulated environment are far less costly than the measurements taken from the real robot and impose no risks when a failure happens, they may not necessarily reflect real behavior, since not every physical aspect can be accurately modeled and simulated. Fortunately, simulations are often highly *correlated* with the real behavior. The key insight behind our approach is to learn the robot performance model by using only a few expensive observations on a real system, but several cheap samples from all sources.

B. Challenges

Although we can take relatively cheap samples from simulation, it is impractical and naive to exhaustively run simulation environments for all possible configurations:

- **Exponentially growing space**. The configuration space of just 20 parameters with binary options for each comprises of $2^{20} \approx 1m$ possible configurations. Even for this small configuration space, if we spend just one minute for collecting each sample, it will take about 2 years to perform an exhaustive sampling. Therefore, we can sample only a subset of this space, and we need to do the sampling purposefully.
- **Negative transfer**. Not all samples from a simulator reflect the real behavior and, as a result, they would not be useful for learning a performance model. More specifically, the measurements from simulators typically contain noise and for some configurations, the data may not have any relationship with the data we may observe on the real system. Therefore, if we learn a model based on these data, it becomes inaccurate and far from real system behavior. Also, any reasoning based on the model predictions becomes misleading or ineffective.
- **Limited budget**. Often, different sources of data exist (*e.g.*, different simulators, different versions of a system) that we can learn from and each may impose a different cost. Often, we are given a limited budget that we can spend for either source and target sample measurements.

C. Problem formulation: Black-box model learning

In order to introduce the concepts in our approach concisely, we define the model learning problem using mathematical

notations. Let X_i indicate the i-th configuration parameter, which ranges in a finite domain $Dom(X_i)$. In general, X_i may either indicate (i) an integer variable (*e.g.*, the *number of iterative refinements* in a localization algorithm) or (ii) a categorical variable (*e.g.*, *sensor names* or binary options (*e.g.*, *local vs global localization method*). The configuration space is mathematically a Cartesian product of all of the domains of the parameters of interest $\mathbb{X} = Dom(X_1) \times \cdots \times Dom(X_d)$.

A black-box response function $f : \mathbb{X} \rightarrow \mathbb{R}$ is used to build a performance model given some observations of the system performance under different settings. In practice, though, the observation data may contain noise, *i.e.*, $y_i = f(\boldsymbol{x}_i) + \epsilon_i, \boldsymbol{x}_i \in \mathbb{X}$ where $\epsilon_i \sim \mathcal{N}(0, \sigma_i)$ and we only partially know the response function through the observations $\mathcal{D} = \{(\boldsymbol{x}_i, y_i)\}_{i=1}^{d}, |\mathcal{D}| \ll |\mathbb{X}|$. In other words, a response function is simply a mapping from the configuration space to a measurable performance metric that produces interval-scaled data (here we assume it produces real numbers). Note that we can learn a model for all measurable attributes (including accuracy, safety if suitably operationalized), but here we mostly train predictive models on performance attributes (*e.g.*, response time, throughput, CPU utilization).

Our main goal is to learn a reliable regression model, $\hat{f}(\cdot)$, that can predict the performance of the system, $f(\cdot)$, given a limited number of observations $\mathcal{D}$. More specifically, we aim to minimize the *prediction error* over the configuration space:

$$\arg\min_{\boldsymbol{x} \in \mathbb{X}} pe = |\hat{f}(\boldsymbol{x}) - f(\boldsymbol{x})| \tag{1}$$

In order to solve the problem above, we assume $f(\cdot)$ is reasonably smooth, but otherwise little is known about the response function. Intuitively, we expect that for "near-by" input points $\boldsymbol{x}$ and $\boldsymbol{x}'$ their corresponding output points y and y' to be "near-by" as well.

D. State-of-the-art

In literature, model learning has been approached from two standpoints: (i) sampling strategies and (ii) learning methods.

1) Sampling: Random sampling has been used to collect unbiased observations in computer-based experiments. However, random sampling may require a large number of samples to build an accurate model [36]. More intelligent sampling strategies (such as Box-Behnken and Plackett-Burman) have been developed in the statistics and machine learning communities, under the umbrella of experimental design, to ensure certain statistical properties [28]. The aim of these different experimental designs is to ensure that we gain a high level of information from sparse sampling (partial design) in high dimensional spaces. Relevant to the software engineering community, several approaches tried different designs for highly configurable software [18] and some even consider cost as an explicit factor to determine optimal sampling [35].

For finding optimal configurations, researchers have tried novel ways of sampling with a feedback embedded inside the process where new samples are derived based on information gained from the previous set of samples. A recent solution [8] uses active learning based on a random forest to find a good design. Recursive Random Sampling (RRS) [42] integrates a restarting mechanism into the random sampling to achieve high search efficiency. Smart Hill Climbing (SHC) [41] integrates importance sampling with Latin Hypercube Design (LHD). An approach based on direct search [45] forms a simplex in the configuration space, and iteratively updates the simplex through a number of operations to guide the sample generation. Quick Optimization via Guessing (QOG) [31] speeds up the optimization process exploiting some heuristics to filter out sub-optimal configurations. Recently, transfer learning has been explored for configuration optimizations by exploiting the dependencies between configurations parameters [9] and measurements for previous versions of big data systems using an approach called TL4CO in DevOps [3].

2) Learning: Also, standard machine-learning techniques, such as support-vector machines, decision trees, and evolutionary algorithms have been tried [21], [43]. These approaches trade simplicity and understandability of the learned models for predictive power. For example, some recent work [44] exploited a characteristic of the response surface of the configurable software to learn Fourier sparse functions by only a small sample size. Another approach [36] also exploited this fact, but iteratively construct a regression model representing performance influences in an active learning process.

3) Positioning in the self-adaptive community: Performance reasoning is a key activity for decision making at runtime. Time series techniques [12] shown to be effective in predicting response time and uncovering performance anomalies ahead of time. FUSION [14], [13] exploited inter-feature relationships (*e.g.*, feature dependencies) to reduce the dimensions of configuration space, making runtime performance reasoning feasible. Different classification models have been evaluated for the purpose of time series predictions in [2]. Performance predictions also have been applied for resource allocations at runtime [20], [22]. Note that the approaches above are referred to as black-box models. However, another category of models known as white-box is built early in the life cycle, by studying the underlying architecture of the system [17], [19] using Queuing networks, Petri Nets, and Stochastic Process Algebras [4]. Performance prediction and reasoning have also been used extensively in other communities such as component-based [5] and control theory [15].

4) Novelty: Previous work attempted to improve the prediction power of the model by exploiting the information that has been gained from the target system either by improving the sampling process or by adopting a learning method. Our approach is orthogonal to both sampling and learning, proposing a new way to enhance model accuracy by exploiting the knowledge we can gain from other *relevant* and possibly *cheaper* sources to accelerate the learning of a performance model through transfer learning.

