PLUS: Performance Learning for Uncertainty of Software

Catia Trubiani
Gran Sasso Science Institute
L'Aquila, Italy

Sven Apel
University of Passau
Passau, Germany

***Abstract*—Uncertainty is particularly critical in software performance engineering when it relates to the values of important parameters such as workload, operational profile, and resource demand, because such parameters inevitably affect the overall system performance. Prior work focused on monitoring the performance characteristics of software systems while considering influence of configuration options. The problem of incorporating uncertainty as a first-class concept in the software development process to identify performance issues is still challenging. The PLUS (Performance Learning for Uncertainty of Software) approach aims at addressing these limitations by investigating the specification of a new class of performance models capturing how the different uncertainties underlying a software system affect its performance characteristics. The main goal of PLUS is to answer a fundamental question in the software performance engineering domain: How to model the variable configuration options (i.e., software and hardware resources) and their intrinsic uncertainties (e.g., resource demand, processor speed) to represent the performance characteristics of software systems? This way, software engineers are exposed to a quantitative evaluation of their systems that supports them in the task of identifying performance critical configurations along with their uncertainties.**

***Index Terms*—Performance, Machine Learning, Uncertainty**

I. Introduction

Many application domains, such as distributed systems and cyber-physical systems, have strict performance requirements (e.g., system response time). In fact, it has been demonstrated that, if performance requirements are not met, a variety of negative consequences (such as damaged customer relations, business failures, lost income) can impact a software system's success [1]. Quantitative (predictive or measured) performance models are of key relevance to understand the presence of performance issues. However, it is often difficult to quantify the performance behavior of a software system since performance is affected by many factors: early design choices (e.g., software component interactions), implementation details (e.g., different choices of data structures), deployment (e.g., settings of hardware resources), and the execution environment (e.g., operating system). Moreover, performance evaluation is typically also affected by a certain degree of uncertainty related to the development process, as there are parameters that, at early stages, are simply unknown (e.g., the deployment environment). This initial uncertainty propagates to later stages involving the software system's execution and evolution in that there will be parameters varying at runtime (e.g., fluctuating workloads) [2].

Uncertainty is a fact of today's software-intensive systems, and it is necessary to consider it as a first-class concern in the design, implementation, and deployment of those systems [3]. By uncertainty we mean not knowing the exact impact of system's constituents parts (e.g., software components) along with external factors influencing the system's execution (e.g., workload) on performance properties of interest. That is, we are not able to precisely specify the impact of system settings as a crisp performance value.

To keep track of uncertainties, feature models can be used. In fact, they represent a well-known formalism for capturing variability, commonalities, and configuration rules of software systems. A *feature model* is a compact representation of the set of products of a software product line or configurations of a system, in terms of features and logical relationships among them [4]. To illustrate the role and the importance of uncertainty, an example of a feature model is shown in Figure 1. It represents a wind generator system (WGS) consisting of a set of services regulated by the speed of wind. There are two mandatory features:

- $SupplyEnergy$ (SE): provide energy sources to users that are interested to acquire them from the system;
- $MonitorSpeed$ (MS): check regularly the speed of turbines to verify the correct functionality of the system.

If the wind is light, then one optional feature is activated:

- $IncreasePrice$ (IP): the production of energy is reduced, and it is necessary to increase the price to which the new users buy the energy under production.

If the wind is strong, then $IncreasePrice$ feature is deactivated, the $MonitorSpeed$ feature has an increased monitoring frequency, and the

- $PromoteEnergy$ (PE) optional feature is activated: the production of energy is augmented, and it is necessary to provide discounts for attracting a larger number of users.

The performance analysis of WGS involves uncertain parameters that are unknown at early development (e.g., the number of users interested in acquiring energy), but also further time-varying parameters (e.g., the frequency of monitoring the turbines conditioned by the speed of the wind).

