Efficient Experiment Selection in Automated Software Performance Evaluations

Dennis Westermann[1], Rouven Krebs[1], Jens Happe[1]

SAP Research, Karlsruhe, Germany
{dennis.westermann|rouven.krebs|jens.happe}@sap.com

Abstract. The performance of today's enterprise applications is influenced by a variety of parameters across different layers. Thus, evaluating the performance of such systems is a time and resource consuming process. The amount of possible parameter combinations and configurations requires many experiments in order to derive meaningful conclusions. Although many tools for automated performance testing are available, controlling experiments and analyzing results still requires large manual effort. In this paper, we apply statistical model inference techniques, namely Kriging and MARS, in order to adaptively select experiments. Our approach automatically selects and conducts experiments based on the accuracy observed for the models inferred from the currently available data. We validated the approach using an industrial ERP scenario. The results demonstrate that we can automatically infer a prediction model with a mean relative error of 1.6% using only 18% of the measurement points in the configuration space.

1 Introduction

Performance engineering is a crucial discipline throughout development and hosting of enterprise applications. However, the sheer size and complexity of software systems and development processes hinders the application of performance engineering in many cases. Especially in large enterprise applications, the performance of a system is affected by a variety of parameters. Understanding their influences (and capturing them in a performance model) requires a huge number of experiments and "what-if" analyses in order to draw meaningful conclusions.

State-of-the-art 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 [3, 9, 12, 20]. 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 [26, 28]. In both cases, the amount of possible parameter combinations and configurations makes the measurement process time and resource consuming. While 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).

In this paper, we present a fully automated approach that (i) selects and conducts experiments, (ii) uses statistical inference techniques to derive a prediction model based on the measured data, (iii) validates the prediction model, and (iv) iteratively determines new experiments that maximise the information gain and thus increase the accuracy of the model. The statistical inference techniques that we use in our experiments are Multivariate Adaptive Regression Splines (MARS) [7] and Kriging [29]. MARS has already been succesfully applied for software performance analyses [3, 6, 10]. Kriging is a geostatistical interpolation technique that has been applied to various research areas dealing with spatial data. However, to the best of our knowledge Sacks et al. [25] are the only ones that applied Kriging to analyze data measured in computer system experiments. The strength of both methods is that they provide robust predictions and do not require any prior knowledge about the underlying dependencies in the data (e.g., in contrast to simple linear regression).

The contributions of this paper are (a) the description and comparison of three automated experiment selection methodologies for the efficient derivation of statistical performance prediction models and (b) the application of the Kriging interpolation technique for software performance analyses.

We validate our approach in two case studies. The results demonstrate that adaptive experiment selection can yield accurate prediction models with a significantly reduced amount of measurements. Moreover, we show that the geostatistical interpolation technique Kriging can be applied for the analysis of performance measurements. In fact, Kriging outperforms MARS for some problem classes.

The remainder of this paper is organized as follows. Section 2 gives an overview of our ongoing research and brings this paper into line with our overall motivation. In Section 3, we discuss related research approaches. Section 4 provides basics of statistical model inference using MARS and Kriging. In Section 5, we describe the three experiment selection algorithms that we apply in our approach. A real-world case study as well as detailed validation results are illustrated in Section 6. Finally, Section 7 concludes the paper.

2 Motivation and Overview

In this section, we give an overview of our overall approach for performance predictions of enterprise applications. Applying Software Performance Engineering (SPE) in practice is still a challenging task. In most cases, software vendors built their applications on a large basis of existing components such as middleware, legacy applications, or third-party services. Software architects are facing questions like "How does middleware A affect the performance of my application?", or "Will the application under development meet the performance requirements?". A software as a service provider wants to know, for example, "What happens to the performance of my system if I double the amount of underlying virtual machines?" or "What happens to performance if the number of users increases?". Existing model-driven performance engineering approaches mainly realise a pure top down prediction approach. Software architects have to provide a complete model of their system in order to conduct performance analyses. Measurement-based performance evaluations, by contrast, depend on the availability of the application and can only be applied in late development cycles. Our approach aims at integrating model-driven and measurement-based performance predictions in order to build practical performance models of enterprise applications.

