Scalable Performance Models for Highly Configurable Systems

Jeho Oh
Department of Computer Science
Austin, Texas, USA
jeho@cs.utexas.edu

Oleg Zilmano
Department of Computer Science
Austin, Texas, USA
ozilman@cs.utexas.edu

1 INTRODUCTION

Highly Configurable Systems (HCSs) are systems with parameters so that users can configure the functionality and running environment of the system. An active research topic in HCSs is learning an accurate performance model. A performance model aims to predict the performance of a configuration based on its parameters. A performance model is valuable to understand the properties of an HCS and find configurations that have optimal performance.

Prior work utilized different techniques including multivariate and [5–7, 13]. However, they were evaluated only on small HCSs (<54 parameters). Recently, Acher et al. [2] tried a variety of different machine learning techniques on a Linux kernel (>15000 parameters), which yielded lower accuracy than their predecessors' findings. Applicability of prior work on larger HCSs is unknown, while Acher et al. mentioned that some known machine learning approaches are not applicable for the scale of Linux.

We aim to evaluate state-of-the-art performance modelling techniques for larger HCSs (90 to 988 parameters). One thing we are particularly interested in is to use small training sets and see if we can achieve good results for cases where training data is limited, which as commonly pursued by prior work. In addition, we aim to analyze the results and gather an understanding on why some methods cannot scale and why some methods work better than others. Furthermore, we will try to improve the accuracy by applying different dimensionality reduction techniques, such as Principal Component Analysis (PCA).

So far, we have replicated two state-of-the-art approaches, *DECART* and *DeepPerf*. While we could observe similar accuracy for the small HCSs that they used for evaluation, both approaches showed much higher prediction error for larger HCSs. We plan to investigate the cause of this issue and find ways to mitigate the problem.

2 BACKGROUND

2.1 Highly Configurable Systems

HCSs define parameters which represent increments in system functionalities [**?**]. A user can customize an HCS by selecting the value of its parameters and produce an executable *configuration*. Real-world HCSs often have a large number of parameters. The size of an HCS *configuration space*, the set of all configurations, can exceed 10^{82}, which is the estimated number of atoms in the universe [15].

In HCS, most parameters are Boolean and have complex dependencies. These dependencies are usually known and specified to manage the valid selection of the user. Dependencies can make many parameter combinations invalid so that merely random selecting the value of individual features rarely yields a valid configuration [9]. For example, only 10^{-59}% of all possible combinations of parameters in the Fiasco micro-kernel [1] are legal [8].

Due to its diversity of selection, users may want to find a configuration with certain performance such as response time or memory usage. Predicting the performance of a configuration, however, is not trivial. Parameters often affect the performance in a non-linear manner, while understanding its effect requires analyzing the performance of different configurations as performance is a comprehensive property of a configuration [11]. However, configuration data are often unavailable and costly to collect. As an extreme case, a new power management system may take weeks to evaluate a configuration while there are no prior data available to utilize [10, 14]. This raises a need to learn a performance model of an HCS with practical effort.

2.2 Learning Performance Models for HCSs

A number of approaches were proposed to learn a performance model from HCSs. We describe here some state-of-the-art approaches.

DECART [5] uses *Classification and Regression Tree* (CART) method to learn a performance model. CART analyzes random samples and creates a decision tree based on parameters identified as significant to performance.Prediction involves traversing the tree based on parameter values of a given configuration. *DECART* improves the accuracy of the CART model with automated resampling and parameter tuning.

SPLConquerer [13] uses step-wise linear regression to assign performance attributes to features and their interactions. Instead of random sampling, *SPLConquerer* uses different sampling heuristics to select the configurations based on their feature values. Prediction adds up the attribute value of features selected by a given configuration.

PerLasso [7] formulate the performance model as a Boolean function and treat the parameters as Fourier coefficients to provide a bound on accuracy over number of samples. Then to learn the coefficients, they use LASSO regression.

DeepPerf [6] is a deep neural net framework designed to effectively model performance for HCSs. It uses a multi-layered deep feed forward neural net with L1 regularization on the weights of the first layer. *DeepPerf* includes a system for automatic parameter sweeping to optimize hyper-parameters λ, learning rate, and the number of layers of the net.