Prior work focused on monitoring the performance of software products while considering the influence of configuration options [5]–[9]. But it is infeasible to evaluate all system variants when increasing the number of features and all possible

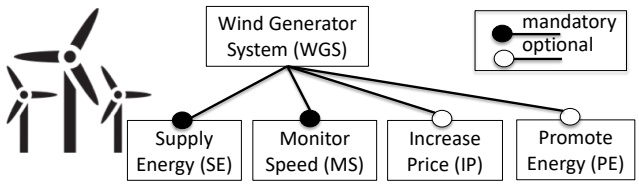

Fig. 1. Feature Model of a Wind Generator System (WGS)

values for uncertain parameters since the configuration space explodes. In this context, one challenge is to understand how many samples are needed to provide accurate performance estimates by means of machine learning (ML) techniques. There is an inevitable trade-off between the number of evaluated samples and the accuracy of ML techniques [10], hence we have to derive an ideal set of performance measurements leading to an acceptable prediction accuracy. The usage of ML techniques mitigates the need for evaluation of all system variants at the cost of providing performance indicators that are approximations instead of actual measurements.

In this paper, we present a new approach, called PLUS (Performance Learning for Uncertainty of Software), that aims at integrating *uncertainty* into the software development process as a first-class concern, deserving its own analysis to support the performance evaluation of software systems. The key intuition is that uncertain parameters can only be controlled by making them explicit and by consulting their specification (e.g., lower/upper bounds or probability distributions) to drive the performance evaluation process. To this end, we propose a new class of performance models, called *uncertainty-influence models*, that quantify the influence of uncertain parameters on system performance characteristics.

Uncertainty-influence models describe how configuration options (i.e., software and hardware resources) and their intrinsic uncertainties (e.g., system workload) influence the performance behavior of a software system. For example, in Figure 1, the two alternative features (i.e., IP and PE) are associated to the energy production; $PromoteEnergy$ is activated if the wind is strong. This implies that the performance (e.g., the system's response time) is also affected by this environmental aspect, since the system computation is exposed to a varying presence of available features.

PLUS makes the performance analysis an integrated activity in the software development process, so that performance characteristics and their uncertainties are continuously exposed to system engineers, who can proactively select the configuration alternatives suitable to avoid performance issues.

II. Related work

Two lines of research are related to PLUS: (i) performance evaluation based on machine learning (ML) and (ii) uncertainty analysis. Guo et al. [11], [12] makes use of a statistical learning technique to build an explicit performance model representing the correlation between feature selections and their performance characteristics. Product-line performance verification is conducted to investigate the ability to identify defects [8]. Sarkar et al. [10], [13], [14] study multiple sampling strategies to evaluate the performance of configurable systems considering the prediction accuracy and the measurement effort simultaneously. Siegmund et al. [15] use a combination of kernel density estimation and a genetic algorithm to rescale a given attribute-value profile to a variability model. Nair et al. [16] found that, even if performance models are not accurate when deriving analysis results, they are valuable in ranking system configurations. However, the state-of-the-art approaches [8], [11], [13], [15]–[17] do not consider uncertainties in the learning of performance characteristics.

Uncertainty may span multiple dimensions: (i) known knowns (uncertainty does not exist); (ii) known unknowns (uncertainty is shared among the stakeholders); (iii) unknown unknowns (uncertainty is uncertain itself). Perez et al. [18] propose a methodology to guide software engineers in recognizing and managing different types of uncertainty. There are some approaches aiming to optimize the Quality-of-Service (QoS) properties of systems when variability is present, e.g., workload bursts are incorporated into models to derive more accurate performance predictions [19], or reconfigurations at runtime [20]. Esfahani et al. [21] outline that existing decision-making approaches do not provide a quantitative method for comparing architectural alternatives. This issue has been tackled by some approaches that investigate the idea of embedding uncertainties in the software analysis process. Uncertainty and its propagation in availability analytical models is investigated by Devaraj et al. [22]; others consider to embed the parameters uncertainties in model-based performance analysis while dealing with variable features [23], and in the reliability evaluation of software architectures [24]. However, none of these approaches [18]–[24] quantifies the propagation of input uncertainties into performance analysis results.

