Automated Inference of Goal-Oriented Performance Prediction Functions

Dennis Westermann, Jens Happe, Rouven Krebs, Roozbeh Farahbod
SAP Research
Vincenz-Priessnitz-Str. 1
Karlsruhe, Germany
{dennis.westermann, jens.happe, rouven.krebs, roozbeh.farahbod}@sap.com

ABSTRACT

Understanding the dependency between performance metrics (such as response time) and software configuration or usage parameters is crucial in improving software quality. However, the size of most modern systems makes it nearly impossible to provide a complete performance model. Hence, we focus on scenario-specific problems where software engineers require practical and efficient approaches to draw conclusions, and we propose an automated, measurement-based model inference method to derive goal-oriented performance prediction functions. For the practicability of the approach it is essential to derive functional dependencies with the least possible amount of data. In this paper, we present different strategies for automated improvement of the prediction model through an adaptive selection of new measurement points based on the accuracy of the prediction model. In order to derive the prediction models, we apply and compare different statistical methods. Finally, we evaluate the different combinations based on case studies using SAP and SPEC benchmarks.

1. INTRODUCTION

Performance (i.e., response time, throughput, and resource utilization) is one of the most important quality characteristics as it directly influences user perception and costs. In order to provide responsive and scalable applications, software providers need a profound understanding of the performance properties of the system throughout the software engineering lifecycle, which in turn calls for appropriate tools and processes that facilitates achieving such an understanding with minimal additional effort.

However, the performance of a system is subject to many influences such as system configuration and usage. In order to get an understanding of the performance characteristics of a system, software architects have to identify the performance-relevant parameters and analyse the influences of these parameters on the performance metrics of interest. Considering the various layers and different technologies involved in today's software systems, this can become a very challenging task.

Recent performance engineering research approaches [14] use architectural information and detailed performance behavior descriptions in order to build prediction models. In most cases, the performance models are a combination of simulation models built using domain-specific languages and measurements to calibrate, validate or extend the models [2, 4]. In industrial practice, performance measurements are, for example, used to benchmark systems, customize configuration settings, or test the quality of a new release before shipment [22, 24]. In both cases, the amount of possible parameter combinations and configurations as well as the required expert knowledge make the evaluation process time consuming, costly, and error-prone. Although many tools provide automation for generating load and getting monitoring information, there is still a lot of manual effort remaining to analyze the measured data and to decide how many and which measurements to conduct in order to reach a certain goal (e.g., finding a performance-optimized configuration).

Our overall approach aims at reducing this manual effort as much as possible and consists of five major steps (based on [10]): (i) Defining the context and the goal of the performance evaluation (ii) specifying potential performance influencing parameters and setting up a measurement environment that allows to change these parameters programmatically, (iii) identifying those parameters that influence performance, (iv) determining potential interdependencies between performance-relevant parameters, and (v) quantifying the functional relationship between interdependent, performance-relevant parameters and the performance metric of interest. In this paper, we focus on on automating the last step. Although existing literature [27] points out that the number of input parameters in such functions can be limited to up to eight parameters without loosing any information, the number of potential measurement points in the measurement space spanned by these parameters can still become very large. Thus, it requires solid investigations on how to explore that space automatically and efficiently in order to support the software architect in building accurate prediction functions. We present three measurement point selection algorithms that iteratively select new measurement points based on the accuracy of the prediction model inferred with the data measured in previous iterations. To validate the inferred prediction models after each iteration, we compare three different strategies. To derive the prediction models we apply different statistical regression and interpolation methods, namely MARS, CART, GP,

ASE'12, September 3–7, 2012, Essen, Germany

and Kriging. MARS and CART have already been successfully applied in different performance prediction case studies [2, 27]. In our previous publication [30], we showed that the geostatistical interpolation method Kriging can be an efficient estimator for two-dimensional performance questions. In this paper, we apply a combination of classical multidimensional scaling and Kriging in order to use the method to quantify multidimensional functional dependencies. In [6], we developed an analysis method that applies genetic programming (GP) to derive performance prediction functions from measurement data.

In our case studies, we applied different combinations of measurement point selection algorithms, statistical model inference methods, and validation methods in order to automatically derive performance prediction functions. Our main goal is to identify the most efficient (in terms of required measurement points to derive an accurate prediction model) combinations of measurement point selection algorithm, statistical model inference methods, and validation methods. Moreover, we want to evaluate if the automated measurement-based prediction approach is applicable in industrial scenarios with multiple variable configuration and workload parameters.

The contributions of this paper are as follows: (i) the identification of an automated approach that creates accurate prediction models with the least possible number of measurements, (ii) a flexible methodology that allows to evaluate the applicability of model inference methods from other research fields to the prediction of performance data (as demonstrated in this paper by the use of the geostatistical method Kriging), and to evaluate the efficiency and accuracy of different combinations of parameter space exploration strategies and statistical model inference methods with minimum effort, (iii) a detailed evaluation of a set of combinations in two industrial case studies where our results provide an overview on the applicability of the presented methods in practical scenarios and thus can help researchers and practitioners to choose an appropriate combination of methods for their specific scenarios.

The remainder of the paper is structured as follows. In Section 2, we give an overview on our overall approach for the derivation of performance prediction functions. In Section 3, we describe the three measurement point selection algorithms that we use in our evaluation. Section 4 introduces the statistical methods that we use to derive the prediction functions. A discussion on related research is provided in Section 5. In Section 6, we present the case studies that we use to evaluate the different combinations of measurement point selection algorithms, statistical model inference techniques, and validation strategies. In Section 7, we outline and discuss the results of the evaluation. Finally, Section 8 summarizes the paper and gives an outlook on future work.

2. SYSTEMATIC APPROACH

The goal of our work is to help stakeholders in the software engineering lifecycle, such as software architects, developers or system administrators, to answer specific performance questions by supporting them in understanding the performance characteristics of the software system under development. In this section, we provide an overview on our approach and outline how the content of this paper relates to the overall research direction.

A recent survey among IT administrators [27] shows that performance is often considered as one of the most important quality characteristics of a software system. But when it comes to the question how the administrators proceed in order to understand the performance behaviour of their systems, the answers are in most cases applying rules of thumb based on their own (or a colleague's) expert knowledge, running small benchmarks, or monitoring during runtime. However, none of these approaches provides dependable information with respect to the performance behaviour of a system under different conditions. We believe that this behaviour is caused by the lack of systematic approaches that provide a simple interface to the stakeholders and help them to answer their questions without the burden of analysing large tables of measurement data. The recent success of companies like New Relic [17] confirms this assumption.

