Self-Tuning Resource Demand Estimation

Johannes Grohmann*, Nikolas Herbst*, Simon Spinner* and Samuel Kounev*
*University of Würzburg, Email:[first_name].[last_name]@uni-wuerzburg.de

***Abstract*—The average time a resource needs to process incoming requests in a monitored workload mix is a key parameter of stochastic performance models. Direct measurement of these resource demands is usually infeasible due to instrumentation overheads causing measurement interferences and perturbation in production environments.**

Thus, a number of statistical estimation approaches (e.g., based on optimization, regression or Kalman filters) have been proposed in the literature each coming with different strengths and run-time overheads. Most approaches offer parameters in order to customize the behavior of the estimator influencing the estimation quality and the required computation time. However, their configuration usually requires exhaustive testing, as default parameters normally do not provide optimal performance.

In this paper, we propose a self-tuning approach based on discrete optimization that can be used to automatically tune the parameters of resource demand estimation methods, tailoring them to the specific application scenario and thus improving their accuracy. We apply and compare different techniques on a representative data set with varying load levels and number of workload classes. We show that our selected approach for parameter tuning can automatically improve the estimation quality of certain estimators by up to 25%.

I. Introduction

Performance models can be used to answer performance-related questions for a software system during system design, capacity planning at deployment time or during system operation. Different performance modeling formalisms exist, e.g., stochastic performance models such as Queueing Networks (QN) [1] or Queueing Petri Nets (QPN) [2], or architecture-level performance models such as the Palladio Component Model (PCM) [3] or the Descartes Modeling Language (DML) [4].

A key parameter of stochastic performance models are *resource demands* (also known as service demands). A resource demand is the average time a unit of work (e.g., request or transaction) spends obtaining service from a resource (e.g., CPU or hard disk) in a system over all visits excluding any waiting times [5], [6]. Different requests can be grouped into different *workload classes*. Requests from the same workload class normally have similar resource demands.

Timely and precise resource demand estimates are a crucial input to proactive auto-scaling mechanisms used for elastic resource provisioning. Besides the standard threshold-based auto-scalers, more sophisticated auto-scaling mechanisms (that leverage queueing theory, control theory or time series analysis) require as input, in addition to average CPU-utilization measurements, the resource demands of the different workload classes [7].

However, the direct measurement of resource demands is not feasible during operation in most realistic systems [8] due to instrumentation overheads and possibly measurement interferences. Furthermore, Willneker et al. [9] showed that statistical estimation approaches can provide a comparable accuracy to direct measurements. We therefore focus on statistical approaches to resource demand estimation.

The advantage of estimation approaches compared to direct measurement techniques is their general applicability and low overheads. Estimation approaches typically rely only on coarse-grained measurements from the system (e.g., CPU utilization, and end-to-end average response times) which can be monitored easily with state-of-the-art tools without the need for fine-grained code instrumentation. These measurements are routinely collected for many applications (e.g., in data centers). Therefore, approaches to resource demand estimation are also applicable on systems serving production workloads.

Over the years, a number of approaches to resource demand estimation have been proposed using different statistical estimation techniques (e.g., linear regression [10], [11] or Kalman filters [12], [13]) and based on different laws from queueing theory. When selecting an appropriate approach for a given scenario, a user has to consider different characteristics of the estimation approach, such as the expected input parameters, its accuracy and its robustness to measurement anomalies. Depending on the constraints of the application context, only a subset of the estimation approaches may be applicable.

To the best of our knowledge, the only publicly available tool providing different ready-to-use approaches to resource demand estimation is LibReDE (Library for Resource Demand Estimation) [14].[1] The latter currently supports implementations of the following estimation approaches:

- Service Demand Law [11]
- Least Squares (LSQ) regression using queue-lengths and response times (Response Time Regression) [15]
- LSQ regression using utilization law (Utilization Regression) [10]
- Approximation with response times (Response Time Approximation) [11]
- Recursive optimization using response times (Menascé Optimization) [16]
- Recursive optimization using response times and utilization (Liu Optimization) [17]
- Kalman Filter (KF) using utilization law (Wang Kalman Filter) [12], [18]
- KF using response times and utilization (Kumar Kalman Filter) [13], [19]

[1]LibReDE: Available for download at http://descartes.tools/librede.

