Statistical Inference of Software Performance Models for Parametric Performance Completions

Jens Happe[1]*, Dennis Westermann[1], Kai Sachs[2], Lucia Kapová[3]

[1] SAP Research, CEC Karlsruhe, Germany
{jens.happe|dennis.westermann}@sap.com
[2] TU Darmstadt, Germany
sachs@dvs.tu-darmstadt.de
[3] Karlsruhe Institute of Technology (KIT), Germany
kapova@ipd.uka.de

Abstract. Software performance engineering (SPE) enables software architects to ensure high performance standards for their applications. However, applying SPE in practice is still challenging. Most enterprise applications include a large software basis, such as middleware and legacy systems. In many cases, the software basis is the determining factor of the system's overall timing behavior, throughput, and resource utilization. To capture these influences on the overall system's performance, established performance prediction methods (model-based and analytical) rely on models that describe the performance-relevant aspects of the system under study. Creating such models requires detailed knowledge on the system's structure and behavior that, in most cases, is not available. In this paper, we abstract from the internal structure of the system under study. We focus on message-oriented middleware (MOM) and analyze the dependency between the MOM's usage and its performance. We use statistical inference to conclude these dependencies from observations. For ActiveMQ 5.3, the resulting functions predict the performance with a relative mean square error 0.1.

1 Introduction

With the rising complexity of today's software systems, methods and tools to achieve and maintain high performance standards become more and more important. Software performance engineering [27] and model-based performance prediction (surveyed in [3] and [18]) provide software architects and developers with tools and methods to systematically estimate the expected performance of a software system based on its architectural specification. Performance engineering of today's enterprise applications entails a high degree of complexity. Enterprise application systems are very large and are rarely developed from scratch. In most cases, a sound base of software exists on which developers build their new applications. Such software bases include middleware platforms, third party components (or services), and legacy software. Up to date performance models for these systems are not available in most cases. Moreover, knowledge about the structure and performance of these systems is limited. However, the software basis of an application can have a major influence on its overall performance and thus has to be

* This work is supported by the European Community's Seventh Framework Programme (FP7/2001-2013) under grant agreement no.216556.

considered in performance predictions. Most existing approaches use established prediction models [3, 18] to estimate the performance of already existing complex software systems. Their main focus lies on the questions: i) "How can we automatically derive or extract the models we need?" and ii) "How can we estimate the resource demands / quantitative data needed for our models?" Approaches addressing the first question analyze call traces [7] or use static code analyses [20] to derive models of software systems. Approaches addressing the second question (e.g. [23, 19]) use benchmarking and monitoring of the system to extract model parameters. In order to apply these approaches, software architects have to instrument large parts of the system and conduct precise measurements. Furthermore, they are bound to the assumptions of the prediction model used. For example, if a network connection is modeled with FCFS scheduling, it won't capture the effect of collisions on the network. Another drawback of these approaches is that they do not scale with respect to the increasing size and complexity of today's software systems. Size and complexity can become inhibiting factors for quality analyses. The process of creating performance prediction models for those systems requires heavy effort and can become too costly and error-prone. For the same reason, many developers do not trust or understand performance models, even if such models are available. In case of legacy systems and third party software, the required knowledge to model the systems may even not be available at all. In such scenarios, re-engineering approaches (e.g. [20]) can help. However, re-engineering often fails due to the large and heterogeneous technology stack in complex application systems.

In our approach, we handle the complexity of large scale enterprise application systems by creating goal-oriented abstractions of those parts that cannot be modeled or only with high effort. For this purpose, we apply automated systematic measurements to capture the dependencies between the system's usage (workload and input parameters) and performance (timing behavior, throughput, and resource utilization). To analyze the measured data we use statistical inference, such as Bayesian networks or multivariate adaptive regression splines (MARS) [10]. The analysis yields an abstract performance model of the system under study. The abstractions are similar to flow equivalent servers used in queueing theory where a network of servers is replaced by a single server with workload-dependent service rate. The resulting models can be integrated in the Palladio Component Model (PCM) [6], a model-driven performance prediction approach. For this purpose, we combine the statistical models with parametric performance completions introduced in our previous work [11]. The combination of statistical models and model-based prediction approaches allows to predict the effect of complex middleware components, 3rd party software, and legacy systems on response time, throughput, and resource utilization of the whole software system.