In light of these observations, we aim at providing a systematic approach that automates performance engineering tasks as much as possible and allows software architects to focus on their specific questions. The approach consists of five central building blocks (based on [10]):

1. Define context and goal: In order to focus only on those parameters of a system that help answering the scenario-specific questions, it is an important precondition of any performance evaluation that the goals of the study as well as the system boundaries are well defined. This includes the selection of appropriate performance metrics and workloads.

2. Specify potential performance-relevant parameters and set up measurement environment: An important precondition for the automated evaluation of performance questions is the definition of the search space, i.e., a definition of parameters that could influence performance in a machine-readable format. In order to vary these parameters automatically, a measurement environment has to be prepared that allows programmatically changing parameter values .

3. Identify performance-relevant parameters: Once the goal and the system parameters are defined and the measurement environment is ready, we can start varying parameter values systematically. The first goal of these systematic measurements is to reduce the number of parameters by identifying those parameters that actually influence performance significantly.

4. Determine parameter interdependencies: As the number of performance-relevant parameters is often still too large to include them in one function, we need an additional step that detects interdependencies between parameters. This can in most cases reduce the number of parameters that have to be included in one function to less than eight [27].

5. Build performance prediction functions: In this final step, we quantify the functional relationship between a set of interdependent parameters and the performance metric of interest. To answer a specific performance question it might be necessary to create and combine a number of these performance prediction functions.

We assume that the first two steps are conducted collaboratively by scenario and performance engineering experts. In our previous research, we focused on the development of a framework called Software Performance Cockpit [1, 29] that supports the stakeholders in these tasks by providing an appropriate editor and a plugin-based architecture that

enables reuse and encapsulates best practices. In [29], we introduce the overall architecture of the framework as well as the general idea behind it. Integrating support for automatically executing steps 3 and 4 into the framework are subject to our future research. In this paper, we assume that these steps have been executed (e.g., as shown in [27]) and we get a set of interdependent parameters for which we want to automatically derive a performance prediction function. To derive these functions efficiently and automatically, we have developed and applied different combinations of parameter space exploration strategies and statistical model inference methods.

3. PARAMETER SPACE EXPLORATION

In this section, we introduce three measurement point selection algorithms (Section 3.1) and two model validation strategies (Section 3.2) that we implemented for the Software Performance Cockpit in order to explore the parameter spaces in systematic experiments. All methods break down the parameter space following the basic work flow depicted in Figure 1.

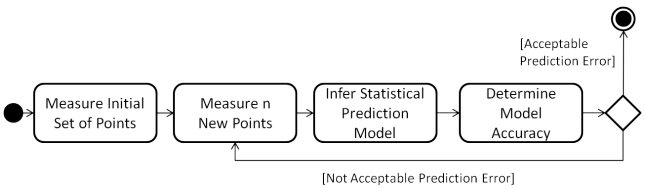

Figure 1: Generic Breakdown Workflow

After measuring an initial set of points, the algorithms start an iteration cycle. The initial measurements are optional but may be necessary to provide some stabilizing points for the analyses (e.g., some analysis methods require a minimum set of points in order to function correctly). Within the iteration cycle the first step is determining a set of new measurement points. Which and how many points are selected in each iteration differs between the different algorithms. In the next step, a statistical analyses is conducted based on the points measured so far. After that, the resulting prediction model is validated using one of the strategies described in Section 3.2. The validation provides the relative prediction error of the inferred model. If this error is below a predefined threshold, the algorithm terminates. If the error is above the threshold, a new iteration is started. In the following, we describe four concrete implementations of the abstract work flow described above. The Random Breakdown algorithm (Section 3.1.1) always determines new points in the whole parameter space. In contrary, the adaptive versions (sections 3.1.3 and 3.1.2) continuously split the parameter space in different sectors and determine new measurement points in those sectors that have the worst prediction accuracy.

3.1 Measurement Point Selection Strategies

3.1.1 Random Breakdown

The Random Breakdown algorithm randomly selects a fixed number of measurement points in each iteration (see Figure 2 (a)). The chosen points are always distributed across the whole parameter space. The number of points to be selected in each iteration can be defined by the performance analyst via the configuration. In some cases it might be more efficient to run more measurements before conducting an analysis while in other cases one wants to build the prediction models more frequently. This mainly depends on the size of the parameter space, the time it takes to get one measurement point and the time it takes to conduct an analysis.

3.1.2 Adaptive Equidistant Breakdown

In contrast to the algorithm described above, the Adaptive Equidistant Breakdown algorithm as well as the Adaptive Random Breakdown algorithm (described in the next section) take the locality and the size of single point prediction errors into account when determining measurement points for the next iteration (see Figure 1). The adaptive algorithms split the parameter space in sectors depending on the locality of the points with the largest prediction errors. We assume that a new measurement point at the area with the highest prediction error raises the accuracy of the overall model at most. Thus, only those sectors that have a prediction error larger than the predefined threshold will be split into equidistant sub sectors and only in these sub sectors new measurement points will be requested.

In the following, we describe the algorithm in detail. First, we introduce some basic data types, variables and functions followed by a listing of the algorithm.

We define $P = \{x | x \in \mathbb{R}^i \wedge x_i \in [0..1]\}$ as a set of all possible measurement points in the parameter space with normalized values. Elements of P are declared as p. Let the elements p_1 and p_2 be opposing positions describing the multidimensional space. Function $f_{center} : P \times P \rightarrow P$ returns the center of the two given points which is calculated by the element-wise arithmetic middle of the two vectors. Furthermore, function $f_{edges} : P \times P \rightarrow P^*$ returns a set of all edges of the embraced space given by p_1 and p_2. In addition, let $e_{sector} \in \mathbb{R}^+$ describe the error of the performance curve at a defined area (called *sector*) and $S = \{p_1 \times p_2 \times e_{sector} | p_1 \in P \wedge p_2 \in P \wedge e_{sector} \in \mathbb{R}^+\}$ the set of sectors in the configuration space. Three subsets of S control the measurement progress. A priority-controlled queue $Q \subset S$ containing sectors, where the error of the curve ran out of the acceptable threshold. The order of priority is based on e_{sector}. The collection $V \subset S$ is the validation set which contains all sectors that describe areas where an accurate prediction has already been observed. $M \subset S$ is the training set which contains the measurement results used to create a prediction model. All subsets of S are mutually disjoint. The function $predict_M : P \rightarrow \mathbb{R}$ creates a prediction results based on the given measurements M for a specific measurement point. The functionality of the method is based on the assumption, that the prediction error of the curve on $f_{center}(p_1, p_2)$ is representative for the error in the spatial field embraced by p_1 and p_2. The parameter $threshold \in \mathbb{R}^+$ is predefined by the performance analyst and gives an option to control the accuracy and thus the runtime of the method based on the chosen validation strategy $VS = \{vs | vs \in \{DSL, DSG, RVS\}\}$, where DSL is the Dynamic Sector validation with local scope, DSG is Dynamic Sector with global scope, and RVS is the Random Validation Set strategy (see Section 3.2).