LibReDE implements features, such as configuring the *Step Size* or the *Window Size* allowing to refine the input processing for all approaches. Additionally, some approaches like the ones based on Kalman Filters (KFs) offer specific parameters in order to customize their behavior for optimal performance. However, it is not trivial to determine how the various parameters should be configured for optimal performance. A poor choice of parameter values can drastically decrease the estimation accuracy of some approaches, while others may remain relatively unaffected. However, finding sufficient or good configurations for a given scenario requires exhaustive testing and/or expert knowledge and is usually not feasible by hand. Therefore, most approaches offer standard parameter configurations based on experience and/or rules of thumb.

However, in case sample measurement traces from the given application domain are available, we can use them in order to optimize the parameter settings and adapt the estimators for the specific domain. Therefore, we propose a self-tuning approach based on discrete optimization that can be used to automatically tune the parameters of resource demand estimation methods, tailoring them to the specific application scenario and thus improving their accuracy.

This self-tuning approach automatically optimizes the parameter settings of the different resource demand estimation techniques mentioned above without requiring any prior knowledge from the user. We extend LibReDE to autonomically optimize generic parameters of black-box estimation approaches during system operation. The evaluation is conducted with representative measurement traces from exhaustive micro-benchmark experiments on a real system. Due to the lack of space, we limit ourselves to the most interesting parameters, the ones based on KFs. Tweaking those can improve the estimation accuracy by up to 25%.

However, note that the proposed approach is of a general nature and supports all available approaches for resource demand estimation. Although not mentioned in this work, we also applied some different algorithms on all of the above mentioned estimators. More detailed evaluations can be found in the work of Grohmann [20].

The remainder of the paper is structured as follows. In Section II, we describe the related work. Section III describes our idea and our approach introducing the parameters to be tuned in Section III-A and the algorithm for parameter self-tuning in Section III-B. We evaluate our approach in Section IV and summarize and conclude our work in Section V.

II. Related Work

There are several works on the evaluation of resource demand estimation approaches. However, most of them only cover one or two approaches.

Rolia and Vetland [10], [21] first did some experiments for Linear Regression (LR) techniques. Pacifini et al. [22], Casale et al. [23], [24] and Stewart et al. [25] extend this by investigating limitations of LR in resource demand estimation and the impact of different factors. The performance of KFs for resource demand estimation is researched by Zheng et al. [13], [26] and Kumar et al. [19]. Kraft et al. [15] and Sharma et al. [27] both compare LSQ regression with their Maximum Likelihood Estimation (MLE) [15] and Independent Component Analysis (ICA) [27] approach, respectively.

Spinner et al. [8] integrated many different approaches into one common implementation and evaluated them. The publicly available tool LibReDE [14] offers open source implementations of currently eight different estimators.

However, the user still has to manually configure the different approaches when using LibReDE. There exist no works on automatic and systematic evaluation of the best parameter settings for a given test set, since previous approaches only do manual testing and develop rules of thumb for a chosen small set of parameters (see [8], [13], [19], [23], [24], [26]).

III. Approach

The general idea of this work is to combine the resource demand estimation approaches with general optimization techniques executed in an automated manner in order to improve the performance of existing resource demand estimation. To this end, we use a set of training traces on which the configuration parameters of different approaches can be optimized in dependence of different input features. These traces usually contain standard information like the average resource utilization for a certain period of time or the arrival and response time per request (broken down according to the different workload classes). They can be collected using historical data from the System Under Study (SUS) or other similar systems.

Since most estimation approaches have multiple configuration parameters, we can try to optimize their result by tuning these parameters. By studying scenarios with known resource demands as a reference or by performing cross-validation, we can tune the parameters in order to minimize the estimation error. In the case of cross-validation, the estimation error can be calculated using the utilization error E_U or the response time error E_R, as defined in Section IV-A. Depending on the considered use case, either one or the other or a combination of both error values can be optimized. The calculated error values are used to rate the quality of each parameter configuration and the one with minimal errors is preferred.