In this paper, we focus on message-oriented middleware platforms which are increasingly used in enterprise and commercial domains. We evaluated our approach using the SPECjms2007 Benchmark for message-oriented systems. The benchmark resembles a supply chain management system for supermarket stores. In our case study, we used MARS and genetic optimization to estimate the influence of arrival rates and message sizes on the performance of a message-oriented middleware (MOM). The comparison of measurements and predictions yielded a relative mean square error of less than 0.1.

The contributions of this paper are i) statistical inference of software performance models, ii) their usage in combination with model-driven performance prediction methods, and iii) the application of our approach to a supply chain management scenario including a validation of predictions and measurements.

The paper is structured as follows. Section 2 gives an overview on our approach. Work related with performance prediction of message-oriented middleware and performance analysis using statistical inferencing is summarized in Section 3. In Section 4, we demonstrate how the performance of message-oriented middleware can be captured using a combination of systematic benchmarking and statistical inference. We discuss our results in Section 5. Finally, Section 6 concludes the paper.

2 Overview

In this section we describe our approach focusing on performance analyses of message-oriented middleware (MOM) platforms. The influence of general configurations and patterns on a MOM's performance (delivery time, throughput, and resource utilization) are well understood [11, 25]. However, the exact quantification of these influences is still cumbersome and has to be done for each implementation and each execution environment. In the case of MOM, the implementation is known to have a large impact on the overall performance [25]. Some implementations scale better with a larger number of processors or make better use of the operating system's I/O features. Capturing such low-level details in a generic performance model is impossible. Even if accurate performance models of a MOM are available, they have to be kept up to date and adjusted for new execution environments. Slight changes in the configuration can already affect the overall performance [11, 25, 16]. To consider such effects in a priori predictions, software architects need an approach to create accurate performance models for their middleware platform, even if the actual implementation is unknown. The performance models have to be parameterized and thus reflect the influence of the system's usage (input parameters, system state, arrival rate) on timing behavior, throughput, and resource utilization.

In our approach, we use systematic measurements and statistical inference to capture the performance of a MOM platform. We abstract from internals of the MOM implementation and identify functional dependencies between input parameters (message size, style of communication) and the observed performance. The resulting models are woven into the software architectural model. The combination of model-driven approaches and measurement-based model inference allows software architects to evaluate the effect of different middleware platforms on the performance of the overall system. Figure 1 illustrates the overall process of combining parametric performance completions with statistical model inference.

The process in Figure 1 is a specialization of the performance completion instantiation by the software architect [11, Figure 6]. We assume that the completion has already been designed and performance-relevant parameters are known. In the following, we describe the steps of the process in more detail.

Benchmarking (Data Collection) In the first step, we measure the influence of performance-relevant parameters for the middleware platform in its target execution environment. A standard industry benchmark (cf. Section 4.3) quantifies the delivery

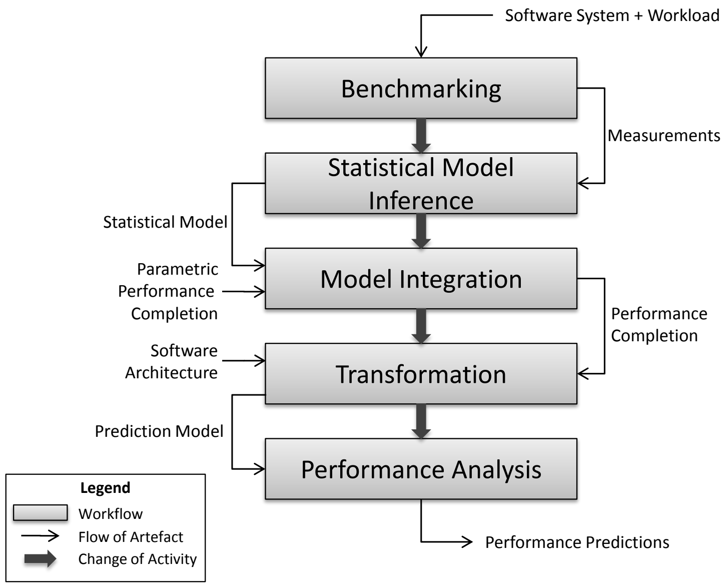


Fig. 1. Statistical Inference of Software Performance Models for Parametric Performance Completions.

time of messages, the MOM's throughput, and the utilization of resources. To capture the influence of parameters, the benchmark is executed several times with different configurations (cf. Section 4.1). In our experiments, we focus on the influence of arrival rates, messages sizes, and persistence of messages (messages are stored on hard disk until they are delivered).