Transfer learning has been applied previously for regression and classification problems in machine learning [32], in software engineering for defect predictions [27], [29], [30] and effort estimation [26] and in systems for configuration optimization [9], [3]. However, in this paper, we enable a generic form of transfer learning for the purpose of sensitivity analysis over the entire configuration space. Our approach can enable (i) performance debugging, (ii) performance tuning, (iii) design-time evolution, or (iv) runtime adaptation in the target environment by exploiting any source of knowledge

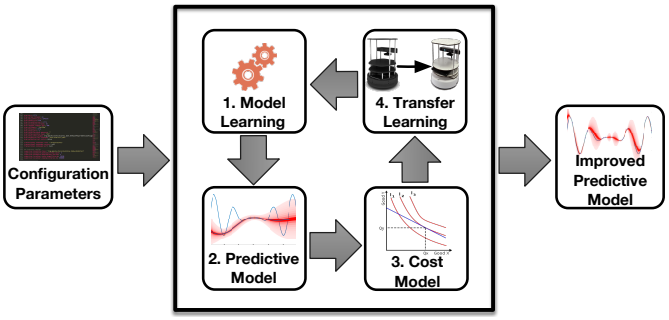


Fig. 2: Overview of cost-aware transfer learning methodology.

from the source environment including (i) different workloads, (ii) different deployments, (iii) different versions of the system, or (iv) different environmental conditions. The performance of the system varies when one or more of these changes happen, however, as we will show later in our experimental results, the performance responses are correlated. This correlation is an indicator that the source and target are related and there is a potential to learn across the environments. To the best of our knowledge, our approach is the first attempt towards learning performance models using cost-aware transfer learning for configurable software.

III. Cost-Aware Transfer Learning

In this section, we explain our *cost-aware transfer learning* solution to improve learning an accurate performance model.

A. Solution overview

An overview of our model learning methodology is depicted in Figure 2. We use the observations that we can obtain cheaply, with the cost of c_s for each sample, from an alternative measurable response function, $g(\cdot)$, that yields different but related response to the target response, $\mathcal{D}_s = \{(\boldsymbol{x}_s, y_s)|y_s = g(\boldsymbol{x}_s) + \epsilon_s, \boldsymbol{x}_s \in \mathbb{X}\}$, and transfer these observations to learn a performance model for the real system using only few observations from that system, $\mathcal{D}_t = \{(\boldsymbol{x}_t, y_t)|y_t = f(\boldsymbol{x}_t) + \epsilon_t, \boldsymbol{x}_t \in \mathbb{X}\}$. The cost of obtaining each observation on the target system is c_t, and we assume that $c_s \ll c_t$ and that the source response function, $g(\cdot)$, is related (correlated) to the target function, and this relatedness contributes to the learning of the target function, using only sparse samples from the real system, *i.e.*, $|\mathcal{D}_t| \ll |\mathcal{D}_s|$. Intuitively, rather than starting from scratch, we transfer the learned knowledge from the data we have collected using cheap methods such as simulators or previous system versions, to learn a more accurate model about the target configurable system and use this model to reason about its performance at runtime.

In Figure 3, we demonstrate the idea of transfer learning with a synthetic example. Only three observations are taken from a synthetic target function, $f(\cdot)$, and we try to learn a model, $\hat{f}(\boldsymbol{x})$, which provides predictions for unobserved response values at some input locations. Figure 3(b) depicts the predictions that are provided by the model trained using only the 3 samples without considering the transfer learning from any source. The predictions are accurate around the three observations, while highly inaccurate elsewhere. Figure 3(c) depicts the model trained over the 3 observations and

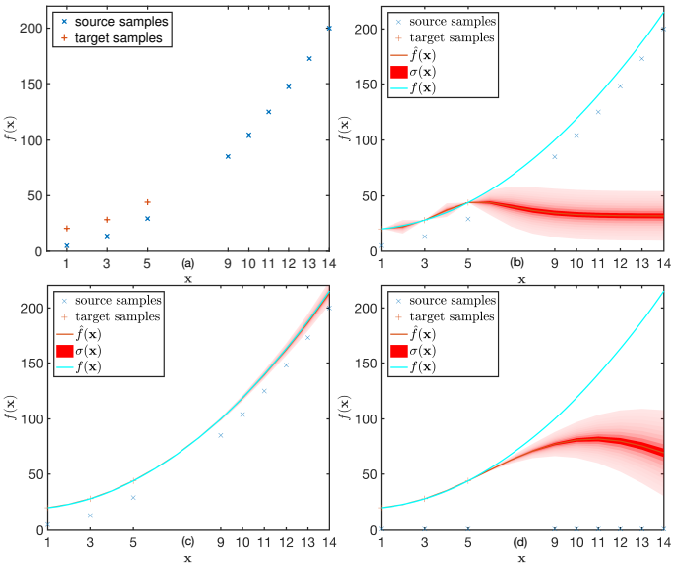


Fig. 3: (a) 9 samples have been taken from a source (respectively 3 from a target); (b) [without transfer learning] a regression model is constructed using only the target samples. (c) [with transfer learning] another regression model is constructed using the target and the source samples. The new model is based on only 3 target measurements while providing accurate predictions. (d) [negative transfer] a regression model is constructed with a transfer from a misleading response function and therefore the model prediction is inaccurate.

9 other observations from a different but related source. The predictions, thanks to transfer learning, are here closely following the true function with a narrow confidence interval (*i.e.*, high confidence in predictions provided by the model). Interestingly, the confidence interval around points $\boldsymbol{x} = 2, 4$ in the model with transfer learning have disappeared, demonstrating more confident predictions with transfer learning.

Given a target environment, the effectiveness of any transfer learning depends on the source environment and how it is related to the target [38]. If the relationship is strong and the transfer learning method can take advantage of it, the performance in the target predictions can significantly improve via transfer. However, if the source response is not sufficiently related or if the transfer method does not exploit the relationship, the performance may not improve or even may decrease. We show this case via the synthetic example in Figure 3(d) where the prediction model uses 9 samples from a misleading function that is unrelated to the target function (the response remains constant as we increase $\boldsymbol{x}$); as a consequence, the predictions are very inaccurate and do not follow the growing trend of the true function.

For the choice of the number of samples from the source and target, we use a simple *cost model* (cf. Figure 2) that is based on the number of source and target samples as follows:

$$\mathcal{C}(\mathcal{D}_s, \mathcal{D}_t) = c_s \cdot |\mathcal{D}_s| + c_t \cdot |\mathcal{D}_t| + \mathcal{C}_{Tr}(|\mathcal{D}_s|, |\mathcal{D}_t|), \quad (2)$$

$$\mathcal{C}(\mathcal{D}_s, \mathcal{D}_t) \leq \mathcal{C}_{max}, \quad (3)$$

where $\mathcal{C}_{max}$ is the experimental budget and $\mathcal{C}_{Tr}(|\mathcal{D}_s|, |\mathcal{D}_t|)$ is the cost of model training, which is a function of source

and target sample sizes. Note that this cost model makes the problem we have formulated in Eq. (1) into a multi-objective problem, where not only *accuracy* is considered, but also *cost* can be considered in the transfer learning process.