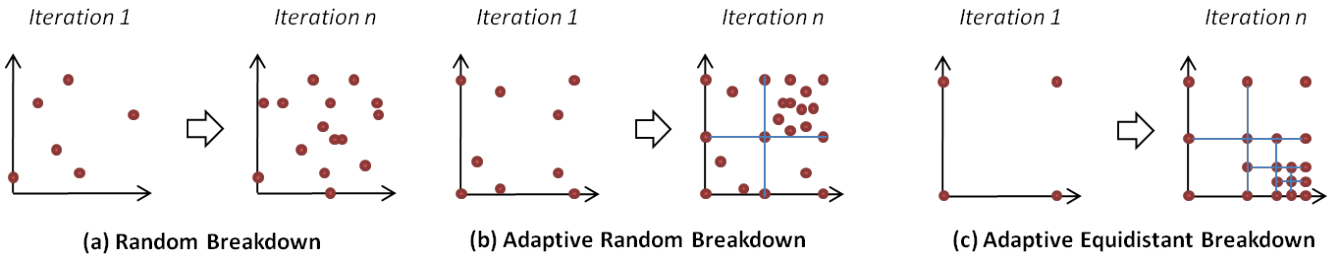

Figure 2: Measurement Point Selection Strategies

```
1: p_1 = (1, 1, ..., 1)
2: p_2 = (0, 0, ..., 0)
3: e_sector = ∞
4: M ← ∅
5: V ← ∅
6: Q ← {< p_1, p_2, e >}
7: while sizeof(Q)!=0 do
8:    T ← ∅
9:    t_tmp ← dequeue(Q)
10:   T ← T ∪ t_tmp
11:   while Q.first.e = t_tmp.e do
12:      t_tmp ← dequeue(Q)
13:      T ← T ∪ t_tmp
14:   end while
15:   for all t in T do
16:      measure all points f_edges(t.p_1, t.p_2), add results to
         M
17:      r_p ← predict_M(f_center(t.p_1, t.p_2))
18:      measure value r_m at point f_center(t.p_1, t.p_2)
19:      e_sector ← |r_m − r_p| / r_m
20:      if vs ≡ RVS ∨ e_sector > threshold then
21:         for all p_tmp in f_edges(t.p_1, t.p_2) do
22:            p_center ← f_center(t.p_1, t.p_2)
23:            t_tmp ←< p_center, p_tmp, e >
24:            enqueue(Q, t_tmp)
25:            M ← M ∪ < r_m, f_center(t.p_1, t.p_2) >
26:         end for
27:      else
28:         V ← V ∪ t
29:      end if
30:   end for
31:   for t in V do
32:      measure value r_m at point f_center(t.p_1, t.p_2).
33:      r_p ← predict_M(f_center(t.p_1, t.p_2))
34:      e ← r_m / |r_m − r_p|.
35:      if e > threshold then
36:         t.e ← e
37:         V ← V \ t
38:         Q ← Q ∪ t
39:      end if
40:   end for
41: end while
42: for all t in V do
43:   measure value r_m at point f_center(t.p_1, t.p_2)
44:   M ← M ∪ < r_m, f_center(t.p_1, t.p_2) >
45: end for
```

Line 1-6 ensure the preconditions for the actual experiment selection which starts in line 7. The primary control structure is the loop over Q starting in 7. Lines 8-14 deal with the selection of those sectors with the highest error. Starting at line 15 the loop body executes measurements (line 16) for each selected sector. Furthermore, it calculates the error for these sectors in line 17-19 and defines new sub sectors to be measured in further iterations (line 20-25) if the error is greater than the defined *threshold*. If the error is less than the *threshold* and the validation strategy is one of the Dynamic Sector strategies, the current sector is stored in V at line 28. To provide faster convergence against the underlying performance functions it brings significant advantages to execute this breadth-first approach over all elements with the same e_{sector}. This ensures that the algorithm goes deeper in the area with the highest prediction faults. Since nearly all interpolation or regression techniques cannot absolutely avoid the influence of new elements in M onto preliminary well predicted sectors, the validation repository V is checked in line 32-34 for negative effects in sectors that have been well predicted before the last modifications. For any element t in V, if the curve is not accurate enough, t is returned to Q (lines 35-39) and thus measured in more detail in later iterations. We expect that the heuristic converges more efficiently if a new measurement has only local effects on the interpolation function. Finally, lines 42-45 copy all elements from V to M as the positions where measured before and thus the data is available but not yet added to the training data of the model.

3.1.3 Adaptive Random Breakdown

As described in the previous section, the Adaptive Random Breakdown algorithm also takes the locality and the size of single point prediction errors into account when determining measurement points for the next iteration (see Figure 2 (b)). Basically, the Adaptive Random Breakdown algorithm is very similar to the Adaptive Equidistant Breakdown algorithm. The only difference is that instead of selecting only the center point of the sector, the Adaptive Random Breakdown algorithm selects a given number of random points within the sectors. Thus, we define function $f_{random} : P \times P \rightarrow P^*$ which returns n random points located in a sector s. Now, we can replace the function f_{center} in lines 16-18 in the listing in Section 3.1.2 with f_{random} in order to determine the sector error. Moreover, we replace f_{center} with f_{random} in lines 32-34 and lines 43-44 as the sector validation is also based on the n random points within a sector and not just the center point of the sector.

3.2 Validation Strategies

In this section, we describe different validation strategies that can be applied in the process shown in Figure 1.