We a heuristic to systematically search and find satisfactory parameter configurations specialized on the given training set. Depending on the size of the training set and the complexity of the algorithm, this usually leads to a very high computational overhead. Nevertheless, this can significantly improve the performance of the estimators, since it is adapting specially to the required target scenario.

Furthermore, in the context of DevOps, the deployment and/or the software version might be subject to change. Additionally, the workload mix or the load level are generally variable. All of these factors influence the resource demands and/or the optimal parameter configuration of the resource demand estimators. Therefore, the self-tuning process can be periodically repeated in order to re-tune the parameters based on historic data. Depending on the frequency of expected changes and the computational demand of the optimization, this re-tuning might be applied more or less often.

A. *Configurable Parameters*

Generally, any parameters can be tuned by our self-tuning approach. All parameters should define a minimum and a maximum value, as our proposed algorithm needs that in order to work. Additionally, a default value should be set, both as a guideline for the algorithm and as a baseline configuration to estimate the achieved gain.

As we want to focus on Kalman Filter (KF) specific parameters in this work, we give a short explanation of the two considered approaches and the configuration options. We use the notation defined by Spinner et al [8].

KFs are a generally applicable statistical estimation method capable of estimating the hidden state of a dynamic system. Zheng et al. [13] and Kumar et al. [19] apply KFs for resource demand estimation, where the hidden states to be estimated are the resource demands of the system. The approach uses the utilization of the resources, the response time and the arrival rate of requests to build the models. We refer to this approach as Kumar Kalman Filter (KKF).

Wang et al. [12], [18] propose a different approach based on the utilization law, which we also include in this work and referring to it as Wang Kalman Filter (WKF).

For lack of space, we omit any further details on both approaches and refer to the original papers or the works of Spinner et al. [8] or Grohmann [20] for elaborate descriptions and implementation details.

The implementation in LibReDE requires five parameters: Three parameters are based on the nature of KFs and parameterize the state model and the noise covariances. We call them *state noise covariance*, *state noise coupling* and *observe noise covariance*.

The other two parameters *bounds factor* and *initial bounds distance* have to do with the bounds of the estimates. They are necessary, as the general description of KFs allows any kind of state variables, but resource demands can be restricted to certain interval (e.g., just postive quantities). They are used to control estimates, when the actual estimates of the KFs get out of bounds [13].

However, due to the abstract nature of our optimization algorithms, no deeper understanding of the parameters is required.

B. *Algorithm for Parameter Self-Tuning*

The proposed self-tuning algorithm is generally abstract and works for any generic parameter providing a minimum and a maximum value. Optionally, a reasonable start value can be defined.

The Stepwise Sampling Search (S3) or Iterative Parameter Optimizer (IPO) [28] was developed as part of our previous work in the context of regression model optimization [29]. Here, we use a version of this algorithm adapted for self-tuning the parameters of resource demand estimation techniques. However, due to the lack of space, we can not go into detail

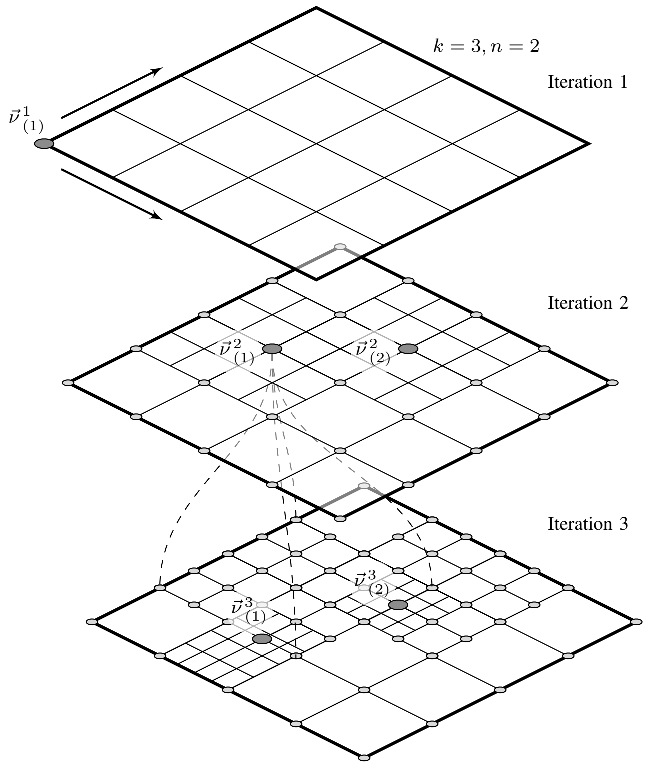

Figure 1. The first three iterations of S3 [28].