B. Model learning: Technical details

We use Gaussian Process (GP) models to learn a reliable performance model $\hat{f}(\cdot)$ that can predict unobserved response values. The main motivation to choose GP here is that it offers a framework in which performance reasoning can be done using mean estimates as well as a confidence interval for each estimation. In other words, where the model is confident with its estimation, it provides a small confidence interval; on the other hand, where it is uncertain about its estimations it gives a large confidence interval meaning that the estimations should be used with precautions. The other reason is that all the computations are based on linear algebra which is cheap to compute. This is especially useful in the domain of self-adaptive systems where automated performance reasoning in the feedback loop is typically time constrained and should be robust against any sort of uncertainty including prediction errors. Also, for building GP models, we do not need to know any internal details about the system; the learning process can be applied in a black-box fashion using the sampled performance measurements. In the GP framework, it is also possible to incorporate domain knowledge as prior, if available, which can enhance the model accuracy [21].

In order to describe the technical details of our transfer learning methodology, let us briefly describe an overview of GP model regression; a more detailed description can be found elsewhere [40]. GP models assume that the function $\hat{f}(\boldsymbol{x})$ can be interpreted as a probability distribution over functions:

$$\boldsymbol{y} = \hat{f}(\boldsymbol{x}) \sim \mathcal{GP}(\mu(\boldsymbol{x}), k(\boldsymbol{x}, \boldsymbol{x}')), \tag{4}$$

where $\mu : \mathbb{X} \to \mathbb{R}$ is the mean function and $k : \mathbb{X} \times \mathbb{X} \to \mathbb{R}$ is the covariance function (kernel function) which describes the relationship between response values, $\boldsymbol{y}$, according to the *distance* of the input values $\boldsymbol{x}, \boldsymbol{x}'$. The mean and variance of the GP model predictions can be derived analytically [40]:

$$\mu_t(\boldsymbol{x}) = \mu(\boldsymbol{x}) + \boldsymbol{k}(\boldsymbol{x})^\intercal (\boldsymbol{K} + \sigma^2 \boldsymbol{I})^{-1} (\boldsymbol{y} - \boldsymbol{\mu}), \tag{5}$$

$$\sigma_t^2(\boldsymbol{x}) = k(\boldsymbol{x}, \boldsymbol{x}) + \sigma^2 \boldsymbol{I} - \boldsymbol{k}(\boldsymbol{x})^\intercal (\boldsymbol{K} + \sigma^2 \boldsymbol{I})^{-1} \boldsymbol{k}(\boldsymbol{x}), \tag{6}$$

where $\boldsymbol{k}(\boldsymbol{x})^\intercal = [k(\boldsymbol{x}, \boldsymbol{x}_1) \quad k(\boldsymbol{x}, \boldsymbol{x}_2) \quad \dots \quad k(\boldsymbol{x}, \boldsymbol{x}_t)]$, $\boldsymbol{I}$ is identity matrix and

$$\boldsymbol{K} := \begin{bmatrix} k(\boldsymbol{x}_1, \boldsymbol{x}_1) & \dots & k(\boldsymbol{x}_1, \boldsymbol{x}_t) \\ \vdots & \ddots & \vdots \\ k(\boldsymbol{x}_t, \boldsymbol{x}_1) & \dots & k(\boldsymbol{x}_t, \boldsymbol{x}_t) \end{bmatrix} \tag{7}$$

GP models have shown to be effective for performance predictions in data scarce domains [21]. However, as we have demonstrated in Figure 3, it may become inaccurate when the samples do not cover the space uniformly. For highly configurable systems, we require a large number of observations to cover the space uniformly, making GP models ineffective in such situations.

C. Model prediction using transfer learning

In transfer learning, the key question is how to make accurate predictions for the target environment using observations from other sources, $\mathcal{D}_s$. We need a measure of relatedness not only between input configurations but between the sources as well. The relationships between input configurations was captured in the GP models using the covariance matrix that was defined based on the kernel function in Eq. (7). More specifically, a kernel is a function that computes a dot product (a measure of "similarity") between two input configurations. So, the kernel helps to get accurate predictions for similar configurations. We now need to exploit the relationship between the source and target functions, g, f, using the current observations $\mathcal{D}_s, \mathcal{D}_t$ to build the predictive model $\hat{f}$. To capture the relationship, we define the following kernel function:

$$k(f, g, \boldsymbol{x}, \boldsymbol{x}') = k_t(f, g) \times k_{xx}(\boldsymbol{x}, \boldsymbol{x}'), \tag{8}$$

where the kernels k_t represent the correlation between source and target function, while k_{xx} is the covariance function for inputs. Typically, k_{xx} is parameterized and its parameters are learned by maximizing the marginal likelihood of the model given the observations from source and target $\mathcal{D} = \mathcal{D}_s \cup \mathcal{D}_t$. Note that the process of maximizing the marginal likelihood is a standard method [40]. After learning the parameters of k_{xx}, we construct the covariance matrix exactly the same way as in Eq. 7 and derive the mean and variance of predictions using Eq. (5), (6) with the new $\boldsymbol{K}$. The main essence of transfer learning is, therefore, the kernel that captures the source and target relationship and provides more accurate predictions using the additional knowledge we can gain via the relationship between source and target.

D. Transfer learning in a self-adaptation loop

Now that we have described the idea of transfer learning for providing more accurate predictions, the question is whether such an idea can be applied at runtime and how the self-adaptive systems can benefit from it. More specifically, we now describe the idea of model learning and transfer learning in the context of self-optimization, where the system adapts its configuration to meet performance requirements at runtime. The difference to traditional configurable systems is that we learn the performance model online in a feedback loop under time and resource constraints. Such performance reasoning is done more frequently for self-adaptation purposes.

An overview of a self-optimization solution is depicted in Figure 4 following the MAPE-K framework [11], [25]. We consider the regression model that we learn via transfer learning in this work as the Knowledge component of the MAPE-K that acts as an interface to which other components can query the performance. For deciding how many observations and from what source to transfer, we use the cost model that we have introduced earlier. At runtime, the managed system is Monitored by pulling the end-to-end performance metrics (*e.g.*, latency, throughput) from the corresponding sensors. Then, the retrieved performance data needs to be Analysed. Next, the model needs to be updated taking into account the new performance observations. Having updated the model, a new configuration may be Planned to replace the current

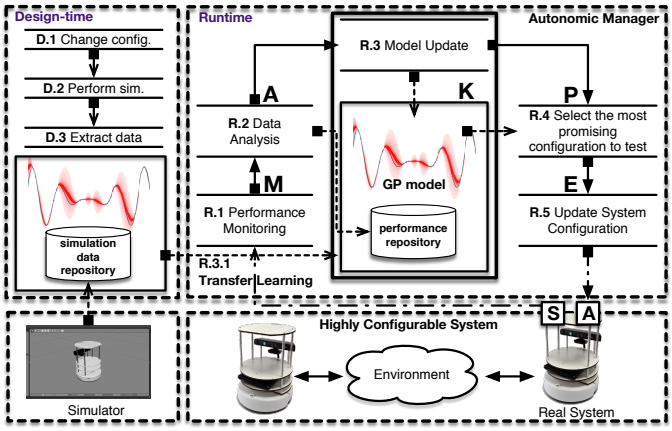


Fig. 4: Integrating transfer learning with MAPE-K loop, where knowledge update is realized with transfer learning.

configuration. Finally, the new configuration will be enacted by Executing platform specific operations. This enables model-based knowledge evolution using machine learning [22].