3.2.1 Random Validation Set

In this strategy, a set of random measurement points in the parameter space is used to determine the accuracy of the prediction model. The size of the validation set can be defined by a performance analyst via the configuration meta model. In each validation run, all measured points in the validation set are compared to the predicted values of the prediction model, and the average relative error is calculated. The advantages of this strategy are that the validation points are distributed across the whole parameter space and that the performance analyst can control the size of the validation set and thus its reliability. However, the disadvantage is that a large validation set can cause long processing times of the validation step and that due to the random selection of the points we might not get enough validation points in those areas that are the most critical.

3.2.2 Dynamic Sector

The Dynamic Sector validation is a strategy that we developed in the course of developing the adaptive breakdown algorithms (see sections 3.1.2 and 3.1.3). Thus, it is closely connected to the adaptive algorithms and can only be applied in combination with one of these. The goal of the strategy is to minimize the measurement overhead for the validation step but providing enough validation points in order to confidently calculate the prediction error of the model. The strategy uses only points that have been measured anyway during the breakdown of the parameter space by the respective algorithms. After measuring a new point, the strategy decides based on the prediction error at the corresponding sector whether the new point will be part of the validation set or of the model training set. If the prediction error of a sector is below a predefined threshold, the adaptive algorithms do not further split the sector (see for example Section 3.1.3). When Dynamic Sector validation is applied, the points measured in the course of this last split will not be added to the model training set but to the validation set. After each iteration of the adaptive algorithms, the strategy checks the prediction errors of the sectors in the validation set. If a change in the model during an iteration causes the prediction error in a sector to go above the predefined threshold, the points of this sector will be removed from the validation set, added to the model set, and the sector is split in multiple sub sectors in order to execute more detailed measurements. Hence, the points that are part of the validation set change dynamically based on the sector prediction errors at a certain point in time. Advantages of this strategy are that it requires no additional measurements in order to build a validation set and that the size of the validation set grows with the number of splits executed by the adaptive algorithms. As the number of splits is an indicator for the complexity of the function that has to be predicted, we get more validation points if we have to infer a more complex function. However, the fact that only those points measured by the breakdown algorithm are used for the validation set implies that the confidence of the calculate model prediction error relies on the quality of the breakdown algorithms. The validation terminates the overall measurement process if (i) all sectors have a prediction error that is less than the predefined threshold (in the following referred to as Dynamic Sector validation with Local prediction error scope (DSL)), or (ii) the average prediction error of all sectors is less than the predefined threshold (in the following referred to as Dynamic Sector validation with Global prediction error scope (DSG)).

4. STATISTICAL MODEL INFERENCE

Statistical inference is the process of drawing conclusions by applying statistics to observations or hypotheses based on quantitative data [9]. The goal is to determine the relationship between input and output parameters observed at some system (sometimes also called independent and dependent variables). Statistical inference of performance metrics does not require specific knowledge on the internal structure of the system under study (compared to model-driven approaches such as surveyed in [14]). However, statistical inference can require assumptions on the kind of functional dependency of input and output variables. The inference approaches mainly differ in their degree of model assumptions. For example, linear regression makes rather strong assumptions on the model underlying the observations (it being linear) while the nearest neighbor estimator makes no assumptions at all. Most other statistical estimators lie between both extremes. Methods with stronger assumptions, in general, need less data to provide reliable estimates, if the assumptions are correct. Methods with less assumptions are more flexible, but require more data. For our black-box inference approach, we focus on flexible methods with less assumptions about the underlying functional dependencies. In the following, we introduce four methods that fulfill this characteristic.

4.1 Multivariate Adaptive Regression Splines (MARS)

Multivariate Adaptive Regression Splines (MARS) [7] is a non-parametric regression technique which requires no prior assumption as to the form of the data. The method fits functions creating rectangular patches where each patch is a product of linear functions (one in each dimension). MARS builds models of the form $f(x) = \sum_{i=1}^{k} c_i B_i(x)$, the model is a weighted sum of basis functions $B_i(x)$, where each c_i is a constant coefficient [7]. MARS uses expansions in piecewise linear basis functions of the form $[x-t]_+$ and $[t-x]_+$. The $+$ means positive part, so that

$$[x-t]_+ = \begin{cases} x-t & \text{, if } x > t \\ 0 & \text{, otherwise} \end{cases}$$

and

$$[t-x]_+ = \begin{cases} t-x & \text{, if } x < t \\ 0 & \text{, otherwise} \end{cases}$$

The model-building strategy is similar to stepwise linear regression, except that the basis functions are used instead of the original inputs. An independent variable translates into a series of linear segments joint together at points called knots [2]. Each segment uses a piecewise linear basis function which is constructed around a knot at the value t. The strength of MARS is that it selects the knot locations dynamically in order to optimize the goodness of fit. The coefficients c_i are estimated by minimizing the residual sum-of-squares using standard linear regression. The residual sum of squares is given by $RSS = \sum_{i=1}^{N} (\widehat{y}_i - \overline{y})^2$, where $\overline{y} = \frac{1}{N} \sum \widehat{y}_i$, where N is the number of cases in the data set and $\widehat{y}_i$ is the predicted value.

4.2 Classification and Regression Trees

Classification and Regression Trees (CART) is a simple and popular method for tree-based regression and classification. Tree-based methods partition the feature space into a set of rectangles, and then fit a simple model in each one [9]. It splits the space in two regions and models the output parameter Y by the mean of Y in each region. Then one or both of these regions are split into two more regions, and this process is continued, until a stopping rule is applied. Finding the best pair of splitting variable and split point in terms of minimum sum of squares is done via a greedy algorithm (see [9] for details). The implementation that we use in our case studies is part of the rpart package [28] of the statistic tool R.