about the algorithm and refer to the original source for more detail [28].

Each evaluation of a point in the parameter space normally takes a considerably long time (around 10 - 30 seconds, depending on the used estimation algorithm and the size of the training set). Hence, we want to keep the number of points to evaluate as low as possible. Multiple dimensions, i.e., multiple different parameters and their interactions are covered by the algorithm although this results in higher computational demand.

S3 can be configured by three parameters: The number of splits per parameter k, the number of *exploration points* considered per iteration n and the maximum number of iterations j_{max}.

Figure 1 illustrates an example for the first three iterations of S3. Depicted is an example with two parameters, a splitting factor k of three and two exploration points n. Starting at the first exploration point $\vec{\nu}^1_{(1)}$ (top layer), 25 *evaluation points* (including the initial value) are created. Every crossing of the lines represents one evaluation point. i.e. a parameter combination to be evaluated. All evaluated points are illustrated by a small circle at the corresponding crossing.

For the second iteration, let's assume the marked points $\vec{\nu}^2_{(1)}$ and $\vec{\nu}^2_{(2)}$ are the most promising of the evaluated points. They are therefore chosen as *exploration* points with the next iteration considering a new set of points between their lower and their upper neighbour. As shown in the picture, the sets do not necessarily need to be disjoint.

From the new set of evaluated points, the new exploration points are determined, which explore their own subspace and so forth until the maximum number of iterations j_{max} is reached. In our example $j_{max} = 3$ and the execution returns the best point found so far after $\vec{\nu}^3_{(1)}$ and $\vec{\nu}^3_{(2)}$ finish their exploration.

Noorshams et al. [28] show that the total complexity of the algorithm is given by $\mathcal{O}(j_{max} \cdot n \cdot (k+2)^l)$. Hence, it can be controlled by tuning the parameters. However, it has to be noted that the number of parameters to be optimized simultaneously heavily influences the computational complexity. Therefore, the number of parameters should be kept at a reasonable size. The other parameters can be tuned as desired in order to face the trade-off between run-time and solution quality.

Note that S3 requires another set of parameters in order to configure its behavior due to the generic nature of the algorithm. We will define and set default values after our analysis in Section IV. This prevents the user from being faced with another set of parameters.

Additionally, S3 does not depend on initial values and is hence not negatively influenced by a poor choice of default values. However, our self-tuning algorithms always evaluate the error for the default values just in case. If a parameter is found to perform better with its default value rather than the computed tuned value, the default value is chosen.

IV. Experimental Analysis

In this section, we evaluate the impact of applying the presented self-tuning algorithm for optimizing the parameters of resource demand estimation techniques. We compare the estimation error when using default values with the estimation error after self-tuning on a representative data set. We distinguish between the utilization error E_U and the response time error E_R as defined in the following section, since different use cases stress both errors differently.

A. Setup

Our training set consists of measurements obtained from running micro-benchmarks on a real system. The micro-benchmarks generate a closed workload with exponentially distributed think times and resource demands. As mean values for the resource demands, we selected 14 different subsets of the base set [0.02s; 0.25s; 0.5s; 0.125s; 0.13s] with number of workload classes $C = \{1; 2; 3\}$. The subsets were arbitrarily chosen from the base set so that the resource demands are not linearly growing across workload classes. The subsets intentionally also contained cases where two or three workload classes had the same mean value as resource demand. The mean think times were determined according to the targeted load level of an experiment. We varied the number of workload classes $C = \{1; 2; 3\}$ and the load level $U = \{20\%; 50\%; 80\%\}$ between experiments. The traces were already used by Spinner et al. [8], where a more detailed description of the test environment can be found.