The underlying model can be updated not only when a new observation is available but also by transferring the learning from other related sources. So at each adaptation cycle, we can update our belief about the correct response given data from the managed system and other related sources. This can be particularly useful when the adaptive systems need to make a decision when the internal knowledge utilized in the MAPE-K loop is not accurate enough, *e.g.*, in early stages of learning when the system needs to react to environmental changes the internal knowledge is not accurate and as a result the adaptation decisions are sub-optimal [23]. In this situation, even the measurements from a noisy source (*e.g.*, simulator) could be beneficial in order to boost the initial performance achievable using only the transferred knowledge, before any further learning is done. Also, another challenge that has been reported in the past is the slow convergent of the learning process [23], in this situation, transfer learning can accelerate the learning process in the target environment given the transferred knowledge compared to the amount of time to learn it from scratch.

IV. Experimental results

We evaluate the *effectiveness* and *applicability* of our transfer learning approach for learning models for highly-configurable systems, in particular, compared to conventional non-transfer learning. Specifically, we aim to answer the following three research questions:

***RQ1**: How much does transfer learning improve the prediction accuracy?*

***RQ2**: What are the trade-offs between learning with different numbers of samples from source and target in terms of prediction accuracy?*

***RQ3**: Is our transfer learning approach applicable in the context of self-adaptive systems in terms of model training and evaluation time?*

We will proceed as follows. First, we will return to our motivating example of an autonomous service robot to explore the benefits and limitations of transfer learning for this single case. Subsequently, we explore the research questions quantitatively, performing experiments on 5 different configurable systems. We have implemented our cost-aware transfer learning approach in Matlab 2016b and, depending on each experiment, we have developed different scripts for data collections and automated configuration changes. The source code and data are available in an online appendix [1].

A. Case study: Robotics software

We start by demonstrating our approach with a small case study of service robots, which we already motivated in Section II-A. Our long-term goal is to allow the service robots to adapt effectively to (possibly unanticipated) changes in the environment, in its goals, or in other parts of the system, among others, by reconfiguring parameters. Specifically, we work with the CoBot robotic platform [39].

To enable a more focused analysis and presentation, we focus on a single but important subsystem and two parameters: The particle-filter-based autonomous localization component of the CoBot software [7] determines the current location of the robot, which is essential for navigation. Among the many parameters of the component, we analyze two that strongly influence the accuracy of the localization and required computation effort, (1) the number of particle estimates and (2) the number of gradient descent refinement steps applied to each of these particles when we receive a new update from sensors. At runtime, the robot could decide to reconfigure its localization component to trade off location accuracy with energy consumption, based on the model we learn.

We have extensively measured the performance of the localization component with different parameters and with different simulated environmental conditions in prior work [24]. For the two parameters of interest (number of particles and refinements), we used 25 and 27 different values each, evenly distributed on the log scale within their ranges ($[5-10,000]$ and $[1-10,000]$), *i.e.*, $|\mathbb{X}| = 25 \times 27 = 675$. Specifically, we have repeatedly executed a specific mission to navigate along a corridor in the simulator and measured performance in terms of CPU usage (a proxy for energy consumption), time to completion, and location accuracy. Each measurement takes about 30 seconds. In Figure 5a, we illustrate how CPU usage depends on those parameters (the white area represents configurations in which the mission fails to complete).

As a transfer scenario, we consider simulation results given different environment conditions. As a target, we consider measurements that we have performed in the default configuration. As a source, we consider more challenging environment conditions, in which the odometry sensor of the robot is less reliable (*e.g.*, due to an unfamiliar or slippery surface); we simulate that the odometry sensor is both miscalibrated ($30\,\%$) and noisy ($\pm 45\,\%$). We assume that we have collected these measurements in the past or offline in a simulator and that those measurements were therefore relatively cheap. As shown in Figure 5b, localization becomes more computationally expensive in the target environment. Our goal is to use (already available) data from the noisy environment for learning predictions in the non-noisy target environment with only a few measurements in that environment.

Observations: For both source and target environments, we can learn fairly accurate models if we take large numbers of samples (say $|\mathcal{D}| > 80\,\% \times |\mathbb{X}|$) in that environment, but predictions are poor when we can sample only a few configurations (say $|\mathcal{D}| < 1\,\% \times |\mathbb{X}|$). That is, we need at least a moderate number of measurements to produce useful models for making self-adaptation decisions at runtime. Also, using the model learned exclusively from measurements in the source environment to predict the performance of the robot in the more noisy environment leads to a significant prediction error throughout the entire configuration space (cf. Figure 5c). However, transfer learning can effectively calibrate a model learned from cheap measurements in the source environment to the target environment with only a few additional measurements from that environment. For example, Figure 5d shows the predictions provided by a model that has been learned by transfer learning using 18 samples from the source and only 4 samples from the target environment. As a result, the model learned the structure of the response properly in terms of changing of values over the configuration space.

In Figure 6a, we illustrate how well we can predict the target environment with a different number of measurements in the source and the target environment. More specifically, we randomly sampled configurations from both source and target between $[0 - 100\%]$ and $[0 - 10\%]$ respectively and we measured the accuracy as the relative error comparing predicted to actual performance in all 675 configurations. Note $|\mathcal{D}_s| = 0$ corresponds to model learning without transfer learning. We can see that, in this case, predictions can be similarly accurate when we replace expensive measurements from the target environment by more, but cheaper samples. Moreover, we have noticed that the models learned without transfer learning are highly sensitive to the sample selection (cf. Figure 6b). The variance becomes smaller when we increase the number of source samples (along with the y-axis) leading to more reliable models. In this case study, all model learning and evaluation times (both for normal learning and transfer learning) are negligible ($< 5s$), cf. Figure 6c,d.

Relatedness: In this case study, we can also observe that the relatedness between the source and target matters by changing the environments more or less aggressively. For that purpose, we simulated six alternative source environments by adding noise with different power levels to the source we considered previously (*i.e.*, odometry miscalibration of 30 % and noise of $\pm 45\,\%$, cf., Figure 7). Typically, the stronger the target environment is changed from the source (default) environment, the larger the difference in performance is observed for the same configuration between the two environments. This relationship is visible in correlation measures in Figure 7). The prediction error reported in Figure 7 indicates that transfer learning becomes more effective (*i.e.*, produces accurate models with a few samples from the target environment) if measurements in the source and target are strongly correlated. This also demonstrates that even transfer from a response with a small correlation helps to learn a better model comparing with no transfer learning.

Based on this observation, we need to decide (i) from which alternative source and (ii) how many samples we transfer to derive an accurate model. In general, we can: either (a) select

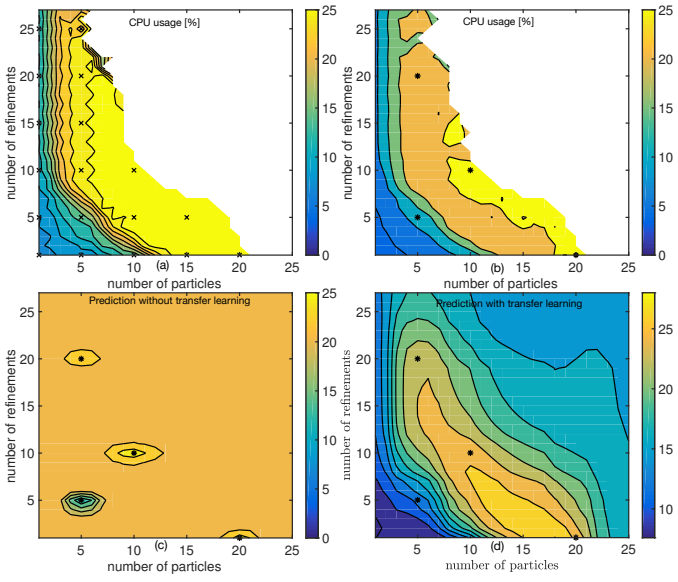


Fig. 5: Illustration of transfer learning: (a) a source response function from which we can take samples; (b) a target response function which we are interested to learn; (c) a prediction without transfer learning; (d) a prediction with transfer learning.