2.1 Software Performance Curves

The main idea of our approach is to apply goal-oriented, systematic measurements to already existing parts of a system. The result of the systematic measurements is a quantification of the dependencies between the system's usage (workload and parameters) and performance (timing behavior, throughput, and resource utilization). We refer to the statistical models describing these dependencies as software performance curves. Formally, a performance curve describes the performance $\mathcal{P}$ (response time, throughput, and resource utilisation) of a system in dependence on a set of input parameters $A_1, \ldots, A_n$ with $n \in \mathbb{N}$. It is a function $f\colon A_1 \times A_2 \times \ldots \times A_n \to \mathbb{R}$, where each input parameter A_i is a number ($\subset \mathbb{R}$), an enumeration, or a boolean value. The function's result represents the performance metric of interest. The benefit of these statistically inferred models is that they do not require specific knowledge on the internal structure of the system under study (e.g., in contrast to other approaches that use statistical inference to estimate parameters of queuing networks [15, 17, 22]). Thus, the software performance curves are a black-box description of the performance behaviour of the system under test. The derivation of the performance curves requires the execution of many measurements in different settings. Therefore, we developed a framework called Software Performance Cockpit [33, 32] that encapsulates best practices and allows for separation of concerns regarding the different aspects of a performance evaluation. Using this framework we can automatically control measurements, trigger analyses and export results [34].

2.2 Integrating Software Performance Curves and Model-driven Performance Analyses

In order to use the software performance curves to decide on design alternatives, plan capacities, or identify performance critical system configurations we propose to integrate the curves with model-driven performance engineering approaches (such as surveyed in [1] and [14]). Figure 1 illustrates the approach.

For those parts of a system that are under development we apply an existing approach for model-driven performance engineering [2]. Software architects specify the system's components, behaviour, deployment, and usage (*System Modelling*). This activity results in a *System Model* that describes the newly developed parts as well as its usage. In order to consider the effect of existing parts in a performance analysis, we need to include them in the prediction model. The *Measurement*s described above result in *Performance Data* of the system. Such data can be used for *Model Inference*. The resulting software performance curves consider the effect of system external parts on performance, these models have to be integrated with or made available in model-driven prediction approaches (*Integration*). This step merges both model types and creates a common basis for further performance analysis (*Prediction*). Based on the *Performance Predictions*, software architects and performance analysts can decide about design alternatives, plan capacities, or identify critical components.

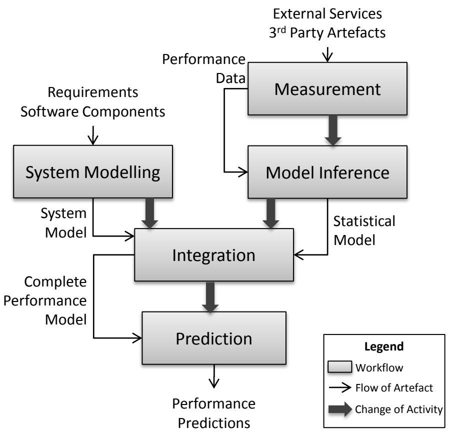


Fig. 1. Overview of integrating model-driven and measurement-based performance analysis.

In this paper we focus on the *Measurement* and *Model Inference* parts. In our future work, we will describe how performance curves can be integrated with the Palladio Component Model (PCM) [2].

3 Related Work

In this section, we present 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 itself [5, 13, 19] or use the results to build performance models (or tests) [35, 24, 30, 12] or detect errors [19, 20].

Reussner et al. [24] 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 multi-dimensional scattered data. Another starting point for our work is the approach of Woodside et al. [35] and Courtois et al. [3]. 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.

Denaro et al. [5] 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.

In [12], Jin et al. introduce an approach called BMM that combines **B**enchmarking, production system **M**onitoring, and performance **M**odelling. Their goal is to quantify the performance characteristics of real-world legacy systems under various load conditions. However, the measurements are driven by the upfront selection of a performance model (e.g layered queuing network) which is later on built based on the measurement results.