Model Inference (Data Aggregation) The collected data is used to infer (parameters of) a prediction model. In Section 4.3, we use statistical inference techniques [13], more specifically Multivariate Adaptive Regression Splines (MARS) [10] and genetic optimization, to derive the influence of a MOM's usage on its performance.

Other inference techniques, such as [21, 23, 19], can be used to estimate parameters of queueing networks if the (major) resources and the structure of the system under study are known. However, these approaches are bound to the assumptions of the underlying queueing model, such as FCFS or PS scheduling, which may not hold in reality. Furthermore, they cannot (directly) predict the effect of input parameters (such as message size) on performance.

Statistical inference of performance metrics does not require specific knowledge on the internal structure of the system under study. However, statistical inference can require assumptions on the kind of functional dependency of input (independent) and output (dependent) 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 (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.

Model Integration The models inferred in the previous step are integrated into software performance models to predict their effect on the overall performance of the system. We use the Palladio Component Model (PCM) [6] in combination with parametric performance completions [11] to evaluate the performance of the system under study. The PCM is well suited for our purposes since it captures the effect of input parameters on software performance. Stochastic expressions of the PCM can be used to directly include the functions resulting from the statistical analysis into the middleware components of a parametric performance completion. Performance completions allow software architects to annotate a software architectural model. The annotated elements are refined by model-to-model transformations that inject low-level performance influences into the architecture [15]. Completions consist of an architecture-specific part that is newly generated for each annotation (adapter components) and an architecture-independent part that models the consumption of resources (middleware components). Completions are parametric with respect to resource demands of the middleware. The demands have to be determined for each middleware implementation and for each execution environment.

Transformation Finally, model-to-model transformations integrate the completion into architectural models [15]. The performance of the overall system can be determined using analytical models (such as queueing networks or stochastic Petri nets) or simulations.

In this paper, we focus on the first two steps (Benchmarking and Statistical Model Inference). A description of the remaining steps can be found in [11, 15].

3 Related Work

Current software performance engineering approaches can be divided in (i) early-cycle predictive model-based approaches (surveyed in [3] and [18]), (ii) late-cycle measurement-based approaches (e.g. [1, 2, 4]), and (iii) combinations of measurement-based and model-based approaches (e.g. [8, 19]) [29]. Late-cycle measurement-based approaches as well as the approaches that combine model-based and measurement-based performance engineering mainly rely on statistical inferencing techniques to derive performance predictions based on measurement data.
Zheng et al. [30] apply Kalman Filter estimators to track parameters that cannot be measured directly. To estimate the hidden parameters, they use the difference between measured and predicted performance as well as knowledge about the dynamics of the performance model. In [23] and [19], statistical inferencing is used for estimating service demands of parameterized performance models. Pacifici et al. [23] analyze multiple kinds of web traffic using CPU utilization and throughput measurements. They formulate and solve the problem using linear regressions. In [19], Kraft et al. apply a linear regression method and the maximum likelihood technique for estimating the service demands of requests. The considered system is an ERP application of SAP Business Suite with a

workload of sales and distribution operations. Kumar et al. [21] and Sharma et al. [26] additionally take workload characteristics into account. In [21], the authors derive a mathematical function that represents service times and CPU overheads as functions of the total arriving workload. Thereby, the functional representation differs depending on the nature of the system under test. The work focuses on transaction-based, distributed software systems. Sharma et al. [26] use statistical inferencing to identify workload categories in internet services. Using coarse grained measurements of system resources (e.g. total CPU usage, overall request rate), their method can infer various characteristics of the workload (e.g. the number of different request categories and the resource demand of each category). They apply a machine learning technique called independent component analysis (ICA) to solve the underlying blind source separation problem. The feasibility of their approach is validated using an e-commerce benchmark application.
Other researchers focus on measurement-based and/or analytical performance models for middleware platforms. Liu et al. [22] build a queuing network model whose input values are computed based on measurements. The goal of the queuing network model is to derive performance metrics (e.g. response time and throughput) for J2EE applications. The approach applied by Denaro et al. [9] completely relies on measurements. The authors estimate the performance of a software system by measurements of application specific test cases. However, both approaches simplify the behavior of an application, and thus, neglect its influence on performance. Recently, Kounev and Sachs [17] surveyed techniques for benchmarking and performance modeling of event-based systems. They reviewed several techniques for (i) modeling message-oriented middleware systems and (ii) predicting their performance under load considering both analytical and simulation-based approaches.