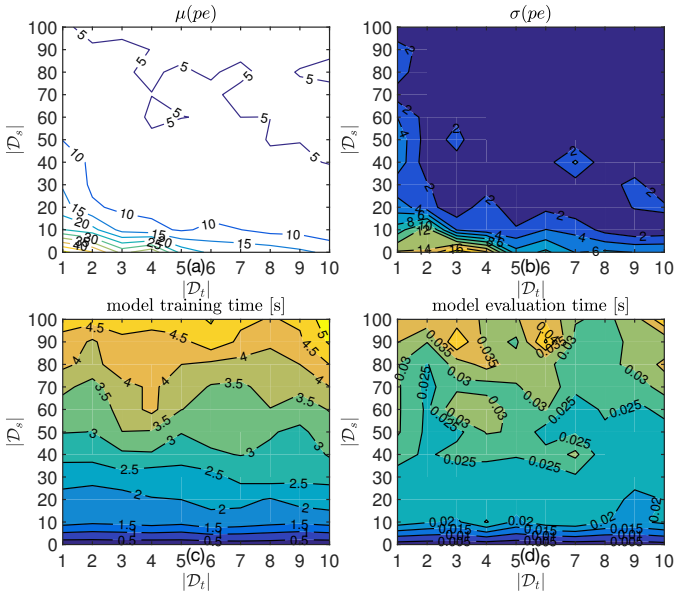
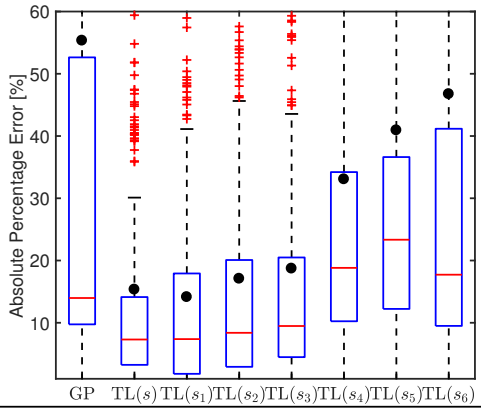


Fig. 6: (a) Prediction error of the trained model, (b) variance of predictions (transfer learning contributes to lower the uncertainty regarding model predictions, therefore, more reliable learning), (c) training and (d) evaluation overhead.

the most related source and sample only from that source, or (b) select fewer samples from unrelated sources and more from the more related ones to transfer. Note that our transfer learning method supports both strategies. For this purpose, we use the concept of relatedness (correlation) between source and target. The former is simpler, but we need to know, a priori, the level of relatedness of different sources. The latter is more sophisticated using a probabilistic interpretation of relatedness to learn from different sources using a concept of distance from the target. Our method supports this through the



Sources	s	s_1	s_2	s_3	s_4	s_5	s_6
noise-level	0	5	10	15	20	25	30
corr. coeff.	0.98	0.95	0.89	0.75	0.54	0.34	0.19
$\mu(pe)$	15.34	14.14	17.09	18.71	33.06	40.93	46.75

Fig. 7: Prediction accuracy of the model learned with samples from different sources of different relatedness to the target. GP is the model without transfer learning.

kernel function in Eq. 8 that embeds the knowledge of source correlations and exploits that knowledge in the model learning process. The selection of a specific strategy depends on several factors including: (i) the cost of samples from each specific source (samples from some sources may come for free, *i.e.* $c_s = 0$ and we may want to use more samples from this source despite the fact that it may be less relevant), (ii) the bound on runtime overhead (more samples leads to a higher learning overhead), and (iii) domain for which the model is used (some domain are critical and some adversaries may manipulate the input data exploiting specific vulnerabilities of model learning to compromise the whole system security [33]). We leave the investigation of strategy selection as a future work.

B. Experiment: Prediction accuracy and model reliability

With our case study, we demonstrated our approach on a specific small example. To increase both internal and external validity, we perform a more systematic exploration on multiple different configurable systems.

Subject systems: First, we again use the localization component of the CoBot system, but (since we are no longer constrained by plotting results in two dimensions) explore a larger space with 4 parameters. In addition, we selected highly-configurable systems as subjects that have been explored for configuration optimization in prior work [21]: three stream processing applications on Apache Storm (WordCount, RollingSort, SOL) and a NoSQL database benchmark system on Apache Cassandra. WordCount is a popular benchmark [16] featuring a three-layer architecture that counts the number of words in the incoming stream and it is essentially a CPU intensive application. RollingSort is a memory intensive system that performs rolling counts of incoming messages for identifying trending topics. SOL is a network intensive system, where the incoming messages will be routed through a multi-layer network. From the practical perspective, these benchmark applications are based on popular big data engines (*e.g.*, Storm) and understanding the performance influence of configuration parameters can have a large impact on the maintenance of modern systems that are based on these highly-configurable engines. The Cassandra measurements was done using scripts that runs YCSB [10] and was originally developed for a prior work [3].

The notion of source and target set depends on the subject system. In CoBot, we again simulate the same navigation mission in the default environment (source) and in a more difficult noisy environment (target). For the three stream processing applications, source and target represent different workloads, such that we transfer measurements from one workload for learning a model for another workload. More specifically, we control the workload using the maximum number of messages which we allow to enter the stream processing architecture. For the NoSQL application, we analyze two different transfers: First, we use as a source a query on a database with 10 million records and as target the same query on a database with 20 million records, representing a more expensive environment to sample from. Second, we use as a source a query on 20 million records on one cluster and as target a query on the same dataset run on a different cluster, representing hardware changes. Overall, our subjects cover different kinds of applications and different kinds of *transfer scenarios* (changes in the environment, changes in the workload, changes in the dataset, and changes in the hardware).

Experimental setup: As independent variables, we systematically vary the size of the learning sets from both source and target environment in each subject system. We sample between 0 and 100 % of all configurations in the source environment and between 1 and 10 % in the target.

As a dependent variable, we measure learning time and prediction accuracy of the learned model. For each subject system, we measure a large number of random configurations as the evaluation set, independently from configurations sampled for learning, and compare the predictions of the learned model $\hat{f}$ to the actual measurements of the configurations in the evaluation set $\mathcal{D}_o$. We compute the *absolute percentage error (APE)* for each configuration x in the evaluation set $\frac{|\hat{f}(\boldsymbol{x})-f(\boldsymbol{x})|}{f(\boldsymbol{x})} \times 100$ and report the average to characterize the accuracy of the prediction model. Ideally, we would use the whole configuration space as evaluation set ($\mathcal{D}_o = \mathbb{X}$), but the measurement effort would be prohibitively high for most real-world systems [21], [36]; hence we use large random samples (cf. size column in Table I).

