Herausgeber et al. (Hrsg.): Name-der-Konferenz,
Lecture Notes in Informatics (LNI), Gesellschaft für Informatik, Bonn 2017 11

Identifying Software Performance Changes Across Variants and Versions

Stefan Mühlbauer[1] Sven Apel[2] Norbert Siegmund[3]

Abstract: Performance changes of configurable software systems can occur and persist throughout their lifetime. Finding optimal configurations and configuration options that influence performance is already difficult, but in the light of software evolution, configuration-dependent performance changes may lurk in a potentially large number of different versions of the system. Building on previous work, we combine two perspectives—variability and time—and devise an approach to identify configuration-dependent performance changes *retrospectively* across the software variants and versions of a software system. In a nutshell, we iteratively sample pairs of configurations and versions and measure the respective performance, which we use to actively learn a model that estimates how likely a commit introduces a performance change. For such commits, we infer the configuration options that best explain observed performance changes. Pursuing a search strategy to measure selectively and incrementally further pairs, we increase the accuracy of identified change points related to configuration options and interactions. Our evaluation with both real-world software systems and synthesized data demonstrates that we can pinpoint performance shifts to individual configuration options and commits with high accuracy and at scale.

Keywords: Software Performance; Configurable Software Systems; Software Evolution

Modern software systems often provide configuration options to customize their functionality and non-functional characteristics, such as energy consumption and performance. Determining the influence of individual configuration options and their interactions on performance (henceforth called performance influences) follows a two-step process. First, a set of configurations is selected and then measured via a benchmark or an application-specific workload. Second, a machine learning technique, such as linear regression or classification and regression trees, is applied to learn a performance model using the measurements as a training set. With this model, we can estimate the performance of unseen configurations.

However, software is not a static entity, and as the code base changes so does performance. If the performance influences change, prediction models from older versions make inaccurate estimations for newer versions. Maintaining accurate prediction models throughout a software's lifetime vastly increases the cost of measurement since we would blindly select configurations throughout the version history of a system to find the relevant change points.

By contrast, information about which performance influences change at what revision could guide practitioners to re-learn models only upon such an occurrence or guide future testing efforts. We address the problem of detecting change points in space (configuration dimension)

[1] Universität Leipzig, Institut für Informatik, Leipzig, Deutschland, muehlbauer@informatik.uni-leipzig.de
[2] Universität des Saarlandes, Saarland Informatics Campus, Saarbrücken, Deutschland, apel@cs.uni-saarland.de
[3] Universität Leipzig, Institut für Informatik, Leipzig, Deutschland, norbert.siegmund@informatik.uni-leipzig.de

and time (version dimension) and devise a novel approach that *retrospectively* identifies shifts in performance influences across a software system's development history [MAS20].

In previous work, we presented how one can identify performance shifts in a series of performance measurements of a single software configuration [MAS19]. Since configuration-specific performance changes often emerge from changes in performance influences, it is key to identify such causative options. To this end, we pinpoint performance change not only to a specific commit, but also a set of responsible configuration options. Starting from performance measurements of a small sample of random configurations and revisions, our active learning and sampling strategy first estimates the temporal location of likely change points (candidate commits). If a performance shift is observed at a candidate commit for multiple configurations, the shift is marked for further analysis (candidate shift). Second, we attribute each candidate shift to one or more configuration options based on the configurations for which it manifests. Last, to increase the reliability of these estimations, we iteratively add new measurements to the training set and repeat the previous steps until the set of identified change points does not change over a couple of iterations

The main challenge with this approach is the combinatorial complexity of the configuration-and-revision space. To wisely select configurations and revisions, we balance the budget of measurements between two objectives, *exploration* and *exploitation*. To capture undetected performance shifts, we select configurations and revisions randomly (exploration). We dedicate the remaining budget to re-evaluating the approach's candidate solutions in each iteration (exploitation). Throughout the iterations, we shift the ratio of measurements towards exploitation.

We show for three real-world software systems that our approach can precisely pinpoint performance changes to a single commit and set of configuration options. Further experiments with synthetic data demonstrate its scalability regarding the number of commits of a version history, the number of configuration options, and the degree of option interactions.

Data Availability We provide all measurement data via the original companion web site [4].

Literatur

[MAS19] Mühlbauer, S.; Apel, S.; Siegmund, N.: Accurate Modeling of Performance Histories for Evolving Software Systems. In: Proceedings of the International Conference on Automated Software Engineering (ASE). IEEE, S. 640–652, Nov. 2019.

[MAS20] Mühlbauer, S.; Apel, S.; Siegmund, N.: Identifying Software Performance Changes across Variants and Versions. In: Proceedings of the International Conference on Automated Software Engineering (ASE). ACM, S. 611–622, 2020.

[4] https://archive.softwareheritage.org/browse/origin/directory/?origin_url=https://github.com/AI-4-SE/Changepoints-Across-Variants-And-Versions