These approaches report that they can achieve small performance estimation error ($< 10\%$) with using under $5n$ samples in overall where n is the number of parameters in an HCS. Their evaluation, however, were all conducted on HCSs with small number of parameters where the largest HCS had only 54 parameters. Real-world HCSs often have much larger parameters but the scalability

of those methods are unknown yet. Thus, we plan to evaluate these approaches on larger HCSs and analyze the results for possible improvements.

2.3 Applicable Machine Learning Approaches

Aside from the methods mention in the section above, a wide array of machine learning techniques is mentioned in the literature on performance modeling of configurable systems, with varying degrees of success. Linear regression, polynomial regression, classification and regression trees (CART), neural networks, random forests, and support vector machines (SVM) are among the prominent methods being successfully applied, as presented in [12].

When looking at machine learning approaches for configurable systems, there are two main considerations: predictive power, and interpretability of results . Interpretable results means that the outputs of the machine learning algorithm can provide a human operator useful information about the relationship between the parameters and the performance, such that she could make better informed decisions when selecting configuration options. In other words, information on which parameters, or combinations of thereof, are likely to influence the performance, and in what way. Interpretability and prediction power are two distinct goals, and different techniques are better suited for each one. If interpretability is of concern, then two of the strongest function approximation techniques, neural networks and SVMs, are not suitable, as very little could be understood from looking on weights of these models. In this case, some of the successful approaches include Lasso and CART. If we are concerned with predictive power alone, there are no restriction on the techniques that can be used.

2.3.1 Regularization. One technique that has been popular in a few recent works is using L1 regularization [6]. As described by Ha et al. [7], in configurable software systems the Fourier coefficients of the performance as function of the configuration parameters are presumed to be highly sparse. This implies that only a subset of the configuration parameters has significant impact on the performance. Based on that assumption of sparsity of configurable systems, that is supported by empirical results in several papers [4, 14], adding L1 regularization should help reach better prediction accuracy. The authors of the *DeepPerf* paper reported improving performance of a deep neural network model by introducing L1 regularization, achieving better results then any previous methods on smaller HCSs.

2.3.2 Feature Selection. For highly configurable software systems, due to the large number of inputs feature selection may play an important role in successful application of machine learning techniques [2]. Features selection be done manually based on prior knowledge of the system of which configuration parameters are likely to be influential [2, 12]. A different way to reduce the number of features is by removing features that have low variance, based on the assumption that. Another important is removing highly correlated set of features, for example if features X_1 and X_2 have a correlation of 1, then they are in essence, one feature and could be removed from the set. A step up from this is using PCA to provide a reduced set of features capturing the most important correlation information, however this has not been widely discussed in the literature on configurable systems, to the best of our knowledge.

3 IMPLEMENTATION

For the implementation of the *DeepPerf* we used the original code of the authors of the papers, available online.[1]. DeepPerf has a system to automatically tune the hyper-parameters. For the implementation of *DECART* we also used the original code from the authors [2]. *DECART* is based on CART model, but automatically search for the optimal hyperparameters for a given HCS. The hyperparameter was sampled by grid search and evaluated by 10-fold cross validation which are suggested as the default by the authors.

In addition, we did our own implementation of feed forward neural net framework in PyTorch, based on the *DeepPerf* tool (The original was implemented in Tensorflow). We used python 3.6 and PyTorch 1.4.0+cpu [reference]. We made some modifications to the neural architecture, [3]. Our neural net topology is similar to the that of *DeepPerf*, however in our case we set the number of layers to be eight, and instead, we fluctuate the number of nodes in each layer to modify the model's capacity. We do not scale the inputs as they are all zeros or ones, but since the performance value ranges value drastically between different systems we normalize the outputs to be between 0 and 100. For out optimizer we used the Adam optimizer with the default recommended values, and we initialized the weights using Xavier initialization method [3]. At this point we are still not using cross-validation of any kind, are still working out the architecture.

4 EVALUATION

4.1 Research Questions

We defined following research question to evaluate existing approaches.

- RQ1: Do prior work scale for larger HCSs?

We also defined additional research questions to analyze and improve over the prior work.

- RQ2: Can we find a better neural architecture for performance models?
- RQ3: Can we improve the accuracy of the model by dimensionality reduction?