The measured and predicted metric depends on the subject system: For the CoBot system, we measure average CPU usage during the same mission of navigating along a corridor as in our case study; we use the average of three simulation runs for each configuration. For the Storm and NoSQL experiments, we measure average latency over a window of 8 and 10 minutes respectively. Also, after each sample collection, the experimental testbed was cleaned which required several minutes for the Storm measurements and around 1 hour (for offloading and cleaning the database) for the Cassandra measurements. We sample a given number of configurations in the source and target randomly and report average results and standard deviations of accuracy and learning time over 3 repetitions.

TABLE I: Overview of our experimental datasets. "Size" column indicates the number of measurements in the datasets and "Testbed" refer to the infrastructure where the measurements are taken and their details are in the appendix.

	Dataset	Parameters	Size	Testbed
1	CoBot(4D)	1-odom_miscalibration, 2-odom_noise, 3-num_particles, 4-num_refinement	56585	C9
2	wc(6D)	1-spouts, 2-max_spout, 3-spout_wait, 4-splitters, 5-counters, 6-netty_min_wait	2880	C1
3	sol(6D)	1-spouts, 2-max_spout, 3-top_level, 4-netty_min_wait, 5-message_size, 6-bolts	2866	C2
4	rs(6D)	1-spouts, 2-max_spout, 3-sorters, 4-emit_freq, 5-chunk_size, 6-message_size	3840	C3
5	cass-10	1-trickle_fsync, 2-auto_snapshot, 3-con._reads, 4-con._writes 5-file_cache_size_in_MB 6-con._compactors	1024	C6x,C6y
6	cass-20			

Results: We show results of our experiments in Figure 8. The 2D plot shows average errors across all subject systems. The results in which the set of source samples $\mathcal{D}_s$ is empty represents the baseline case without transfer learning. Although the results differ significantly among subject systems (not surprising, given different relatedness of the source and target) the overall trends are consistent.

First, our results show that transfer learning can achieve high prediction accuracy with only a few samples from the target environment; that is, transfer learning is clearly beneficial if samples from the source environment are much cheaper than samples from the target environment. Given the same number of target samples, the prediction accuracy becomes better up to even 3 levels of magnitudes as we include more samples from the source.

Second, using transfer learning with larger samples from the source environment reduces the variance of the prediction error. That is, models learned with only a few samples from the target environment are more affected by the randomness in sampling from the configuration space. Conversely, transfer learning improves the quality and reliability of the learned models, by reducing sensitivity to the sample selection.

Third, the model training time increases monotonically when we increase the sample size for target or source (cf. Figure 6c for the CoBot case study and for the other subject systems in the appendix). At the same time, even for large samples, it rarely exceeds 100 seconds for our subject systems, which is reasonable for online use in a self-adaptive system. Learning time is negligible compared to measurement times in all our subject systems.

Finally, the time for evaluating the models (*i.e.*, using the model to make a prediction for a given configuration), does not change much with models learned from different sample sizes (cf. Figure 6d) and is negligible overall (< 300ms).

Summary: we see improved accuracy and reliability when using transfer learning with additional source samples (RQ1); we see clear trade-offs among using more source or target samples (RQ2); and we see that training and evaluation times are acceptable to be applied for self-adaptive systems (RQ3).

C. Discussion: Trade-offs and cost models

Our experimental results clearly show that we can achieve accurate models both with a larger number of samples from source or target environment or both: In Figure 8, we can see how many models for different numbers of source and target samples yield models with similar accuracy. Those different points with the same accuracy can be interpreted as indifference curves (a common tool in economics [6]). We can invest different measurement costs to achieve models with equivalent utility. In Figure 8a, we show those indifference curves more explicitly for the CoBot system. Using the indifference curves enables us to decide how many samples from either a source or a target we should use for the model learning process that is most beneficial in increasing predictive accuracy over unseen regions of the configuration space while satisfying the budget constraint. Given concrete costs and budgets, say, samples from the target environment are 3 times more expensive to gather than samples from the source environment, we plotted an additional line representing our budget and find the combination of source and target samples that produces the best prediction model for our budget (see intersection of the budget line with the highest indifference curve in Figure 8). Although we will not know the indifference lines until we run extensive experiments, we expect that the general trade-offs are similar across many subject systems and transfer scenarios and that such a cost model (cf. Eq. 2) can help to justify decisions on how many samples to take from each environment. We leave a more detailed treatment to future work.

In this context, we can fix a cost model (*i.e.*, fixing specific values for c_s, c_t) and transform the indifference diagrams into a multi-objective goal and derive the Pareto front solutions as shown in Figure 9. This helps us to locate the feasible Pareto front solutions within the sweet spot.

D. Threats to validity and limitations

1) Internal validity: In order to ensure internal validity, we repeated the execution of the benchmark systems and measure performance for a large number of configurations. For doing so, we have invested several months for gathering the measurements, which resulted in a substantial dataset. Moreover, we used standard benchmarks so that we are confident in that we have measured a realistic scenario.

2) External validity: In order to ensure external validity, we use three classes of systems in our experiments including: (i) a robotic system, (ii) 3 different stream processing applications, and (iii) a NoSQL database system. These systems have different numbers of configuration parameters and are from different application domains.

3) Limitations: Our learning approach relies on several assumptions. First, we assume that the target response function is smooth. If a configuration parameter has an unsteady performance behavior, we cannot learn a reliable model, but only approximate its performance close to the observations. Furthermore, we assume that the source and target responses are related. That is they are correlated to a certain extent. The more related, the faster and better we can learn. Also, we need the configurable system to have a deterministic performance behavior. If we replicate the performance measurements for the same system, the observed performance should be similar.