4.3 Genetic Programming (GP)

Genetic Programming (GP) aims at deriving computer programs or mathematical equations and is thus well-suited for symbolic regression [13]. GP does not require any assumptions about the input/output parameter dependency and optimizes the structure of the equation simultaneously with the coefficients. It uses an iterative approach to approximate an optimal solution. During each iteration (*generation*), the *population*, consisting of a certain number of *individuals*, evolves. This evolution is performed by *reproducing*, *mutating* and *crossing-over* individuals of the previous generation. Each individual represents a candidate solution and has a *fitness* value expressing the quality of the solution. The aim is to maximize the fitness over many generations. The individuals in GP are usually represented as tree structures and recombinations are tree operations such as randomly exchanging subtrees between two trees. The GP algorithm used in this paper is specially optimized for the inference of performance prediction functions [6]. To improve the generalization of the result models, our GP algorithm further applies techniques to prevent overfitting. In the following we describe the our basic steps for applying GP to software performance engineering. In the first step, GP is initialized with randomized data. After the initialization, the genetic algorithm begins to evolve the individuals. The evolution starts with an evaluation of individuals by using the measurement data (Step 2). Then, the algorithm selects and reproduces fit individuals (Step 3 and 4) and repeats steps 2-4 for a given number of iterations (generations). Finally, the algorithm terminates (Step 5) when a given termination criteria, such as the desired accuracy level or runtime constraints, are fulfilled. The result of the algorithm is a performance prediction function expressed through a mathematical equation.

4.4 Kriging

Kriging is a generic name for a family of spatial interpolation techniques using generalized least-squares regression algorithms [16]. It is named after Daniel Krige who applied the method to a mineral ore body [15]. Examples of Kriging algorithms are Simple, Ordinary, Block, Indicator, or Universal Kriging. In [16], the authors provide a comprehensive review of multiple Kriging algorithms as well as other spatial interpolation techniques. Generally, the goal of spatial interpolations is to infer a spatial field at unobserved sites using observations at few selected sites. According to [16], nearly all spatial interpolation methods share the same general estimation formula:

$$\hat{Z}(x_0) = \sum_{i=1}^{n} \lambda_i Z(x_i)$$

where the estimated value of an attribute at the point of interest x_0 is represented by $\hat{Z}$, the observed value at the sampled point x_i is Z, the weight assigned to the sampled point is λ_i, and the number of sampled points used for the estimation is represented by n. Furthermore, the semivariance (γ) of Z between two data points is an important concept in geostatistics. It is defined as:

$$\gamma(x_i, x_0) = \gamma(h) = \frac{1}{2} var[Z(x_i) - Z(x_0)]$$

where h is the distance between point x_i and x_0 and $\gamma(h)$ is the semivariogram (commonly referred to as variogram)[16].

The Kriging implementation [19] that we applied in our experiments uses the Ordinary Kriging algorithm to estimate unknown points. As described above the estimated values are computed as simple linear weighted average of neighboring measured data points. The weights are determined from the fitted variogram with the condition that they must add up to 1 which is equivalent to the process of reestimating the mean value at each new location [5].

As in geostatistics the problems typically have two input parameters (the geo-coordinates), we could not find an implementation of Kriging that allows more than two input parameters. Hence, we decided to combine Kriging with Classical Multidimensional Scaling (CMDS) [3] in order to use the method for problems with more than two input variables. Using CMDS we reduce the dimensionality of the input parameter space from n to 2. The implementation [20] takes a set of dissimilarities and returns a set of points such that the distances between the points are approximately equal to the dissimilarities. We selected CMDS as although it reduces the dimensions it keeps the distances between the different points which is an essential characteristic for combining it with Kriging.

5. RELATED WORK

This section describes related work in the area of measurement based performance analysis. Various approaches explore the influence of different parameters on the performance of software applications. The authors focus on the instrumentation [4, 12] or use the results to build performance models or tests [31, 21].

Reussner et al. [21] introduce an approach to benchmark and compare different OpenMPI implementations. Their approach combines performance metrics with linear interpolation techniques to assess the implementation's overall performance behaviour. To maximise the information gain of subsequent experiments, they identify those points with the (potentially) largest error in the current prediction model. While this approach presents one of the starting points of our work, it is limited to the evaluation of a single parameter and simple linear interpolation techniques that are not suited for multidimensional scattered data. Another starting point for our work is the approach of Woodside et al. [31] and Courtois et al. [2]. They introduce a workbench for automated measurements of resource demands in dependence of configuration and input parameters. The results are fitted by different statistical methods resulting in so-called resource functions that capture performance metrics

with respect to the given parameters. However, the authors did not compare different experiment selection methodologies or different analysis methods. Moreover, they did not apply the approach to parameter spaces with more than two independent parameters.

In [12], Jung et al. introduce an approach for the automatic instrumentation of applications called Mulini that is based on AOP and code generation techniques. They weave non-functional specifications into staging implementations in order to explore large configuration parameter spaces. They apply their approach to bottleneck detection of a reference application called RUBiS. Their tool Mulini automatically monitors, collects, and analyses a significant number of performance metrics in iterative staging executions. While the approach of Jung et al. allows the collection of large amounts of data and the evaluation of the influence of different parameters, the authors neither perform any further analysis on the collected data (such as symbolic regression or machine learning techniques) nor do they optimise the number of measurements required.

Denaro et al. [4] propose an approach for early performance testing of distributed applications. Their core assumption (similar to Gorton et al. [8]) is that the middleware is the determining factor of an application's performance. However, the usage of middleware features (like transaction or persistence) is determined by the application. Therefore, Denaro et al. use architecture designs to derive performance test cases that can be executed and used to estimate the applications performance in the target environment. Gorton et al. also conduct measurements in the target environment but use the results to calibrate a prediction model which is then used to predict the application's performance. Both approaches do not explicitly evaluate the influence of parameters of configurations. The measurements are focused on specific scenarios. While this is sufficient for the author's purposes, it is not enough to capture the influence of different configurations and input parameters on performance.

6. CASE STUDIES

In this section, we introduce the case studies that we use to evaluate the combinations of measurement point selection algorithms and statistical model inference methods. In the case studies, we automated the execution of standard industry benchmarks (i.e. SPECjjbb2005 and SAP SD Benchmark) using our Software Performance Cockpit framework in order to derive performance prediction functions in different realistic scenarios. The following subsections are structured as follows. After an introduction to the case study, we list the five best and worst performing *combinations of measurement point selection algorithm, validation strategy and model inference technique (Comb)*. Moreover, we briefly comment the results. A detailed evaluation and discussion of the results is then provided in Section 7. The result tables provide the following metrics:

The *used measurement points (MP)* compared to the possible number of measurement points spanned by the parameter space. As we aim at generating an accurate prediction model with the least possible number of measurements, the number of measurements that a combination of measurement point selection, validation strategy and analysis technique requires to create its model is one of the most important criteria.

The *mean relative error (MRE)* of the predictions (in %). To derive this metric, we used a random set of measurement points within the parameter space. This validation set is independent of the training and validation sets used during the derivation of the prediction models. The mean relative prediction error is also used to define the target threshold for the validation strategies and thus for determining when to terminate the algorithm.