In this report, we show the data we have collected so far for RQ1 and RQ2.

4.2 Experiment setting

We replicated state-of-the-art performance modeling approaches to evaluate their scalability and compare with our approach. For the state-of-the-art approaches, we used *DECART*, *DeepPerf*, and *PerLasso* which are described in Section 2.2. Additionally, we ran our proposed neural network to see how it fares against the recent state of the art methods.

[1] https://github.com/DeepPerf/DeepPerf

[2] https://github.com/jmguo/DECART

[3] Work in progress - this serves the base into our own ML method for performance prediction for HCS. We reserve the possibility of using different tools aside from deep learning,such as SVM, PCA techniques ets, to improve the results. TBD at this point

For the evaluation, we replicated the experiment setting that was commonly used by prior work [5–7, 13]. For each HCS, the performance model was trained with n, $3n$, and $5n$ samples to see the effect of sample size, where n is the number of parameters in the HCS. Once the model is trained, we used n additional samples to derive the prediction accuracy.

For the prediction accuracy, we measured the mean relative error (MRE), also called mean absolute percentage error (MAPE), which is defined as follows:

$$MRE[\%] = \frac{1}{|C|}\sum_{i=0}^{|C|}\frac{|\hat{y}-y|}{y}\cdot 100$$

|C| is the size of the data set, $\hat{y}$ is the predicted performance value, and y is the real performance value of the configuration sample.

We did 5 experiments to control the randomness and reported the mean value of these 5 results, together with the 95% confidence bounds.

4.3 Subject Systems

We first evaluated with six smaller HCSs which were commonly used in prior work as a sanity check:

- **LLVM** is a compiler infrastructure with 11 parameters. Test suite compilation times were measured.
- **x264** is a video encoder library for H.264/MPEG-4 AVC format with 16 parameters. Sintel trailer encoding times were measured.
- **BerkeleyDBC** is an embedded database system with 18 parameters. Benchmark response times were measured.
- **Dune** is a multi-gird solver with 14 parameters. Time to solve a Poisson's equation were measured.
- **HSMGP** is a stencil-grid solver with 17 parameters. Time to solve a Poisson's equation were measured.
- **HiPAcc** is a image procssing framework with 25 parameters. Time needed for solving a test set of partial differential equations were measured.

To evaluate the scalability of prior work, we used five HCSs that are based on the KConfig configuration tool [8]:

- **axTLS** is a server framework with 94 parameters.
- **Toybox** is a Linux command line utility with 316 parameters.
- **Fiasco** is a real time microkernel with 234 parameters.
- **Busybox** is a executable UNIX common utilities with 998 parameters.
- **uClibc-ng** is a library for embedded Linux with 269 parameters.

Performance models for these HCSs predicted the build size of the configurations.

4.4 Results

So far, we used *DECART* to learn the data from all HCSs except Dune, HSMGP, HiPAcc, and Busybox. For *DeepPerf*, we learn the data from larger HCSs. We did not have time to run our method on the smaller HCS therefore the empty column in table 1. Since our method is bound to change anyhow as it currently not offering a major improvement for larger HCSs, it is of not high importance at this point. We will add the data as we obtain the samples and train the model.

Table 1 shows the results for the smaller HCSs. The rows represent different HCSs and the columns represent different approaches. For each HCS and sample size, the lowest MRE is highlighted as bold. Note that, the data from *DeepPerf* is taken from their paper to show a comparison. We will replace them with our replicated data in the final report. As we have not collected the data from our approach yet, the entry for our method is left as to be announced (TBA).

Table 1: Average MRE and 95% confidence interval for smaller HCS

Subject System	Sample Size	*DECART*	*DeepPerf*	Our method
LLVM	n	5.38±0.52	**5.09±0.80**	TBA
	3n	4.1±0.31	**2.54±0.15**	TBA
	5n	2.43±0.13	**1.99±0.15**	TBA
BerkeleyDBC	n	150.16 ±153.23	**133.6±54.33**	TBA
	3n	**8.60 ±2.40**	13.1±3.39	TBA
	5n	6.47 ±3.88	**5.82±1.33**	TBA
X264	n	10.59 ±1.27	**10.43 ±2.28**	TBA
	3n	5.38 ±0.83	**2.31±0.31**	TBA
	5n	2.58 ±0.82	**0.87±0.11**	TBA

From Table 1, we observed:

- For all HCSs and approaches, MRE was below 10% for sample sizes $3n$ and $5n$.
- For all HCSs and approaches, both mean and variance of MRE decreased as sample size increases.
- *DeepPerf* showed smaller MRE compared to *DECART* with an exception of BerkeleyDBC using $3n$ samples.
- For *DECART*, the difference between our MRE and the MRE reported by the authors are less than 2.5%p.