In total, 210 traces of approximately one hour run time were collected: They were randomly split into training and validation set, resulting in a total of 168 training traces and 42 validation traces. The optimizations were performed on a virtual machine hosted on a XenServer hypervisor equipped with an Intel® Xeon® E5-2640 v3 (6 Cores) @ 2,6 GHz processor and 32 GB of RAM running Windows 10.

For error analysis we separate between the relative response time error E_R and the absolute utilization error E_U shown in Equation 1:

$$\begin{aligned} E_R &= \frac{1}{C}\sum_{c=1}^{C}\left|\frac{\tilde{R}_c - R_c}{R_c}\right|, \\ E_U &= \left|\sum_{c=1}^{C}(X_c \cdot \tilde{D}_c) - U\right| \end{aligned} \tag{1}$$

with C being the number of workload classes, R_c the average measured response time of workload class c over all resources, $\tilde{R}_c$ the predicted average response time based on the estimated resource demands, X_c the measured throughput of workload class c, $\tilde{D}_c$ the estimated resource demand of workload class c and U the average measured utilization over all resources.

B. Results

Table I shows the five configurable parameters for KFs with their default value as well as with the minimum and maximum values chosen by us.

Table I
PARAMETERS FOR KALMAN FILTERS (KFS) WITH DEFAULT VALUE, LOWER AND UPPER BOUNDS.

Parameter name	Lower bound	Upper bound	Default
Initial bounds distance	0.0	0.1	0.0001
Bounds factor	0.0	1.0	0.9
State noise covariance	0.0	2.0	1.0
Observe noise covariance	0.0	0.1	0.0001
State noise coupling	0.0	2.0	1.0

Table II shows the performance of different runs of S3 optimizing the KKF in terms of response time error. We used three configurations of S3: The first one tunes all parameters with one split point per parameter, three exploration points per iteration, and three iterations in total. (5.1-3-3)

As tuning all parameters at once takes a lot of time, we run two alternative approaches splitting the parameters into two groups: The parameters concerning the bounds, i.e., the initial bounds distance and the bounds factor, and the parameters dealing with the coupling and covariances, i.e., the state noise covariance, the state noise coupling, and the observe noise covariance.

We use two different configurations to solve them. The second (2.4-5-5) configuration denotes the S3 run optimizing the initial bounds distance and the bounds factor. It uses four splits per evaluation and five exploration points over five iterations. The third (3.3-3-3) iteration optimized the coupling and covariance parameters and runs with $k = 3$, $n = 3$ and

$j_{max} = 3$. Note that, we performed a lot more tests than shown here, in order to pick the shown configurations. We aimed for roughly comparable runtimes in order to keep the results comparable.

Table II
COMPARISON OF DIFFERENT CONFIGURED RUNS OF S3 FOR THE KKF OPTIMIZING RESPONSE TIME. THE BOLD PARAMETERS MARK THE VALUES THAT HAVE BEEN OPTIMIZED.

Algorithm	Default	5.1-3-3	2.4-5-5	3.3-3-3
Opt. time (hh:mm:ss)	–	05:05:45	04:05:08	03:17:44
Avg. est. time (ms)	141.571	119.238	181.595	139.238
Rel. improvement	–	15.775%	−28.271%	1.648%
Avg. E_R	0.229	0.177	0.171	0.176
Rel. improvement	–	22.760%	25.257%	22.954%
Avg. E_U	0.034	0.040	0.034	0.039
Init. bounds dist.	0.0001	**0.0**	**0.1**	0.0001
Bounds factor	0.9	**1.0**	**1.0**	0.9
State noise cov.	1.0	**0.0**	1.0	**0.0**
Observe noise cov.	0.0001	**0.1**	0.0001	**0.1**
State noise coupl.	1.0	**0.0**	1.0	**0.0**

We see the benefit of self-tuning in Table II. All configurations of S3 found significant improvements of over 20%. However, the runs themselves do not differ as much as one might expect. Interestingly, the 2.4-5-5 optimizing only the initial bounds and the bounds factor of KKF performs best with an improvement of over 25%. This is due to the fact that a focus on less parameters reduces the computational complexity allowing to consider more iterations and evaluation points. If we assume independent parameters, we should always split up the parameters and optimize them separately. This matches our expectations from Section III-B. However, 5.1-3-3 still performs surprisingly well, although we had to prune its search space to only 1 split per parameter ($k = 1$) in order to keep the tuning time acceptable.