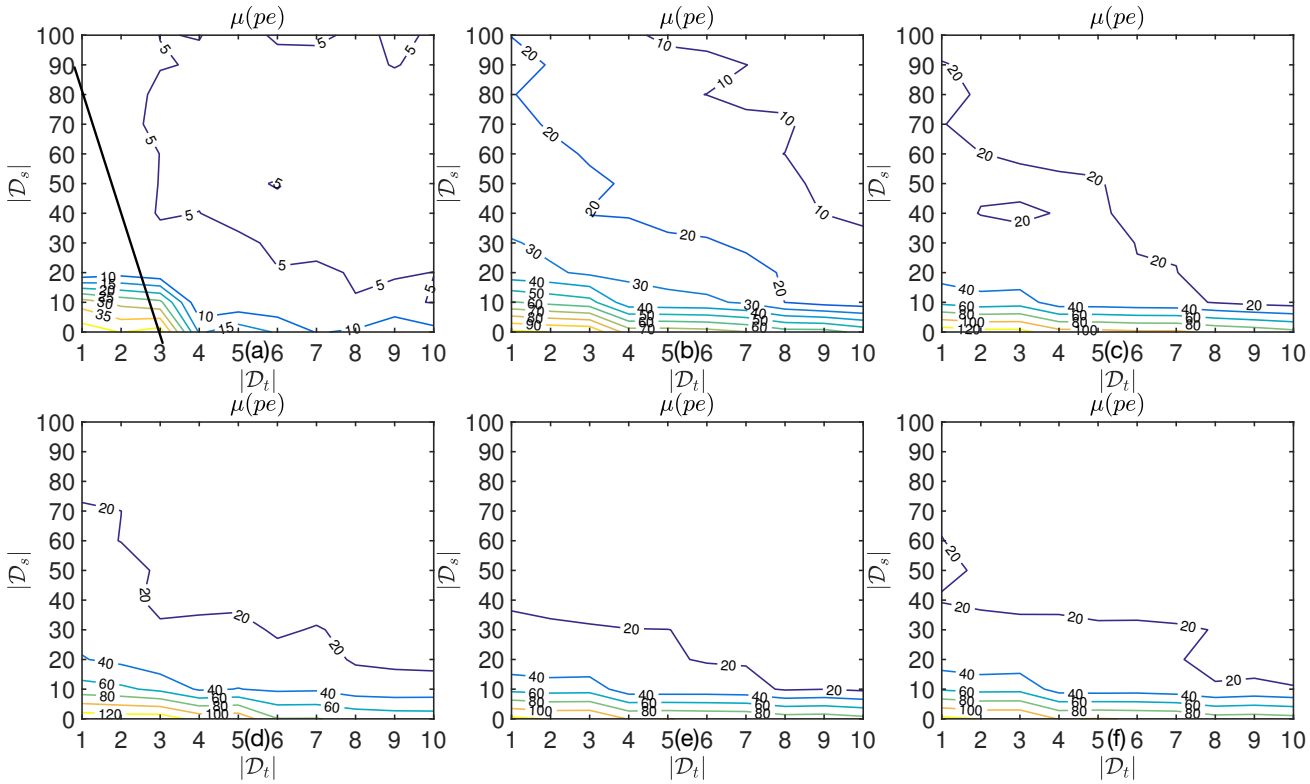


Fig. 8: Prediction accuracy for (a) CoBot, (b) WC, (c) SOL, (d) RS, (e) cass (hardware change), (f) cass (DB size change).

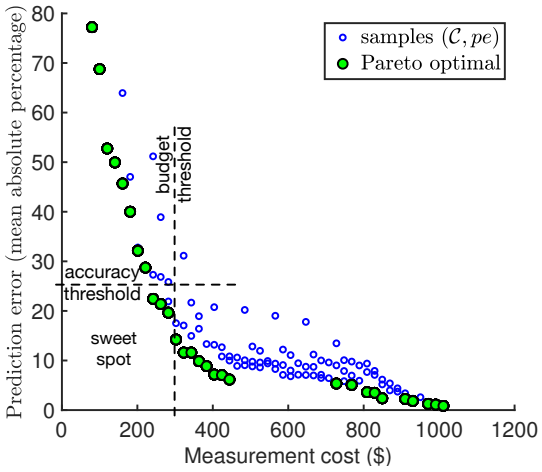


Fig. 9: A two-objective optimization goal. We are interested in the solutions in the targeted sweet spot.

V. Conclusions

Today most software systems are configurable and performance reasoning is typically used to adapt the configuration in order to respond to environmental changes. Machine learning and sampling techniques have been previously proposed to build models in order to predict the performance of unseen configurations. However, the models are either expensive to learn, or they become extremely unreliable if trained on sparse samples. Our cost-aware transfer learning method is orthogonal to the both previously proposed directions promoting to learn from other cheaper sources. Our approach requires only very few samples from the target response function and can learn an accurate and reliable model based on sampling from other relevant sources. We have done extensive experiments with 5 highly configurable systems demonstrating that our approach (i) improves the model accuracy up to several orders of magnitude, (ii) is able to trade-off between different number of samples from source and target, and (iii) imposes an acceptable model building and evaluation cost making appropriate for application in the self-adaptive community.

Future directions. Beside performance reasoning, our cost-aware transfer learning can contribute to reason about other non-functional properties and quality attributes, *e.g.*, energy consumption. Also, our approach could be extended to support configuration optimization. For example, in our previous work, we have used GP models with Bayesian optimization to focus on interesting zones of the response functions to find optimum configuration quickly for big data systems [21]. In general, our notion of cost-aware transfer learning is independent of a particular black-box model and complementary to white-box approaches in performance modeling such as queuing theory. A more intelligent way (*e.g.*, active learning) of sampling the source and the target environment to gain more information is also another fruitful future avenue.

Acknowledgment

This work has been supported by AFRL and DARPA (FA8750-16-2-0042). Kaestner's work is also supported by NSF awards 1318808 and 1552944 and the Science of Security Lablet (H9823014C0140). Siegmund's work is supported by the DFG under the contract SI 2171/2.

REFERENCES

[1] Online appendix: https://github.com/pooyanjamshidi/transferlearning.

[2] I. D. P. Anaya, V. Simko, J. Bourcier, N. Plouzeau, and J.-M. Jézéquel. A prediction-driven adaptation approach for self-adaptive sensor networks. In *Proceedings of the 9th International Symposium on Software Engineering for Adaptive and Self-Managing Systems (SEAMS)*, pages 145–154. ACM, 2014.

[3] M. Artač, editor. *Deliverable 5.2: DICE delivery tools-Intermediate version*. 2017. http://www.dice-h2020.eu/.

[4] S. Balsamo, A. Di Marco, P. Inverardi, and M. Simeoni. Model-based performance prediction in software development: A survey. *IEEE Transactions on Software Engineering (TSE)*, 30(5):295–310, 2004.

[5] S. Becker, L. Grunske, R. Mirandola, and S. Overhage. Performance prediction of component-based systems. In *Architecting Systems with Trustworthy Components*, pages 169–192. Springer, 2006.

[6] B. R. Binger and E. Hoffman. *Microeconomics with calculus*. W. W. Norton and Company, 1988.

[7] J. Biswas and M. M. Veloso. Localization and navigation of the CoBots over long-term deployments. *The International Journal of Robotics Research (IJRR)*, 32(14):1679–1694, 2013.

[8] 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.

[9] H. Chen, W. Zhang, and G. Jiang. Experience transfer for the configuration tuning in large scale computing systems. *SIGMETRICS*, 37(2):51–52, 2009.

[10] B. F. Cooper, A. Silberstein, E. Tam, R. Ramakrishnan, and R. Sears. Benchmarking cloud serving systems with YCSB. In *Proc. of Symposium on Cloud Computing (SOCC)*. ACM, 2010.

[11] R. de Lemos, H. Giese, H. A. Müller, M. Shaw, J. Andersson, M. Litoiu, B. Schmerl, G. Tamura, N. M. Villegas, T. Vogel, D. Weyns, L. Baresi, B. Becker, N. Bencomo, Y. Brun, B. Cukic, R. Desmarais, S. Dustdar, G. Engels, K. Geihs, K. M. Göschka, A. Gorla, V. Grassi, P. Inverardi, G. Karsai, J. Kramer, A. Lopes, J. Magee, S. Malek, S. Mankovskii, R. Mirandola, J. Mylopoulos, O. Nierstrasz, M. Pezzè, C. Prehofer, W. Schäfer, R. Schlichting, D. B. Smith, J. P. Sousa, L. Tahvildari, K. Wong, and J. Wuttke. Software engineering for self-adaptive systems: A second research roadmap. In *Software Engineering for Self-Adaptive Systems II*, pages 1–32. Springer, 2013.