The mean relative error alone can sometimes cause misleading conclusions. For example, in cases where a large (simple) part of a function is fitted very well, the mean relative error can be under a certain threshold although there might be an important area where the predictions are bad. That is why we also use the metrics LT15, LT30, and *Highest Error (HE)* as an indicator for the reliability of the predictions. The first two metrics define *the percentage of predictions that have a prediction error that is less than 15% (LT15) or 30% (LT30)*, respectively. The HE is the highest single point prediction error (in %) observed in the validation.

The order within the best 5 and the worst 5 entries in the tables is based on a combined consideration of these five metrics. If two combinations did not differ strongly in the prediction error, we ranked those better that required less measurements. However, the actual weighting of the metrics can differ depending on the scenario and the goal of the performance evaluation. Although it is not our most important criteria, in the evaluation section we also roughly look at the time it takes an analysis technique to create the prediction model, i.e. without the time for measuring the data.

6.1 Enterprise Application Customization

In this case study, we address the problem of customizing an SAP ERP application to an expected customer workload. The workload of an enterprise application can be coarsely divided into batch workload (background jobs like monthly business reports) and dialogue workload (user interactions like displaying customer orders). This workload is dispatched by the application server to separate operating system processes, called work processes, which serve the requests [24]. At deployment time of an SAP system the IT administrator has to allocate the available number of work processes (depending on the size of the machine) to batch and dialogue jobs, respectively. With the performance prediction function derived in this case study, we enable IT administrators to find the optimal amount of work processes required to handle the dialogue workload with the constraint that the average response time of dialogue steps should be less than one second. The system under test consists of the enterprise resource planning application SAP ERP2005 SR1, an SAP Netweaver application server and a MaxDB database (version 7.6.04-07). The underlying operating system is Linux 2.6.24-27-xen. The system is deployed on a single-core virtual machine (2,6 GHz, 1024KB cache). To generate load on the system we used the SAP Sales and Distribution (SD) Benchmark. This standard benchmark covers a sell-from-stock scenario, which includes the creation of a customer order with five line items and the corresponding delivery with subsequent goods movement and invoicing. Each benchmark user has his or her own master data, such as material, vendor, or customer master data to avoid data-locking situations [23]. The de-

pendent variable is the average response time of dialogue steps ($AvgResponseTime$). The independent variables in this setup are (i) the number of active users ($NumUser$) where the domain ranges from 60 to 150 and (ii) the number of work processes for dialogue workload ($NumWP$) varied from 3 to 6. Thus, we are looking for the function $f(NumUser, NumWP) = AvgResponseTime$. The full parameter space consists of 360 measurement points. The determination of a single measurement point (including repeated measurements to control measurement noise) takes approximately one hour which means that in the worst case the IT administrator has to measure 15 days in order to determine the optimal configuration. As stated earlier, we do not aim at modelling the complete ERP system and varying all potential configuration, workload and tuning parameters of a system at once. Instead, the goal is to provide a practical automated evaluation that helps the administrator to determine the optimal allocation of work process for a given workload type and a given system configuration. In the process of enterprise application customization this is only one question among many others which is why it is important to provide a flexible, automated approach that does not make assumptions about underlying functional dependencies. Table 1 shows the five best and worst performing combinations of our prediction approach.

Table 1: Results for Enterprise Application Customization

Top 5					
Comb	MP	MRE	HE	LT15	LT30
AEB DSG GP	21	8.7	36.0	81.8	98.7
AEB DSG Kriging	38	6.0	43.3	88.3	96.1
ARB DSG MARS	38	7.3	31.8	89.6	98.7
AEB DSG MARS	53	7.4	31.7	87.0	98.7
AEB DSL Kriging	54	2.8	38.8	94.8	98.7
Worst 5					
Comb	MP	MRE	HE	LT15	LT30
AEB RVS CART	69	31.7	92.9	26.0	51.3
ARB RVS CART	77	28.7	92.0	35.1	57.9
ARB DSG CART	77	28.7	92.0	35.1	57.9
ARB DSL CART	77	28.7	92.0	35.1	57.9
RB RVS CART	77	28.7	92.0	35.1	57.9

Even the worst combination can derive a prediction model with an acceptable prediction error while requiring only one fourth of the measurement points. For the combinations that performed best the result is even better. For the combination of AEB, DSG and GP we were able to build a prediction model with an average relative prediction error of 8.7% using only 21 measurement points. The Kriging method in combination with AEB and DSG also performed very good with a relative prediction error of only 6% and 38 required measurement points. Thus, applying our approach can reduce the time necessary to derive an optimal configuration from 15 to one or two days of measurement. Here, one can see that although we varied only two independent parameters it is essential to provide efficient evaluation methods in order to derive results in a reasonable time frame.

6.2 Java Virtual Machine Tuning

The Java Virtual Machine (JVM) is one of the most important components when it comes to performance tuning of a Java-based applications [11, 25]. However, getting the best performance out of the JVM often requires detailed manual tuning of command line options wrt. heap sizes or garbage collection. In this case study, we address the problem of tuning the parameters of a JVM to the special characteristics of an application. The application that we use in our experiments is the SPECjbb2005 Java Server Benchmark [26]. The benchmark emulates a three-tier client/server system (with emphasis on the middle tier) and exercises the implementations of the JVM, JIT (Just-In-Time) compiler, garbage collection, threads as well as some aspects of the operating system [26]. The system modelled by the benchmark is a wholesale company, with warehouses that serve a number of districts. Customers initiate a set of operations, such as placing new orders or requesting the status of an existing order. Additional operations are generated within the company, such as processing orders for delivery or entering customer payments [26]. The system under test consist of the SPECjbb2005 benchmark (configured to run with 10 warehouses), Java HotSpot(TM) Client VM (build 17.0-b17), and Microsoft Windows XP Professional Version 2002 SP3. The software runs on a standard desktop dual-core machine with 3 GHz per CPU and 3.5 GB RAM. The dependent variable in this scenario is the average throughput of a benchmark run ($AvgThroughput$) measured in SPECjbb2005 bops (business operations per second). The independent variables are as follows (see [18] for a detailed description of the parameters): (i) the heap size ($HeapSize$) where we configured the possible variation from 300 MB to 950 MB in steps of 25 MB, (ii) the garbage collector ($GarbageCollector$) implementation which is either SerialGC, ParallelGC, or ConcMarkSweepGC, (iii) a boolean value that indicates whether biased locking ($BiasedLocking$) is enabled, (iv) the survivor ratio ($SurvivorRatio$) varied from 10 to 42 in steps of 8, and (v) the new generation ratio ($NewGenerationRatio$) which is expressed in a share of the total heap size ranging from 10% to 40% and varied in steps of 10%. Thus, we are looking for the function $f(HeapSize, GarbageCollector, BiasedLocking, SurvivorRatio, NewGenerationRatio) = AvgThroughput$. Again, we are not interested in modelling the complete system but focus on a specific component and within this component only on a subset of parameters (from which we assume that they are performance-relevant). The goal is to help the administrator in solving a specific practical problem by automating time-consuming tasks. The full parameter space consists of 3240 measurement points. In this scenario, the determination of a single measurement point takes approximately 5 minutes. Table 2 outlines the five best and worst performing combinations for this scenario.