Miller et al. [19] propose Paradyn, a tool for the automatic diagnoses of performance problems. They apply dynamic instrumentation to control the instrumentation in search of performance problems. Paradyn starts looking for high-level problems for a whole application and, once the general problem is found, inserts further instrumentations to find more specific causes. Miller et al. focus on the detection of performance problems and do not measure parameter spaces systematically.

4 Statistical Model Inference

Statistical model inference is the process of learning from data [11]. A variety of methodologies have been developed in statistical science [11, 18, 21] in order to extract patterns and trends from data or to fit curves to the data. In this paper, we focus on so called supervised learning problems [11] where the goal is to predict the value of an observed metric based on a number of input parameters. The different statistical methods have their own characteristics that mainly differ in their degree of model assumptions. For example, linear regression makes rather strong assumptions on the model underlying the observations (they are 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. In the course of this paper, we apply and compare two different methodologies, namely MARS and Kriging. Both methods are able to deal with the assumption that we have less or no knowledge about the structure of the data. MARS has already been successfully applied in software performance prediction [3, 6, 10]. Geostatistical interpolation methods, such as Kriging, are designed to analyse irregularly spaced set of data points in a three dimensional space [29]. Characteristics of geostatistical data are (i) high costs to get the value of interest for these points and (ii) that near measurements are more interrelated to each other then distant ones [31]. We assume that these characteristics are also true for measured performance data. Measuring a single configuration of an enterprise application often requires extensive effort. Moreover, in most cases a minor change in a configuration has less effect on the performance metric of interest than larger changes. Furthermore, the adaptive experiment selection method presented in Section 5.3 creates an irregularly spaced set of data points. For this reasons, we decided to investigate the use of Kriging to derive software performance curves. In the following, we briefly introduce the two methods.

4.1 Kriging

Kriging is a generic name for a family of spatial interpolation techniques using generalised least-squares regression algorithms [18]. It is named after Daniel Krige who applied the method to a mineral ore body [16]. Examples of Kriging algorithms are Simple, Ordinary, Block, Indicator, or Universal Kriging. In [18], 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 [18], 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)[18].

Figure 2 shows an example variogram with an exponential variogram model. The *nugget* (or nugget effect) is a contribution to variability without spatial continuity [29]. The *range* is the distance where the model first flattens out and the *sill* is the value at which the variogram model reaches the range.

The Kriging implementation [23] 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 neighbouring 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 [4].

4.2 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

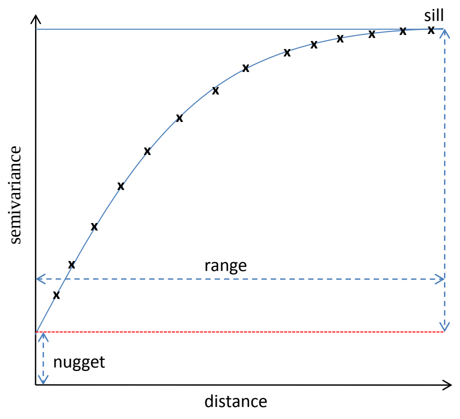


Fig. 2. Sample Variogram

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

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

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 [3]. 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.

5 Experiment Selection

In order to automatically derive a software performance curve with the least possible number of measurements, we need an iterative algorithm that (i) selects new experiments for each iteration, (ii) infers a statistical model based on the available data after each iteration, and (iii) is aware of the quality of the inferred model. In the context of this paper, an experiment (or configuration point) is defined as one configuration of all parameters (i.e., it corresponds to one point in the configuration space). The configuration space is spanned by the configuration parameters and their corresponding domains. In this section, we present three experiment selection methodologies that fulfill the requirements mentioned above by applying different strategies (see Figure 3).

The random experiment selection strategy randomly selects a fixed number of new experiments. The equidistant experiment selection strategy splits the parameter space in equidistant areas. The adaptive experiment selection strategy selects new experiments in those areas of the parameter space that show the worst predictions. Each of the three methodologies can be combined with various model inference techniques.