Table III
COMPARISON OF DIFFERENT CONFIGURED RUNS OF S3 FOR THE WKF OPTIMIZING RESPONSE TIME. THE BOLD PARAMETERS MARK THE VALUES THAT HAVE BEEN OPTIMIZED.

Algorithm	Default	5.1-3-3	2.4-5-5	3.3-3-3
Opt. time (hh:mm:ss)	–	04:50:06	02:57:20	02:50:32
Avg. est. time (ms)	155.310	111.143	75.571	161.190
Rel. improvement	–	28.438%	51.341%	−3.787%
Avg. E_R	1.056	1.030	1.030	1.034
Rel. improvement	–	2.423%	2.404%	2.042%
Avg. E_U	0.021	0.021	0.021	0.021
Init. bounds dist.	0.0001	**0.1**	**0.1**	0.0001
Bounds factor	0.9	**1.0**	**1.0**	0.9
State noise cov.	1.0	**0.75**	1.0	**0.125**
Observe noise cov.	0.0001	**0.0**	0.0001	**0.0015625**
State noise coupling	1.0	**0.75**	1.0	**0.125**

Table III shows even less variance than Table II. Although all algorithms find different values for the given parameters, they all manage to achieve a gain between two and three percent in terms of estimation error. Even though this gain is not quite as notable as for KKF in Table II, we still note that the given default configurations are far from optimal, since all algorithms can find a comparable improvement.

Additionally, we added the average absolute utilization error E_U as metric to Tables II and III in order to monitor the change in E_U although we are optimizing E_R. We can see in Table II that the bounds factor does not seem to influence E_U very much. Hence, we can optimize the bounds with 2.4-5-5 and achieve a gain of over 25% for E_R, while non-negatively influencing E_U. The other two approaches show an increase of E_R. However, we argue that this impact is still acceptable, while optimizing E_R.

Table III shows no negative impact at all. We can therefore tune E_R of WKF without major influences on E_U.

Note that while the estimation error is deterministic for a fixed test set, the repeatabilty in terms of estimation time is generally quite low in our experiments, as the execution times varied in each experiment. Therefore, despite the promising results in Table III, we cannot say that the average estimation time is significantly affected by the parameter tuning, given that the variance between individual runs is bigger than the variance of the different configurations. Furthermore, estimation times are not in the focus of this work as we aim to optimize the estimation error and are therefore do not consider the estimation time in the self-tuning algorithms. However, we plan to extend our work in this direction in the future.

We repeated the same experiments optimizing the utilization error E_U. Although results and optimization times were similar, the overall gain was below 1% for all runs. This is due to the fact that the utilization error is already quite small for both KFs. We therefore do not cover the results and just note that all runs behaved comparable, matching our expectations from the observations above.

C. Limitations

We included only one individual optimization run in this work. We repeated the experiments several times with shuffled training and validation data sets in order to validate their repeatability. The relative improvements in the estimation error varied not more than by 10% over three repetitions. However, the random shuffling of training and validation data set made it impractical to add different runs to our evaluation.

We compared the results of the S3 algorithm with other heuristics like local-search or brute-force. Due to the lack of space, we are not able to include this in our work, but like to note that the results proved S3 to be optimal in terms of solution gain vs optimization time.

We ignore estimation times for analysis and optimization in this work. However, we plan to tackle this problem in future work.

V. CONCLUSION

We proposed an approach to automatically tune the parameters of resource demand estimation approaches. In the evaluation, we focused on the step size as well as five specific parameters. Our test set consisted of 210 representative measurement traces from exhaustive micro-benchmark experiments on a real system. The results show optimization of the Kalman Filter specific parameters can decrease the response time error of the Kumar Kalman Filter by over 25%.