The results in Table 2 show that this case study was the most complex in terms of the black-box inference of a performance prediction function. Even the best combinations have a highest prediction error (HE) of 300 to 400 percent. However, the overall error as well as the efficiency of the prediction models built by the first three combinations is still acceptable, which demonstrates the robustness of these combinations. One reason for the complexity of this scenario is that we included an enumeration variable ($GarbageCollector$) and a boolean variable ($BiasedLocking$) where we do not necessarily have monotonically increasing values which makes prediction harder for most of the statistical analyses techniques. Moreover, the large highest error

Table 2: Results for JVM Tuning Scenario

Top 5					
Comb	MP	MRE	HE	LT15	LT30
RB RVS MARS	276	20.7	403.1	77.1	86.7
AEB RVS MARS	342	20.3	301.4	73.6	87.0
RB RVS Kriging	365	25.3	955.1	73.7	86.9
AEB DSG MARS	1076	16.3	259.8	79.1	88.0
AEB DSL MARS	1325	17.3	287.9	79.8	88.0
Worst 5					
Comb	MP	MRE	HE	LT15	LT30
ARB DSG Kriging	1001	73.0	964.0	46.7	65.0
ARB DSL Kriging	1011	76.3	957.8	42.6	62.4
RB RVS GP	1388	26.9	485.3	47.3	74.9
ARB DSL MARS	2027	23.9	384.4	70.2	85.3
AEB RVS CART	3111	26.4	432.5	68.8	82.3

values are an indicator that the granularity that we selected for the parameter variations was not fine-grained enough. Obviously, there are areas in the parameter space where we did not have enough information in order to build an accurate model. However, for this experiments we had to limit the parameter space to 3240 measurement points as we had to measure the full space upfront in order to compare the different strategies and validate the results. In a real application scenario one would chose one of the efficient combinations and define the parameter variations more fine-grained which should lead to more accurate results while not necessarily requiring much more measurements.

7. EVALUATION AND DISCUSSION

In this section, we discuss the results of the case studies presented in Section 6. We start by evaluating the four statistical model inference techniques in isolation and then summarize the results.

Classification and Regression Tree (CART) is a very fast method that built all the prediction models in the case studies in milliseconds. However, in the real case studies the results have been very bad, especially with respect to the reliability of the predictions. According to our experiments, CART works best in combination with Adaptive Equidistant Breakdown (AEB) or Random Breakdown (RB) measurement point selection and Random Validation Set (RVS) validation. It does not work very well with the Dynamic Sector (DS) validation strategies.

Genetic Programming (GP) achieved very good results in fitting the function in the enterprise application customization scenario. However, it was not able to efficiently derive a prediction function in the JVM tuning scenario. The best results have been achieved in combination with AEB measurement point selection and DSG or RVS validation, respectively. It did not work very well with the combination Adaptive Random Breakdown (ARB) and RVS. The biggest problem of the GP approach is its runtime. In average, it took the approach approximately 20 minutes to build a prediction model which adds up to a large amount of analysis time when using it in our iterative process (Figure 1).

Kriging is in terms of runtime somewhere in the middle between CART and GP. It becomes slower with increasing number of measurement points which is mainly caused by the classical multidimensional scaling (CMDS) implementation that we run before the actual prediction model is built using the Kriging implementation (see Section 4.4). In general, the results of the JVM tuning scenario have shown that our approach with the CMDS in combination with Kriging is working and able to derive accurate prediction models. The best results could be achieved in the enterprise application customization scenario, where we varied only two input variables and thus the dimension reduction step has not been executed. In this scenario, Kriging has been the most efficient method. Like GP, it worked best with the combinations AEB/DSG and AEB/RVS and delivered the worst results with ARB measurement point selection.

Multivariate Adaptive Regression Splines (MARS) is the only method that achieved very good results in all case studies. From a runtime perspective MARS was also able to build prediction models within seconds (at least with the size of the training data in our scenarios). It worked most efficient in combination with AEB measurement point selection and DSG validation. Good results have also been achieved with the combinations AEB/RVS and RB/RVS. The worst results with ARB measurement point selection.

In summary, MARS together with AEB measurement point selection and DSG validation has been the only combination that achieved very good results in all case studies. Only for the enterprise application customization case study, GP and especially Kriging performed slightly better (but also in combination with AEB/DSG). CART turned out to be the worst method, and is based on our experiences not suited for black-box inference of performance prediction functions. Kriging and GP are in general able to fit black-box models and can be good alternatives to MARS. Especially, if there is only one or two input parameters but a large number of measurement points Kriging can be an efficient option. The main problem with GP is the time it takes to create a prediction model which makes it not the perfect option for an iterative approach with repeated generation of prediction models. Regarding the measurement point selection algorithms and validation strategies there is a clear tendency that AEB is the most efficient algorithm that provides especially in combination with DSG and RVS validation the best results independent of the analysis method. The prediction models derived by the simple RB are in most cases very accurate and reliable. However, compared to AEB it required in most cases more measurement points to build the model.