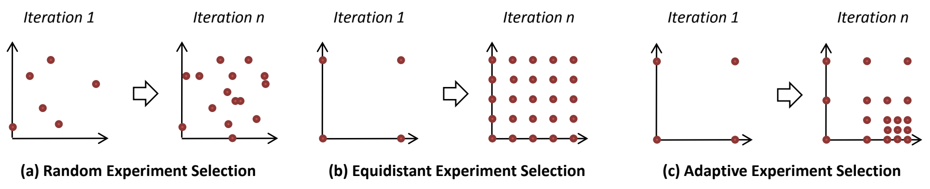

Iteration 1
Iteration n
Iteration 1
Iteration n
Iteration 1
Iteration n
(a) Random Experiment Selection
(b) Equidistant Experiment Selection
(c) Adaptive Experiment Selection

Fig. 3. Experiment Selection Methodologies

5.1 Random Experiment Selection based on Global Prediction Error

The first algorithm randomly selects new experiments in order to minimize the global prediction error. In each iteration a fixed number of n randomly selected configuration points are measured (see Figure 3 (a)). Based on a set of validation points (measured before the first iteration starts), we calculate the mean relative error (MRE) of the prediction model. The algorithm terminates when the error is below a predefined threshold, a predefined number of measurements has been reached, or a predefined measurement time has expired.

5.2 Equidistant Experiment Selection based on Global Prediction Error

The second algorithm determines the measurement points for the next iteration by stepwise equidistant splitting of the configuration space (see Figure 3 (b)). We define $P = \{x | x \in \mathbb{R}^i \wedge \forall x_i \in [0..1]\}$ as a set of all possible positions in the configuration space with normalized values. An element $p \in P$ describes one experiment or configuration point. Let the elements $p_1, p_2 \in P$ be two opposing positions that describe the multidimensional configuration space. Function $f_{center} : P \times P \to P$ returns the center of the two given points which is calculated by the element-wise arithmetic middle of the two vectors. Furthermore, function $f_{edges} : P \times P \to P^*$ returns the set of all edges of the embraced space defined by two configuration points. The function computes all possible element-wise combinations of the two given configuration points. We use $H_{i,n,z}$ as a set which helps to find the experiments for each iteration. H contains a set of tuples describing areas of the space to be measured, therefore every element in H consists two opposing positions. The process itself does continuously divide the space into equidistant areas based on previously computed areas. Thereby, the measurement iteration is expressed by the index i. The index n is used to iterate over the tuples stored in H. The third index $z \in [1, 2]$ defines which of the two positions stored in every element of H is referenced. Formally, the set of experiments to be executed is computed as follows:

$$H_0 = f_{edges}(p_1, p_2) \times f_{center}(p_1, p_2)$$

$$H_i = \bigcup_{n=1}^{|H_{i-1}|} f_{edges}(H_{i-1,n,1}, H_{i-1,n,2}) \times f_{center}(H_{i-1,n,1}, H_{i-1,n,2})$$

$$Experiments_i = (\bigcup_{n=1}^{|H_i|} H_{i,n,1} \cup H_{i,n,2}) \setminus Experiments_{i-1}$$

The experiments that will be demanded in the next iteration include all the points within the tuples of H that have not yet been measured.

The termination criteria as well as the validation procedure are the same as those defined for the random selection methodology (see 5.1).

5.3 Adaptive Experiment Selection based on Local Prediction Error

In contrast to the algorithms described in the previous sections, this algorithm takes the locality and the size of single prediction errors into account when determining experiments for the next iteration (see Figure 3 (c)). We assume that a new experiment at the area with the highest prediction error raises the accuracy of the overall model at most. Another difference to the previous experiment selection methodologies is that it does not include all determined points for an iteration in the training data, but uses a subset of these points for validation. Thus, this methodology does not require the creation of a validation set before the actual iteration starts. 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. As in Section 5.2, we define $P = \{x | x \in \mathbb{R}^i \wedge \forall x_i \in [0..1]\}$ as a set of all possible positions in the configuration space with normalized values. Elements of P are declared as p. The elements p_1 and p_2 are opposing positions necessary to describe a multidimensional space. Function $f_{center} : P \times P \to P$ returns the center of the two given points which is calculated by the element-wise arithmetic middle of the two vectors. Furthermore, function $f_{edges} : P \times P \to P^*$ returns a set of all edges of the embraced space given by p_1 and p_2.