Summarizing, to the best of our knowledge, applying ML to performance evaluation of software systems is not effective without embedding the specification of uncertain parameters. PLUS aims at embedding probability distribution functions (PDFs) of uncertain parameters as part of the sampled system configurations and determining an optimal set of these samples balancing accuracy and analysis effort. To this end, we will specify a new class of performance models, called *uncertainty-influence models*, which capture the underlying uncertainties of a software system with its performance characteristics, allowing us to efficiently learn the influence of uncertain parameters on performance.

III. The PLUS Approach

Figure 2 provides an overview of PLUS, which consists of four main activities: sampling, learning, prediction, and optimization. Note that the feedback arrow from optimization to sampling indicates a round-trip process; in fact, the selected optimal system configurations can be evaluated until no performance issues are experienced anymore. This introduces the possibility of selecting different features and/or changes the bounds and the distributions of uncertain parameters, thus facilitating a deeper investigation of how uncertainty propagates to the performance analysis results.

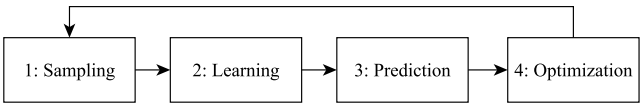

Fig. 2. Overview of PLUS

To better explain the proposed activities, Table I reports their application on the WGS example (see Figure 1), and compares PLUS w.r.t. the state of the art [6].

1: Sampling. The first step consists of selecting the system configurations, along with their uncertain parameters, and quantifying their performance. To sample uncertain parameters, a common method is to associate an ad-hoc specification to them that provides knowledge on their nature, such as lower and upper bounds, PDFs, etc. Depending on the source of uncertainty, parameters may involve different characteristics, resulting in different distributions (e.g., Uniform, Normal, etc.) of possible values. State-of-the-art approach [6] defines the sampling step as follows: let C be the overall set of configurations, that is, $C = \{c_1, \ldots, c_n\}$, we measure them individually and associate a real value with each, that is, $\pi : C \rightarrow \mathbb{R}$ (set of Real numbers); $S \subseteq C$ is the sample set, which depends on the sampling heuristics (e.g., t-wise sampling) [6]. Performance-influence models have been defined as a function from system configurations to a performance measure, where performance can be any quantifiable property (e.g., system response time) [6]. In Table I, we can notice that feature SE has been measured and shows a response time of 2 seconds. In PLUS, we define sampling as follows: let C be the overall set of configurations, we measure each of them and associate a probability distribution function (PDF) with each, not a value, that is, $\pi : C \rightarrow \mathbb{P}$ (set of Probability values). For example, the normal distribution is denoted by $\mathcal{N}(\mu, \sigma^2)$ with mean μ and variance σ^2, or the uniform distribution is denoted by $\mathcal{U}(a, b)$, where a and b are its minimum and maximum values, respectively. From Table I, we can see that the MS feature shows a response time varying with a uniform distribution between 2 and 4 seconds. The PDFs and their parameters capture the uncertainty that has been observed while measuring the performance of the system configurations. This information is then exploited to learn, predict, and optimize the overall system performance.