Then, table 2 shows the results for the larger HCSs. Its format is the same as Table 1.

Table 2: Average MRE and 95% confidence interval for larger HCS.

Subject System	Sample Size	*DECART*	*DeepPerf*	Our method
axTLS	n	35.35±0.75	36.84±5.25	**32.49 ± 3.76**
	$3n$	33.02±1.00	31.77±2.03	**30.14 ± 1.39**
	$5n$	32.16 ±0.41	31.70±1.87	**29.94 ± 1.58**
Fiasco	n	39.06 ±1.19	**36.38±1.87**	44.79 ± 1.58
	$3n$	37.13 ±1.23	36.34±1.36	**34.25± 2.03**
	$5n$	36.011±0.49	37.74±1.69	**34.00 ± 0.94**
Toybox	n	10.39 ±1.49	**10.3 ±0.05**	12.93 ± 0.25
	$3n$	10.35 ±0.52	**10.35±0.12**	11.39 ± 0.29
	$5n$	**10.07 ±0.48**	10.35±0.37	10.62 ± 0.17
uClibc-ng	n	1871.02±174.20	1812.788±259.82	**1391.68 ± 121.41**
	$3n$	1942.22±74.33	1934.16±86.21	**1252.26± 144.82**
	$5n$	2012.97±62.69	1996.27±232.56	**1197.58 ± 119.24**

From Table 2, we observed:

- Both *DECART* and *DeepPerf* had much higher MRE compared to smaller HCSs. This is apparent for uClibc-ng, where the results for all samples sizes had MRE larger than 1250%.
- Larger sample size did not necessarily led to smaller MRE for *DECART* and *DeepPerf*. We confirmed that except for uClib-ng, the loss did not diverge.

- *DeepPerf* does not outperform *DECART* as much as smaller HCSs.
- Our approach showed smaller MRE except Toybox with all sample sizes and Fiasco with *n* samples.

4.5 Discussion

RQ1. Do prior work scale for larger HCSs?

From our experiments, prior work showed substantially higher MRE for larger HCSs. Contrary to our expectations, larger sample sizes did not necessarily yield lower MRE for larger HCSs. Thus, their claim that the performance model can be learned with small number of samples does not hold for larger HCSs.

For now, we are not clear why larger HCSs are showing such a high MRE. For uClibc-ng, we the loss function diverged during training, which suggest better hyper-parameter optimization is required. For the other systems, we are wondering if the data is too sparse to the samples we have is not enough to reach MRE close to the ones shown for smaller HCSs.

For the rest of this project, we will try to analyze the issues such as training with larger sample sizes, applying dimensionality reduction, and utilizing other learning methods more suitable for sparse data.

Conclusion: Prior work could not scale to larger HCSs.

RQ2: Can we find a better neural architecture for performance models?

We observed that our approach shows slightly lower MRE compared to other approaches in general. It suspect that our manual tuning was better fitted for the specific HCSs we tested, and provided better results than their automatic hyperparameter selection. Nevertheless, the MRE of our approach is still very high compared to the result. We believe that analyzing the issue of prior work may provide us a way to reduce the MRE further.

Conclusion: Our approach shows slight improvement, but not significant yet.

5 CONCLUSION

State-of-the-art performance modeling approaches for HCSs aim to achieve good estimation accuracy with using small number of samples as possible. Although they have demonstrated that it is possible for small HCSs, our evaluation shows that larger HCSs cannot be learned as such. We believe that this is a big limitation. The dataset for small HCSs were introduced in 2012 [14] and still utilized to date, which kept this scalabiltiy issue unforeseen.