In addition, let $e \in \mathbb{R}^+$ describe the error of the performance curve at a defined area or position and $S = \{p_1 \times p_2 \times e | p_1 \in P \wedge p_2 \in P \wedge e \in \mathbb{R}^+\}$. Three subsets of S control the measurement progress. A priority-controlled queue $Q \subset S$ contains tuples describing areas in the configuration space, where the error of the curve ran out of the acceptable threshold. The order of priority is based on e. The collection $V \subset S$ is the validation set which contains all the tuples describing areas where a good prediction has already been observed. $M \subset S$ is the training set which contains the measurement results used to create a performance curve. All subsets of S are mutually disjoint and it holds that $S = Q \cup V \cup M$. The function $predict_M : P \to \mathbb{R}$ creates a prediction results based on the given measurements M for a specific configuration 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.

```
1: $p_1 = (1, 1, \ldots, 1)$
2: $p_2 = (0, 0, \ldots, 0)$
3: $e = \infty$
4: $Q \leftarrow \{< p_1, p_2, e >\}$
5: **while** sizeof(Q)!=0 **do**
6:    $T \leftarrow \emptyset$
7:    $t_{tmp} \leftarrow dequeue(Q)$
8:    $T \leftarrow T \cup t_{tmp}$
9:    **while** $Q.first.e = t_{tmp}.e$ **do**
10:      $t_{tmp} \leftarrow dequeue(Q)$
11:      $T \leftarrow T \cup t_{tmp}$
12:   **end while**
13:   **for all** $t$ in $T$ **do**
14:      measure all points $f_{edges}(t.p_1, t.p_2)$, add results to $M$
15:      measure value $r_m$ at point $f_{center}(t.p_1, t.p_2)$
16:      $r_p \leftarrow predict_M(f_{center}(t.p_1, t.p_2))$
17:      $e \leftarrow \frac{r_m}{|r_m - r_p|}$
18:      **if** $e > threshold$ **then**
19:         **for all** $p_{tmp}$ in $f_{edges}(t.p_1, t.p_2)$ **do**
20:            $p_1 \leftarrow f_{center}(t.p_1, t.p_2)$
21:            $t_{tmp} \leftarrow < p_1, p_{tmp}, e >$
22:            enqueue($Q$, $t_{tmp}$)
23:            $M \leftarrow M \cup < r_m, f_{center}(t.p_1, t.p_2) >$
24:         **end for**
25:      **else**
26:         $V \leftarrow V \cup t$
27:      **end if**
28:   **end for**
29:   **for** $t$ in $V$ **do**
30:      measure value $r_m$ at point $f_{center}(t.p_1, t.p_2)$.
31:      $r_p \leftarrow predict_M(f_{center}(t.p_1, t.p_2))$
32:      $e \leftarrow \frac{r_m}{|r_m - r_p|}$.
33:      **if** $e$ > threshold **then**
34:         $t.e \leftarrow e$
35:         $V \leftarrow V \setminus t$
36:         $Q \leftarrow Q \cup t$
37:      **end if**
38:   **end for**
39: **end while**
40: **for all** $t$ in $V$ **do**
41:    measure value $r_m$ at point $f_{center}(t.p_1, t.p_2)$
42:    $M \leftarrow M \cup < r_m, f_{center}(t.p_1, t.p_2) >$
43: **end for**
```

Line 1-4 ensure the preconditions for the actual experiment selection which starts in line 5. The primary control structure is the loop over Q starting in 5. Lines 6-12 deal with the selection of all elements with highest error. Starting at line 13 the loop body executes measurements (line 14) in the area of each selected tuple. Furthermore, it calculates the error for these areas in line 15-17 and defines new subareas to be measured in further iterations (line 18-25) if the error is greater than the defined $threshold$. If the error is less than the $threshold$ the current tuple is stored in V at line 26. 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. This ensures to step down 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 areas, the validation repository V is checked in line 30-32 for negative effects in areas that have been well predicted before the last modifications. If for any element in V the curve is still not accurate enough it is returned to Q at line 33-37 and thus measured in more detail in later iterations. We expect that the heuristic converges more efficient if a new measurement has only local effects onto the interpolation function. Finally, line 40-43 copies 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.