[12] J. Ehlers, A. van Hoorn, J. Waller, and W. Hasselbring. Self-adaptive software system monitoring for performance anomaly localization. In *Proceedings of the 8th ACM International Conference on Autonomic Computing (ICAC)*, pages 197–200. ACM, 2011.

[13] A. Elkhodary, N. Esfahani, and S. Malek. Fusion: A framework for engineering self-tuning self-adaptive software systems. In *Proceedings of the eighteenth ACM SIGSOFT International Symposium on Foundations of Software Engineering (FSE)*, pages 7–16. ACM, 2010.

[14] N. Esfahani, A. Elkhodary, and S. Malek. A learning-based framework for engineering feature-oriented self-adaptive software systems. *IEEE Transactions on Software Engineering (TSE)*, 39(11):1467–1493, 2013.

[15] A. Filieri, H. Hoffmann, and M. Maggio. Automated multi-objective control for self-adaptive software design. In *10th Joint Meeting on Foundations of Software Engineering (FSE)*, pages 13–24. ACM, 2015.

[16] A. Ghazal, T. Rabl, M. Hu, F. Raab, M. Poess, A. Crolotte, and H.-A. Jacobsen. BigBench: Towards an industry standard benchmark for big data analytics. In *Proceedings of the 2013 ACM SIGMOD International Conference on Management of Data*, pages 1197–1208. ACM, 2013.

[17] H. Gomaa and M. Hussein. Model-based software design and adaptation. In *Proceedings of the 2007 International Workshop on Software Engineering for Adaptive and Self-Managing Systems (SEAMS)*, page 7. IEEE Computer Society, 2007.

[18] J. Guo, K. Czarnecki, S. Apel, N. Siegmund, and A. Wasowski. Variability-aware performance prediction: A statistical learning approach. In *Proceedings of the IEEE/ACM 28th International Conference on Automated Software Engineering (ASE)*, pages 301–311, 2013.

[19] J. Happe, H. Koziolek, and R. Reussner. Facilitating performance predictions using software components. *IEEE Software*, 28(3):27, 2011.

[20] N. Huber, F. Brosig, S. Spinner, S. Kounev, and M. Bahr. Model-based self-aware performance and resource management using the Descartes modeling language. *IEEE Transactions on Software Engineering*, 2017.

[21] P. Jamshidi and G. Casale. An uncertainty-aware approach to optimal configuration of stream processing systems. In *IEEE 24th International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems (MASCOTS)*, pages 39–48, 2016.

[22] P. Jamshidi, C. Pahl, and N. C. Mendonça. Managing uncertainty in autonomic cloud elasticity controllers. *IEEE Cloud Computing*, 3(3):50–60, 2016.

[23] P. Jamshidi, A. Sharifloo, C. Pahl, H. Arabnejad, A. Metzger, and G. Estrada. Fuzzy self-learning controllers for elasticity management in dynamic cloud architectures. In *12th International ACM SIGSOFT Conference on Quality of Software Architectures (QoSA)*, pages 70–79. IEEE, 2016.

[24] P. Kawthekar and C. Kästner. Sensitivity analysis for building evolving & adaptive robotic software. In *IJCAI Workshop on Autonomous Mobile Service Robots*, 2016.

[25] J. O. Kephart and D. M. Chess. The vision of autonomic computing. *Computer*, 36(1):41–50, 2003.

[26] E. Kocaguneli, T. Menzies, and E. Mendes. Transfer learning in effort estimation. *Empirical Software Engineering*, 20(3):813–843, 2015.

[27] R. Krishna, T. Menzies, and W. Fu. Too much automation? The bellwether effect and its implications for transfer learning. In *Proceedings of the 31st IEEE/ACM International Conference on Automated Software Engineering (ASE)*, pages 122–131. ACM, 2016.

[28] D. C. Montgomery. *Design and analysis of experiments*. John Wiley & Sons, 2008.

[29] J. Nam and S. Kim. Heterogeneous defect prediction. In *Proceedings of the Joint Meeting on Foundations of Software Engineering (FSE)*, pages 508–519. ACM, 2015.

[30] J. Nam, S. J. Pan, and S. Kim. Transfer defect learning. In *Proceedings of the International Conference on Software Engineering (ICSE)*, pages 382–391. IEEE Press, 2013.

[31] T. Osogami and S. Kato. Optimizing system configurations quickly by guessing at the performance. In *Proceedings of the 2007 ACM SIGMETRICS International Conference on Measurement and Modeling of Computer Systems*, volume 35, 2007.

[32] S. J. Pan and Q. Yang. A survey on transfer learning. *IEEE Transactions on Knowledge and Data Engineering*, 22(10):1345–1359, 2010.

[33] N. Papernot, P. McDaniel, A. Sinha, and M. Wellman. Towards the science of security and privacy in machine learning. *arXiv preprint arXiv:1611.03814*, 2016.

[34] M. Reckhaus, N. Hochgeschwender, J. Paulus, A. Shakhimardanov, and G. K. Kraetzschmar. An overview about simulation and emulation in robotics. *IEEE International Conference on Simulation, Modeling, and Programming for Autonomous Robots*, pages 365–374, 2010.

[35] A. Sarkar, J. Guo, N. Siegmund, S. Apel, and K. Czarnecki. Cost-efficient sampling for performance prediction of configurable systems. In *30th IEEE/ACM International Conference on Automated Software Engineering (ASE)*, pages 342–352. IEEE, 2015.

[36] N. Siegmund, A. Grebhahn, S. Apel, and C. Kästner. Performance-influence models for highly configurable systems. In *Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering (FSE)*, pages 284–294. ACM, 2015.

[37] S. Thrun, W. Burgard, and D. Fox. *Probabilistic robotics*. MIT press, 2005.

[38] L. Torrey and J. Shavlik. Transfer learning. *Handbook of Research on Machine Learning Applications and Trends: Algorithms, Methods, and Techniques*, 1:242–264, 2009.

[39] M. Veloso, J. Biswas, B. Coltin, and S. Rosenthal. CoBots: Robust symbiotic autonomous mobile service robots. In *International Joint Conference on Artificial Intelligence (IJCAI)*, volume 4423, pages 4423–4429, 2015.

[40] C. K. Williams and C. E. Rasmussen. Gaussian processes for machine learning. *MIT Press*, 2006.

[41] 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.

[42] 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.

[43] N. Yigitbasi, T. L. Willke, G. Liao, and D. Epema. Towards machine learning-based auto-tuning of MapReduce. In *IEEE International Symposium on Modelling, Analysis and Simulation of Computer and Telecommunication Systems (MASCOTS)*, 2013.

[44] Y. Zhang, J. Guo, E. Blais, and K. Czarnecki. Performance prediction of configurable software systems by Fourier learning. In *International Conference on Automated Software Engineering*, pages 365–373, 2015.

[45] W. Zheng, R. Bianchini, and T. D. Nguyen. Automatic configuration of internet services. *ACM SIGOPS Operating Systems Review*, 41(3):219, 2007.