As the Kalman Filter are generally more efficient (around 150ms on avg.) than comparable estimators (see e.g., [8], [20]), their ability to be automatically fine-tuned to a given scenario with significant impact increases their applicability during system run-time.

Note that even small improvements in the estimation quality might have significant impact. Since we estimate the resource demand on a per request basis, the estimated resource demands are multiplied by a factor in dependence of the system size. Hence, a big system will still profit quite notably from small relative improvements.

Challenges and Future Work: We did not consider the Pareto optimization between estimation time and estimation error in this work. However, the estimation time is an important factor especially in online scenarios. Self-tuning parameters could also aim to minimize the estimation time, while keeping the estimation error constant. This work can be seen as a starting point towards improving resource demand estimation for autonomic online optimization during system operation.

During our work, we noticed some interesting clustering effects while optimizing the KFs. Some traces reacted contradictory to the rest when changing certain parameter values. We plan to investigate the root causes for this effect in order to be able to optimize the parameters more precisely.

Acknowledgment

This work was co-funded by the German Research Foundation (DFG) under grant No. (KO 3445/11-1) and by Google Inc. (Faculty Research Award). Arif Merchant, Google Inc., contributed with helpful feedback.

References

[1] G. Bolch, S. Greiner, H. de Meer, and K. S. Trivedi, *Queueing Networks and Markov Chains: Modeling and Performance Evaluation with Computer Science Applications*. New York: Wiley-Interscience, 1998.

[2] F. Bause, "Queueing petri nets-a formalism for the combined qualitative and quantitative analysis of systems," in *Proceedings of the 5th International Workshop on Petri Nets and Performance Models*, oct 1993, pp. 14 – 23.

[3] S. Becker, H. Koziolek, and R. Reussner, "The palladio component model for model-driven performance prediction," *J. Syst. Software*, vol. 82, no. 1, pp. 3 – 22, 2009, special Issue: Software Performance - Modeling and Analysis.

[4] S. Kounev, N. Huber, F. Brosig, and X. Zhu, "A Model-Based Approach to Designing Self-Aware IT Systems and Infrastructures," *IEEE Computer*, vol. 49, no. 7, pp. 53–61, July 2016. [Online]. Available: http://dx.doi.org/10.1109/MC.2016.198

[5] E. D. Lazowska, J. Zahorjan, G. S. Graham, and K. C. Sevcik, *Quantitative system performance: computer system analysis using queueing network models*. Upper Saddle River, NJ, USA: Prentice-Hall, Inc., 1984.

[6] D. A. Menascé, L. W. Dowdy, and V. A. F. Almeida, *Performance by Design: Computer Capacity Planning By Example*. Upper Saddle River, NJ, USA: Prentice Hall PTR, 2004.

[7] A. Bauer, N. Herbst, and S. Kounev, "Design and Evaluation of a Proactive, Application-Aware Auto-Scaler," in *ACM/SPEC ICPE 2017*, April 2017.

[8] S. Spinner, G. Casale, F. Brosig, and S. Kounev, "Evaluating Approaches to Resource Demand Estimation," *Perform. Evaluation*, vol. 92, pp. 51 – 71, October 2015. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0166531615000711

[9] F. Willnecker, M. Dlugi, A. Brunnert, S. Spinner, S. Kounev, and H. Krcmar, "Comparing the Accuracy of Resource Demand Measurement and Estimation Techniques," in *EPEW 2015*, ser. Lecture Notes in Computer Science, M. Beltrán, W. Knottenbelt, and J. Bradley, Eds., vol. 9272. Springer, August 2015, pp. 115–129.

[10] J. Rolia and V. Vetland, "Parameter estimation for performance models of distributed application systems," in *CASCON '95*. IBM Press, 1995, p. 54.

[11] F. Brosig, S. Kounev, and K. Krogmann, "Automated Extraction of Palladio Component Models from Running Enterprise Java Applications," in *VALUETOOLS '09*, 2009, pp. 1–10.