2: Learning. The second step consists of deriving a model that correlates the influence of individual configuration options and their combinations with performance. The state of the art [6] defines the learning step as follows: learn: $S \rightarrow M$; M has the following form in our case: $M = \sum_{t \in T} r_t \cdot t$; T represents the set of terms including features and interactions along with their factor structure, e.g., $T = \{SE, MS, SE \cdot MS\}$, $r_t \in \mathbb{R}$, and t are the individual features or interactions among the features, if any. For example, in Table I, the predictive model for WGS includes the features: SE, MS, IP, and PE; it includes further elements that are interactions between SE and MS as well as IP and PE. In PLUS, we define learning as follows: learn: $S \rightarrow M$; here M has the following form: $M = \sum_{t \in T} r_t \cdot p_t \cdot t$; $p_t \in \mathbb{P}$ represents the PDF of the t-th feature or interaction. An issue that arises in this model is how to combine different PDFs. For example, if features include uncertain parameters regulated by normal and uniform distributions, algebraic methods will be used to sum/combine ($\bowtie$) these two probability functions, but it is unclear how to do that for the general case. In Table I, the predictive model for WGS reports the PDFs of specific features, e.g., p_{SE}, and the interactions to be handled as combination of PDFs, e.g., $p_{SE} \bowtie p_{MS}$.

3: Prediction. The third step consists of using the derived model for predicting performance properties of system configurations not measured before. State-of-the-art approach [6] defines the prediction step as follows: pred: $(C, M) \rightarrow \mathbb{R}$. In Table I, we can notice that the selection of SE, MS, IP, and PE leads to a predicted response time of 140 seconds. In PLUS, instead, we define prediction as follows: pred: $(C, M) \rightarrow \mathbb{P}$. This causes the question of how to predict the PDF of a configuration from the derived model. Specifically, starting from a system configuration C and the model learned in the previous step, it is not obvious to predict the interpolation of PDFs associated to the selected features, and the resulting PDF for the performance results (e.g., system response time) of such system configuration. For example, in Table I, we can see that, when selecting two features showing an interaction, say SE and MS, the PDF associated to the prediction of performance results requires the combination of their distributions: $\mathcal{U}(1, 2) \bowtie \mathcal{U}(2, 4)$.

4: Optimization. The fourth step consists of deriving the optimal system configuration, that is a selection of features that optimizes a certain goal (e.g., response time). In PLUS, we estimate the ability of the software system to tolerate the uncertainty of intrinsic input parameters. Optimization concerns the following actions: (i) select the features that do not generate performance issues; (ii) identify the bounds of uncertain parameters values that lead to performance issues; (iii) modify the feature model and/or the uncertainty specification to get alternatives that meet the stated performance requirements. State-of-the-art approach [6] defines the optimization step as follows: opt: $(O, M) \rightarrow C$ where $\pi(C)$ is optimal according to O (e.g., max/min value). In Table I, when minimizing the system response time ($minVal$), SE and MS are selected only. In PLUS, instead, we define optimization as follows: opt: $(O, M) \rightarrow C$ where $\pi(C)$ is optimal according to O (e.g., max/min variance, largest/smallest outside areas). The difference to the state of the art is that for deriving the optimal configuration, uncertain parameters play a crucial role since they are associated to probabilities, and their distributions become a further source of optimization. In Table I we report the case of maximizing the variance ($maxVar$) of the system response time, and all features are selected. Differently from [6], the optimality of performance results is not determined by a single value, but we need to consider the distribution of values obtained with PDFs of uncertain parameters and their combinations, since they contribute to the overall system performance.

TABLE I
COMPARING THE STATE OF THE ART [6] AND PLUS BY THE EXAMPLE OF THE WIND GENERATOR SYSTEM (WGS)