6 Case Study and Validation

In this section, we demonstrate the efficiency of the approach and the accuracy of the inferred prediction models. Moreover, we apply the software performance curves in a "real-world" scenario using a large enterprise application. For the evaluation we formulate the following research questions:

- **RQ1:** To which extent are the experiment selection methodologies presented in Section 5 more efficient (in terms of necessary measurements to create an accurate prediction model) (i) compared to measuring the full configuration space and (ii) compared to other approaches?
- **RQ2:** Are geostatistical interpolation techniques applicable in software performance analysis scenarios? Are they more efficient compared to multivariate regression?
- **RQ3:** Is the approach applicable to automatically create measurement-based performance models of large enterprise applications in a reasonable amount of time?

In the remainder of this section we present two case studies and a discussion of the results. Figure 4 summarizes the results of the two case studies. The table contrasts the different combinations of experiment selection method and analysis method. To determine the quality of the derived prediction model we compared the prediction for each measurement point in the configuration space with its actual value and calculated the mean relative error (MRE).

6.1 Communication Server Case Study

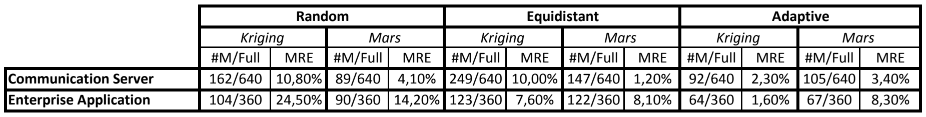
In this case study we applied our approach to a scenario described by Courtois and Woodside [3]. The authors applied a similar adaptive experiment selection approach combined with MARS to derive resource functions for a unicast-based multicast communications server. The basic components of the server are (i) a Supplier Handler that reads, packages, and enqueues incoming messages from the Supplier processes and (ii) a Consumer Router which dequeues a message and sends it to each of its Consumer processes (see [3] for details). The authors derived the following resource function for the Consumer Router component:

$$\textit{Consumer-Router-CPU} = 1436.73 + 0.1314 * h(msgsize - 1)$$
$$-0.0159 * h(-(consumers - 9)) * h(msgsize - 1)$$
$$+808.082 * h(consumers - 9) - 149.399 * h(-(consumers - 9))$$
$$-0.03 * h(consumers - 9) * h(-(msgsize - 21091))$$
$$-0.0092 * h(consumers - 1) * h(-(msgsize - 10808))$$
$$+0.0989 * h(msgsize - 4223) - 0.01 * h(-(consumers - 9)) * h(msgsize - 5424)$$

The domain of message sizes was set between 1 and 64K bytes and the number of consumers varied from 1 to 10. Thus, the full configuration space consists of 640 measurement points. For our case study, we tried to fit this function using the same domains for the two parameters. The results (see Figure 4) show that for this case study the combinations RandomSelection/MARS with 89 measurement points (#M) and an average prediction error of 4.1% and AdaptiveSelection/Kriging with 92 measurement points and an error of 2.3% performed best. Thus, the approaches required only 14% of the full configuration space to create a very good prediction model. Compared to the approach presented in [3] which required 157 measurements to build the prediction model (with an error of 8.58%) we saved 65 measurements (i.e., 41%). However, when comparing the two approaches one has to note that we fitted the simulated function and had not to deal with the real measurement data which might cause additional prediction error.

6.2 Enterprise Application Case Study

The goal of this cast study is to demonstrate that the approach is applicable on real data measured on a large enterprise application. 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 dialog 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 [28]. 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 dialog jobs, respectively. With the performance curve derived in this case study, we enable IT administrators to find the optimal amount of work processes required to handle the dialog workload with the constraint that the average response time of dialog 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 [27]. The dependent variable is the average response time of dialog 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 dialog workload ($NumWP$) varied from 3 to 6. Thus, we are looking for the function $f(NumUser, NumWP) = AvgResponseTime$. The full configuration space consists of 360 measurement points. In order to get statistically stable results we repeated each measurement multiple times. All in all, the determination of a single measurement point 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. The results (see Figure 4) show that our adaptive experiment selection methodologies provides very good results with both analysis methods. The combination AdaptiveSelection/Kriging required only 64 measurement points ($\approx$18% of the full configuration space) to derive a prediction model with a mean relative error of 1.6%. This reduces the time necessary to derive an optimal configuration from 15 to $\approx$2.5 days.