8. SUMMARY

Knowing the functional dependencies between performance relevant system parameters (such as configuration or workload parameters) and performance metrics of interest allows software architects, developers, and system administrators to better design, implement, size or configure a system. In this paper, we proposed an automated approach to inference multidimensional performance prediction functions using a fully automated, iterative process. To the best of our knowledge, we are the first who systematically applied and evaluated (i) different strategies for automatically selecting new measurement points after each iteration, (ii) different validation strategies that allow us to automatically decide when to terminate the measurements, as well as (iii) different statistical model inference methodologies that make less assumptions about the underlying functional dependencies. In general, the best results have been achieved by the combination Adaptive Equidistant Breakdown (AEB) measure-

ment point selection, Dynamic Sector validation with Global prediction error (DSG), and MARS model inference. The industrial case studies have shown that our approach allows software engineers to automatically build prediction models with a mean relative prediction error of less than 20% using only up to 10% of the possible measurement points. The approach of applying the geostatistical interpolation method Kriging to the performance prediction area has been successful, although that, as excepted, the predictions for two-dimensional problems have been more accurate than those for n-dimensional problems that have been transformed into two-dimensional problems using classical multidimensional scaling.

For the future, we plan to investigate why different combinations of methods worked better in one scenario than the other. This could lead to a classification of scenarios based on certain characteristics, which would then serve as heuristics in choosing the best combinations of methods to be applied to a given application context. In order to further automate our approach, we are developing a procedure that automatically selects those system parameters that should be included in a prediction function using a combination of techniques such as experimental designs (e.g., Plackett-Burman designs) and correlation analyses (e.g., Pearson's correlation coefficient).

9. REFERENCES

[1] Software Performance Cockpit. http://www.sopeco.org. last visited: July 2012.

[2] M. Courtois and M. Woodside. Using regression splines for software performance analysis and software characterization. In *Proc. of Workshop on Software and Performance (WOSP)*, NY., 2000. ACM.

[3] T. F. Cox and M. A. A. Cox. *Multidimensional Scaling, Second Edition*. Chapman & Hall/CRC, 2000.

[4] G. Denaro, A. Polini, and W. Emmerich. Early performance testing of distributed software applications. *SIGSOFT Software Engineering Notes*, 29(1):94–103, 2004.

[5] M. J. DeSmith, M. F. Goodchild, and P. A. Longley. *Geospatial Analysis: A Comprehensive Guide to Principles, Techniques, Software Tools*. Troubador.

[6] M. Faber and J. Happe. Systematic Adoption of Genetic Programming for Deriving Software Performance Curves. In *Proceedings of ICPE 2012*, pages 33–44, April 2012.

[7] J. H. Friedman. Multivariate adaptive regression splines. *Annals of Statistics*, 19(1):1–141, 1991.

[8] I. Gorton and A. Liu. Performance Evaluation of Alternative Component Architectures for Enterprise JavaBean Applications. *IEEE Internet Computing*, 7(3):18–23, 2003.

[9] T. Hastie, R. Tibshirani, and J. Friedman. *The Elements of Statistical Learning: Data mining, Inference, and Prediction*. Springer Series in Statistics. Springer, 2nd edition, 2009.

[10] R. Jain. *The art of computer systems performance analysis*. Wiley Interscience, New York, 1991.

[11] L. Jamen. Oracle Fusion Middleware - Performance and Tuning Guide, 2011.

[12] G. Jung, C. Pu, and G. Swint. Mulini: An Automated Staging Framework for QoS of Distributed Multi-Tier Applications. In *ASE Workshop on Automating Service Quality*, 2007.

[13] J. R. Koza, editor. *Genetic programming*, volume On the programming of computers by means of natural selections. MIT Press, Cambridge, Mass., 1993.

[14] H. Koziolek. Performance evaluation of component-based software systems: A survey. *Performance Evaluation Journal*, 2009.

[15] D. G. Krige. A Statistical Approach to Some Basic Mine Valuation Problems on the Witwatersrand. *Journal of the Chemical, Metallurgical and Mining Society of South Africa*, 52(6):119–139, Dec. 1951.

[16] J. Li and A. D. Heap. *A review of spatial interpolation methods for environmental scientists*. Geoscience Australia, Canberra, 2008.

[17] NewRelic. New relic. http://newrelic.com, 2012.

[18] Oracle. Tuning garbage collection with the 5.0 java tm virtual machine. http://www.oracle.com/technetwork/java/gc-tuning-5-138395.html, 2011.

[19] E. J. Pebesma. Multivariable geostatistics in s: the gstat package. *Computers and Geosciences*, 2004.

[20] R Dev Team. Classical multidimensional scaling. http://stat.ethz.ch/R-manual/R-patched/library/stats/html/cmdscale.html, 2011.

[21] R. Reussner, P. Sanders, L. Prechelt, and M. Mueller. SKaMPI: A detailed, accurate MPI benchmark. *Lecture Notes in Computer Science*, 1497:52–??, 1998.

[22] J. Sankarasetty et al. Software performance in the real world: personal lessons from the performance trauma team. In V. Cortellessa, S. Uchitel, and D. Yankelevich, editors, *WOSP*. ACM, 2007.

[23] SAP. Standard Application Benchmarks. http://www.sap.com/solutions/benchmark, March 2011.

[24] T. Schneider. *SAP Performance Optimization Guide: Analyzing and Tuning SAP Systems*. Galileo, 2006.

[25] J. Shirazi. *Java performance tuning - efficient and effective tuning strategies*. O'Reilly, 2003.

[26] SPEC. SPECjbb2005 - Industry-standard server-side Java benchmark (J2SE 5.0). Standard Performance Evaluation Corporation, June 2005.

[27] E. Thereska, B. Doebel, A. X. Zheng, and P. Nobel. Practical performance models for complex, popular applications. In *SIGMETRICS*, pages 1–12, 2010.

[28] T. M. Therneau and B. Atkinson. Cran-package rpart, 2012. http://cran.r-project.org/web/packages/rpart.

[29] D. Westermann, J. Happe, M. Hauck, and C. Heupel. The Performance Cockpit Approach: A Framework for Systematic Performance Evaluations. In *Proceedings of the 36th EUROMICRO SEAA 2010*. IEEE CS, 2010.

[30] D. Westermann, R. Krebs, and J. Happe. Efficient Experiment Selection in Automated Software Performance evaluations. In *Proceedings of EPEW 2011*, Heidelberg, 2011. Springer-Verlag.

[31] C. M. Woodside, V. Vetland, M. Courtois, and S. Bayarov. Resource function capture for performance aspects of software components and sub-systems. In *Performance Engineering, State of the Art and Current Trends*, pages 239–256, UK, 2001. Springer.