	State of the art			PLUS		
	Configurations	Features	π	Configurations	Features	π
Sampling:	c_1	$\{SE\}$	2.0	c_1	$\{SE\}$	$\mathcal{U}(1,2)$
	c_2	$\{MS\}$	4.0	c_2	$\{MS\}$	$\mathcal{U}(2,4)$
	c_3	$\{PE\}$	6.0	c_3	$\{PE\}$	$\mathcal{U}(4,6)$
	c_4	$\{IP\}$	8.0	c_4	$\{IP\}$	$\mathcal{U}(6,8)$
	c_5	$\{SE,\ MS\}$	14.8	c_5	$\{SE,\ MS\}$	$\mathcal{N}(15,2)$
	c_6	$\{SE,\ PE\}$	20.3	c_6	$\{SE,\ PE\}$	$\mathcal{N}(20,2)$
	c_7	$\{SE,\ IP\}$	25.7	c_7	$\{SE,\ IP\}$	$\mathcal{N}(25,2)$
	c_8	$\{MS,\ PE\}$	30.2	c_8	$\{MS,\ PE\}$	$\mathcal{N}(30,2)$
	...	...	...	...	...	...
Learning:	$\mathsf{learn}(S_{WGS}) = 5 + 10 \cdot SE + 20 \cdot MS + 30 \cdot IP + 40 \cdot PE + 15 \cdot SE \cdot MS + 20 \cdot IP \cdot PE$			$\mathsf{learn}(S_{WGS}) = 5 + 10 \cdot \mathcal{U}(1,2) \cdot SE + 20 \cdot \mathcal{U}(2,4) \cdot MS + 30 \cdot \mathcal{U}(6,8) \cdot IP + 40 \cdot \mathcal{U}(4,6) \cdot PE + 15 \cdot \mathcal{N}(15,2) \cdot SE \cdot MS + \ldots$		
Prediction:	$\mathsf{pred}(SE \mapsto 1, MS \mapsto 1, IP \mapsto 1, PE \mapsto 1), M_{WGS}) = 5 + 10 + 20 + 30 + 40 + 15 + 20 = 140$			$\mathsf{pred}(SE \mapsto 1, MS \mapsto 1, IP \mapsto 0, PE \mapsto 0), M_{WGS}) = 5 + 10 \cdot \mathcal{U}(1,2) + 20 \cdot \mathcal{U}(2,4) + 15 \cdot \mathcal{U}(1,2) \bowtie \mathcal{U}(2,4) + \ldots$		
Optimization:	$\mathsf{opt}(minVal, M_{WGS}) = (SE \mapsto 1, MS \mapsto 1, IP \mapsto 0, PE \mapsto 0)$			$\mathsf{opt}(maxVar, M_{WGS}) = (SE \mapsto 1, MS \mapsto 1, IP \mapsto 1, PE \mapsto 1)$		

IV. SUMMARY

The key idea of PLUS is to specify a new class of performance models, uncertainty-influence models, that embed heterogeneous uncertainties (e.g., software components, hardware platforms, workload, operational profile, environmental changes) as first-class entities. Performance analysis captures the system evolution through a dataset of configuration options including: (i) the presence of features along the time, (ii) system uncertainties regulated by PDFs and represented by time series, and (iii) performance measurements expressed as distributions of values. Uncertainty-influence models aim at quantifying how the selected features and their uncertainties affect the system performance, thus to identify the configuration options that most likely generate performance issues. This supports software engineers in the performance evaluation of software systems subject to uncertainties.

ACKNOWLEDGMENT

This work has been partially supported by the German Research Foundation (AP 206/11).

REFERENCES

[1] A. L. Goel and K. Okumoto, "Time-dependent error-detection rate model for software reliability and other performance measures," *IEEE Transactions on Reliability*, vol. 28, no. 3, pp. 206–211, 1979.

[2] R. Calinescu, C. Ghezzi, M. Kwiatkowska, and R. Mirandola, "Self-adaptive software needs quantitative verification at runtime," *Communications of the ACM*, vol. 55, no. 9, pp. 69–77, 2012.

[3] D. Garlan, "Software engineering in an uncertain world," in *Workshop on Future of software engineering research*, 2010, pp. 125–128.

[4] D. Benavides, S. Segura, and A. Ruiz-Cortés, "Automated analysis of feature models 20 years later: A literature review," *Information Systems*, vol. 35, no. 6, pp. 615–636.

[5] N. Siegmund, S. S. Kolesnikov, C. Kästner, S. Apel, D. S. Batory, M. Rosenmüller, and G. Saake, "Predicting performance via automated feature-interaction detection," in *ICSE*, 2012, pp. 167–177.