	Random				Equidistant				Adaptive			
	Kriging		*Mars*		*Kriging*		*Mars*		*Kriging*		*Mars*	
	#M/Full	MRE	#M/Full	MRE	#M/Full	MRE	#M/Full	MRE	#M/Full	MRE	#M/Full	MRE
Communication Server	162/640	10,80%	89/640	4,10%	249/640	10,00%	147/640	1,20%	92/640	2,30%	105/640	3,40%
Enterprise Application	104/360	24,50%	90/360	14,20%	123/360	7,60%	122/360	8,10%	64/360	1,60%	67/360	8,30%

Fig. 4. Case Study Results

6.3 Discussion

The results of the two case studies (see Figure 4) show that the approach presented in this paper can significantly reduce the effort necessary to derive measurement-based performance models with high prediction accuracy. In both cases, our approach was able to derive a very good prediction model using only $\approx$15% of the full configuration space (**RQ1 (i)**). Even the comparison with a similar approach proofed the efficiency of our methodology (**RQ1 (ii)**). The equidistant experiment selection strategy generated the worst results independent of the analysis strategy. The random strategy achieved good results especially in combination with MARS. However, the best results have been achieved by the adaptive experiment selection strategy independent of the analysis strategy. But, the Kriging predictions outperformed MARS in both scenarios which proofs the applicability of geostatistical interpolation techniques for software performance analyses (**RQ2**). The combination AdaptiveSelection/Kriging reduced the time necessary to find the optimal configuration of an enterprise application server from 15 days to $\approx$2.5 days which is an import reduction due to the fact that the time for the configuration of a system in the staging phase is often very limited (**RQ3**).

7 Summary

In this paper, we presented an approach for the automated and efficient selection of experiments in order to derive performance prediction models. We introduced three experiment selection methodologies and combined them with the statistical model inference techniques MARS and Kriging. Moreover, we applied the approach in two case studies and compared the different combinations of experiment selection and analysis methods. In these case studies, the combination of adaptive experiment selection and Kriging achieved the best results. The proposed techniques support software architects and performance analysts to capture the effect of existing software systems on software performance and include these effects into further performance evaluations. In our future work, we will investigate further experiment selection strategies and analysis methods. We will address the curse of dimensionality [11] by applying the approach in case studies with more than two independent parameters. Furthermore, we are going to integrate the measurement-based performance models with model-driven approaches as described in Section 2.

References

1. S. Balsamo, A. Di Marco, P. Inverardi, and M. Simeoni. Model-Based Performance Prediction in Software Development: A Survey. *IEEE Transactions on Software Engineering*, 30(5):295–310, May 2004.
2. S. Becker, H. Koziolek, and R. Reussner. The Palladio component model for model-driven performance prediction. *Journal of Systems and Software*, 82:3–22, 2009.
3. M. Courtois and M. Woodside. Using regression splines for software performance analysis and software characterization. In *Proceedings of the 2nd International Workshop on Software and Performance (WOSP-00)*, pages 105–114, N. Y., Sept. 17–20 2000. ACM Press.
4. M. J. De Smith, M. F. Goodchild, and P. A. Longley. *Geospatial Analysis: A Comprehensive Guide to Principles, Techniques and Software Tools*. Troubador Publishing.
5. G. Denaro, A. Polini, and W. Emmerich. Early performance testing of distributed software applications. *SIGSOFT Software Engineering Notes*, 29(1):94–103, 2004.
6. F. A. H. D. O. H. Eskenazi, E.M. Performance prediction for software architectures. In *Proceedings of PROGRESS 2002 Workshop*, 2002.
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 forEnterprise JavaBean Applications. *IEEE Internet Computing*, 7(3):18–23, 2003.
9. H. Groenda. Certification of software component performance specifications. In *Proceedings of Workshop on Component-Oriented Programming (WCOP) 2009*, pages 13–21, 2009.
10. J. Happe, D. Westermann, K. Sachs, and L. Kapova. Statistical inference of software performance models for parametric performance completions. In *Proceedings of QoSA 2010)*.
11. 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.
12. Y. Jin, A. Tang, J. Han, and Y. Liu. Performance evaluation and prediction for legacy information systems. In *Proceedings of ICSE 2007*, pages 540–549, Washington, DC, USA, 2007. IEEE CS.
13. 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.
14. H. Koziolek. Performance evaluation of component-based software systems: A survey. *Performance Evaluation*, In Press, Corrected Proof, 2009.
15. S. Kraft, S. Pacheco-Sanchez, G. Casale, and S. Dawson. Estimating service resource consumption from response time measurements. In *Proc. of VALUETOOLS 2009*, NY, USA, 2009. ACM.