[12] W. Wang, X. Huang, X. Qin, W. Zhang, J. Wei, and H. Zhong, "Application-Level CPU Consumption Estimation: Towards Performance Isolation of Multi-tenancy Web Applications," in *IEEE CLOUD 2012*, Jun. 2012, pp. 439 –446.

[13] T. Zheng, C. Woodside, and M. Litoiu, "Performance Model Estimation and Tracking Using Optimal Filters," *IEEE TSE*, vol. 34, no. 3, pp. 391–406, May 2008.

[14] S. Spinner, G. Casale, X. Zhu, and S. Kounev, "Librede: A library for resource demand estimation," in *ACM/SPEC ICPE 2014*, ser. ICPE '14. New York, NY, USA: ACM, 2014, pp. 227–228.

[15] S. Kraft, S. Pacheco-Sanchez, G. Casale, and S. Dawson, "Estimating service resource consumption from response time measurements," in *VALUETOOLS '09*, 2009, pp. 1–10.

[16] D. Menascé, "Computing missing service demand parameters for performance models," in *CMG Conference Proceedings*, 2008, pp. 241–248.

[17] Z. Liu, L. Wynter, C. H. Xia, and F. Zhang, "Parameter inference of queueing models for IT systems using end-to-end measurements," *Perform. Evaluation*, vol. 63, no. 1, pp. 36–60, 2006.

[18] W. Wang, X. Huang, Y. Song, W. Zhang, J. Wei, H. Zhong, and T. Huang, "A statistical approach for estimating cpu consumption in shared java middleware server," in *IEEE COMPSAC, 2011*. IEEE, 2011, pp. 541–546.

[19] D. Kumar, A. Tantawi, and L. Zhang, "Real-time performance modeling for adaptive software systems," in *VALUETOOLS '09*, 2009, pp. 1–10.

[20] J. Grohmann, "Reliable Resource Demand Estimation," Master Thesis, University of Würzburg, Am Hubland, Informatikgebäude, 97074 Würzburg, Germany, October 2016. [Online]. Available: http://se2.informatik.uni-wuerzburg.de/pa/publications/download/paper/1159.pdf

[21] J. Rolia and V. Vetland, "Correlating resource demand information with ARM data for application services," in *Proceedings of the 1st international workshop on Software and performance*. ACM, 1998, pp. 219–230.

[22] G. Pacifici, W. Segmuller, M. Spreitzer, and A. Tantawi, "CPU demand for web serving: Measurement analysis and dynamic estimation," *Perform. Evaluation*, vol. 65, no. 6-7, pp. 531–553, 2008.

[23] G. Casale, P. Cremonesi, and R. Turrin, "Robust Workload Estimation in Queueing Network Performance Models," in *Euromicro PDP 2018*, Feb. 2008, pp. 183–187.

[24] ——, "How to Select Significant Workloads in Performance Models," in *CMG Conference Proceedings*, 2007.

[25] C. Stewart, T. Kelly, and A. Zhang, "Exploiting nonstationarity for performance prediction," *SIGOPS Oper. Syst. Rev.*, vol. 41, pp. 31–44, Mar. 2007.

[26] T. Zheng, J. Yang, M. Woodside, M. Litoiu, and G. Iszlai, "Tracking time-varying parameters in software systems with extended Kalman filters," in *CASCON '05*. IBM Press, 2005, pp. 334–345.

[27] A. B. Sharma, R. Bhagwan, M. Choudhury, L. Golubchik, R. Govindan, and G. M. Voelker, "Automatic request categorization in internet services," *SIGMETRICS Perform. Eval. Rev.*, vol. 36, pp. 16–25, Aug. 2008.

[28] Q. Noorshams, D. Bruhn, S. Kounev, and R. Reussner, "Predictive performance modeling of virtualized storage systems using optimized statistical regression techniques," in *ACM/SPEC ICPE 2013*, ser. ICPE '13. New York, NY, USA: ACM, 2013, pp. 283–294.

[29] Q. Noorshams, "Modeling and prediction of i/o performance in virtualized environments," Ph.D. dissertation, Karlsruhe Institute of Technology (KIT), 2015. [Online]. Available: http://digbib.ubka.uni-karlsruhe.de/volltexte/1000046750