[6] N. Siegmund, M. Rosenmüller, M. Kuhlemann, C. Kästner, S. Apel, and G. Saake, "SPL Conqueror: Toward optimization of non-functional properties in software product lines," *Software Quality Journal*, vol. 20, no. 3-4, pp. 487–517, 2012.

[7] N. Siegmund, A. von Rhein, and S. Apel, "Family-based performance measurement," in *GPCE*, 2013, pp. 95–104.

[8] S. Apel, A. v. Rhein, P. Wendler, A. Größlinger, and D. Beyer, "Strategies for product-line verification: Case studies and experiments," in *ICSE*, 2013, pp. 482–491.

[9] N. Siegmund, A. Grebhahn, S. Apel, and C. Kästner, "Performance-influence models for highly configurable systems," in *ESEC/FSE*, 2015, pp. 284–294.

[10] S. Kolesnikov, N. Siegmund, C. Kastner, A. Grebhahn, and S. Apel, "Tradeoffs in modeling performance of highly-configurable software systems," *Software and System Modeling*, 2018, online first.

[11] J. Guo, K. Czarnecki, S. Apel, N. Siegmund, and A. Wasowski, "Variability-aware performance prediction: A statistical learning approach," in *ASE*, 2013, pp. 301–311.

[12] J. Guo, D. Yang, N. Siegmund, S. Apel, A. Sarkar, P. Valov, K. Czarnecki, A. Wasowski, and H. Yu, "Data-efficient performance learning for configurable systems," *Empirical Software Engineering*, vol. 23, no. 3, pp. 1826–1867, 2018.

[13] A. Sarkar, J. Guo, N. Siegmund, S. Apel, and K. Czarnecki, "Cost-efficient sampling for performance prediction of configurable systems," in *ASE*, 2015, pp. 342–352.

[14] C. Kaltenecker, A. Grebhahn, N. Siegmund, J. Guo, and S. Apel, "Distance-based sampling of software configuration spaces," in *ICSE*, 2019.

[15] N. Siegmund, S. Sobernig, and S. Apel, "Attributed variability models: outside the comfort zone," in *ESEC/FSE*, 2017, pp. 268–278.

[16] V. Nair, T. Menzies, N. Siegmund, and S. Apel, "Using bad learners to find good configurations," 2017, pp. 257–267.

[17] V. Nair, Z. Yu, T. Menzies, N. Siegmund, and S. Apel, "Finding faster configurations using FLASH," *IEEE Trans. on Soft. Eng.*, 2018, online first.

[18] D. Perez-Palacin and R. Mirandola, "Dealing with uncertainties in the performance modelling of software systems," in *QoSA*, 2014, pp. 33–42.

[19] G. Casale, N. Mi, L. Cherkasova, and E. Smirni, "Dealing with burstiness in multi-tier applications: Models and their parameterization," *IEEE Trans. on Soft. Eng.*, vol. 38, no. 5, pp. 1040–1053, 2012.

[20] F. Duarte, R. Gil, P. Romano, A. Lopes, and L. Rodrigues, "Learning non-deterministic impact models for adaptation," in *SEAMS*, 2018, pp. 196–205.

[21] N. Esfahani, S. Malek, and K. Razavi, "Guidearch: guiding the exploration of architectural solution space under uncertainty," in *ICSE*, 2013, pp. 43–52.

[22] A. Devaraj, K. Mishra, and K. S. Trivedi, "Uncertainty propagation in analytic availability models," in *SRDS*, 2010, pp. 121–130.

[23] L. Etxeberria, C. Trubiani, V. Cortellessa, and G. Sagardui, "Performance-based selection of software and hardware features under parameter uncertainty," in *QoSA*, 2014, pp. 23–32.

[24] I. Meedeniya, A. Aleti, and L. Grunske, "Architecture-driven reliability optimization with uncertain model parameters," *Journal of Systems and Software*, vol. 85, no. 10, pp. 2340–2355, 2012.