16. 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.
17. D. Kumar, L. Zhang, and A. Tantawi. Enhanced inferencing: Estimation of a workload dependent performance model. In *Proceedings of VALUETOOLS 2009*, 2009.
18. J. Li and A. D. Heap. *A review of spatial interpolation methods for environmental scientists*. Geoscience Australia, Canberra, 2008.
19. B. P. Miller, M. D. Callaghan, J. M. Cargille, J. K. Hollingsworth, R. B. Irvin, K. L. Karavanic, K. Kunchithapadam, and T. Newhall. The paradyn parallel performance measurement tool. *Computer*, 28:37–46, November 1995.
20. A. Mos and J. Murphy. A framework for performance monitoring, modelling and prediction of component oriented distributed systems. In *WOSP '02: Proc. of the 3rd international workshop on Software and performance*, pages 235–236, New York, NY, USA, 2002. ACM.
21. H. J. Motulsky and L. A. Ransnas. Fitting curves to data using nonlinear regression: a practical and non-mathematical review. 1987.
22. G. Pacifici, W. Segmuller, M. Spreitzer, and A. Tantawi. Dynamic estimation of cpu demand of web traffic. In *Proc. of VALUETOOLS 2006*, page 26, New York, NY, USA, 2006. ACM.
23. E. J. Pebesma. Multivariable geostatistics in s: the gstat package. *Computers and Geosciences*, 30:683–691, 2004.
24. R. Reussner, P. Sanders, L. Prechelt, and M. Mueller. SKaMPI: A detailed, accurate MPI benchmark. *Lecture Notes in Computer Science*, 1497:52–??, 1998.
25. J. Sacks, W. J. Welch, T. J. Mitchell, and H. P. Wynn. Design and analysis of computer experiments. *Statistical Science*, 4:409–423, 1989.
26. J. Sankarasetty, K. Mobley, L. Foster, T. Hammer, and T. Calderone. Software performance in the real world: personal lessons from the performance trauma team. In V. Cortellessa, S. Uchitel, and D. Yankelevich, editors, *WOSP*, pages 201–208. ACM, 2007.
27. SAP. SAP Standard Application Benchmarks. http://www.sap.com/solutions/benchmark, March 2011.
28. T. Schneider. *SAP Performance Optimization Guide: Analyzing and Tuning SAP Systems*. Galileo Pr Inc, 2006.
29. P. Switzer. *Kriging*. John Wiley and Sons, Ltd, 2006.
30. D. Thakkar, A. E. Hassan, G. Hamann, and P. Flora. A framework for measurement based performance modeling. In *WOSP '08: Proceedings of the 7th international workshop on Software and performance*, pages 55–66, New York, NY, USA, 2008. ACM.
31. W. Tobler. A computer movie simulating urban growth in the detroit region. *Economic Geography*, 46(2):234–240, 1970.
32. D. Westermann and J. Happe. Performance Cockpit: Systematic Measurements and Analyses. In *ICPE'11: Proceedings of the 2nd ACM/SPEC International Conference on Performance Engineering*, New York, NY, USA, 2011. ACM.
33. D. Westermann and J. Happe. Software Performance Cockpit. http://www.softwareperformancecockpit.org/, March 2011.
34. 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.
35. 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, London, UK, 2001. Springer-Verlag.