For the remainder of this project, we aim to replicate more approaches and try to identify the reason behind this limited scalability. We believe that identifying this cause is the first step to propose a scalable method that outperforms prior work.

REFERENCES

[1] 2018. Fiasco website. http://axtls.sourceforge.net/.

[2] Mathieu Acher, Hugo Martin, Juliana Pereira, Arnaud Blouin, Jean-Marc Jézéquel, Djamel Khelladi, Luc Lesoil, and Olivier Barais. 2019. Learning Very Large Configuration Spaces: What Matters for Linux Kernel Sizes. (2019).

[3] Xavier Glorot and Yoshua Bengio. 2010. Understanding the difficulty of training deep feedforward neural networks. In *Proceedings of the thirteenth international conference on artificial intelligence and statistics*. 249–256.

[4] Siegmund Apel Gou, Czarnecks and Wasowski. 2013. Variability-aware performance prediction: A statistical learning approach. (2013), 301–311.

[5] Jianmei Guo, Dingyu Yang, Norbert Siegmund, Sven Apel, Atrisha Sarkar, Pavel Valov, Krzysztof Czarnecki, Andrzej Wasowski, and Huiqun Yu. 2018. Data-efficient performance learning for configurable systems. *Empirical Software Engineering* 23, 3 (2018), 1826–1867.

[6] Huong Ha and Hongyu Zhang. 2019. DeepPerf: performance prediction for configurable software with deep sparse neural network. In *2019 IEEE/ACM 41st International Conference on Software Engineering (ICSE)*. IEEE, 1095–1106.

[7] H. Ha and H. Zhang. 2019. Performance-Influence Model for Highly Configurable Software with Fourier Learning and Lasso Regression. In *2019 IEEE International Conference on Software Maintenance and Evolution (ICSME)*. 470–480. https://doi.org/10.1109/ICSME.2019.00080

[8] Jeho Oh, Paul Gazzillo, Don Batory, Marijn Heule, and Maggie Myers. 2019. *Uniform Sampling from Kconfig Feature Models*. Technical Report TR-19-02. University of Texas at Austin, Department of Computer Science.

[9] Jörg Liebig, Alexander Von Rhein, Christian Kästner, Sven Apel, Jens Dörre, and Christian Lengauer. 2013. Scalable analysis of variable software. In *Proceedings of the 2013 9th Joint Meeting on Foundations of Software Engineering*. ACM, ACM, New York, NY, USA, 81–91.

[10] Daniel-Jesus Munoz, Mónica Pinto, and Lidia Fuentes. 2018. Finding correlations of features affecting energy consumption and performance of web servers using the HADAS eco-assistant. *Computing* 100, 11 (01 Nov 2018), 1155–1173. https://doi.org/10.1007/s00607-018-0632-7

[11] Jeho Oh, Don Batory, Margaret Myers, and Norbert Siegmund. 2017. Finding near-optimal configurations in product lines by random sampling. In *Proceedings of the 2017 11th Joint Meeting on Foundations of Software Engineering*. ACM, IEEE/ACM, Piscataway, NJ, USA, 61–71.

[12] Juliana Alves Pereira, Hugo Martin, Mathieu Acher, Jean-Marc Jézéquel, Goetz Botterweck, and Anthony Ventresque. 2019. Learning Software Configuration Spaces: A Systematic Literature Review. *arXiv preprint arXiv:1906.03018* (2019).

[13] Norbert Siegmund, Alexander Grebhahn, Sven Apel, and Christian Kästner. 2015. Performance-influence Models for Highly Configurable Systems. In *Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering (ESEC/FSE 2015)*. ACM, New York, NY, USA, 284–294. https://doi.org/10.1145/2786805.2786845

[14] Norbert Siegmund, Sergiy S. Kolesnikov, Christian Kästner, Sven Apel, Don Batory, Marko Rosenmüller, and Gunter Saake. 2012. Predicting Performance via Automated Feature-interaction Detection. In *Proceedings of the 34th International Conference on Software Engineering (ICSE '12)*. IEEE Press, Piscataway, NJ, USA, 167–177. http://dl.acm.org/citation.cfm?id=2337223.2337243

[15] John Villanueva. 2009. How many atoms are there in the Universe? https://www.universetoday.com/36302/atoms-in-the-universe/.