4 Capturing the Performance of Message-oriented Middleware with Statistical Inference

In the following, we demonstrate how the performance of Message-oriented Middleware (MOM) can be captured using statistical inference. For this purpose, we first introduce our method for gathering the required performance data (Section 4.1) as well as the tools and techniques to derive statistical models from measurements (Section 4.2). The application of these methods to message-oriented systems follows in Section 4.3. The resulting models reflect the influence of message size, arrival rate, and configurations on delivery times, resource utilization, and throughput. Finally, we compare our predictions to measurements that are not part of the training set (Section 4.4).

4.1 Measurement Method

In order to apply statistical inferencing to the performance of MOM, we first need to measure the influence of different parameters (e.g., message size and persistence) on its performance. The strategy of sampling the effect of different parameter combinations on performance is critical, since it has to be detailed enough to achieve accurate predictions but must also be kept feasible at the same time (i.e., measurements must not last too long).

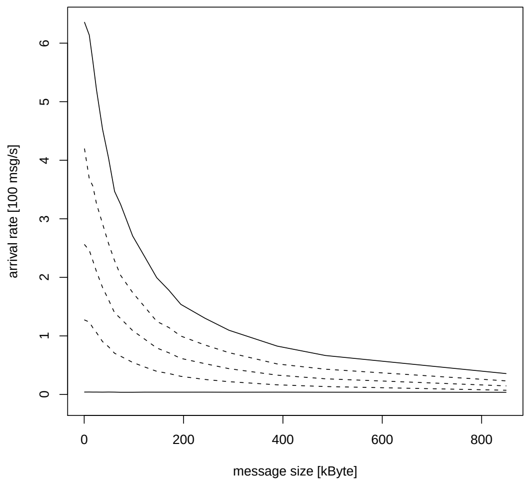


Fig. 2. Measurement strategy.

For this reason, we separate the measurements into three phases. First, we determine the maximal throughput of the system for each message size $m \in M$. While a high throughput can be achieved for small messages, the throughput decreases significantly for larger messages. In Figure 2, the upper solid line illustrates the dependency of the maximal throughput and the size of a message. In queuing theory, the throughput (X) can be computed from the resource utilization (U) and the service demand (D) by $X = U/D$. If we assume that the resource is fully utilized ($U = 1$), we can compute the maximal throughput $X_{max} = 1/D$. In a flow balanced system, the arrival rate (λ) is equal to the throughput of the system. Thus, the maximal arrival rate which can be handled by the system is $\lambda_{max} = X_{max}$. To determine the maximal arrival rate, we increase the load of the system until $U \approx 1$ and still $\lambda \approx X$, i.e., the system can still handle all incoming messages. We focus our measurements on arrival rates between 0 and λ_{max},i.e, $0 < \lambda \leq \lambda_{max}$. The performance measurements for λ_{max} represent the worst case performance of the system if it is still able to process all messages.

In the second phase, we measure the influence of individual parameters without resource contention. For example, we measure the effect of different message sizes on the delivery time when only one message is processed at a time. These measurements provide the baseline of our curve. They represent the best achievable performance for the system under study. In Figure 2, the solid line at the bottom depicts the baseline for a MOM with persistent message delivery (i.e., messages are stored on hard disk until they are delivered).

In the final phase, we sample the performance of the system under study between the best case and the worst case performance. For this purpose, we separate the arrival rate between the best and the worst case performance into N equidistant steps. For each message size $v \in V$, we measure the performance of the MOM for arrival rates of $\lambda_i = \lambda_{max} * i/N$ for $i \in \{1, \ldots, N-1\}$. The dashed lines in Figure 2 show the relative measurements for $N = 4$ steps.

4.2 Statistical Model Inference

Statistical inferencing is the process of drawing conclusions by applying statistics to observations or hypotheses based on quantitative data. The goal is to determine the relationship between input and output parameters observed at some system (sometimes also called independent and dependent variables). In this paper, we use *Multivariate Adaptive Regression Splines* (MARS) and genetic optimization to estimate the dependency between different system characteristics (configuration and usage) and performance metrics of interest.

Multivariate Adaptive Regression Splines (MARS): MARS [10] is a statistical method for flexible regression modeling of multidimensional data which has already been successfully employed in software performance prediction [8]. MARS is a non-parametric regression technique which requires no prior assumption as to the form of the data. The input data may be contaminated with noise or the system under study may be responding to additional hidden inputs that are neither measured or controlled. The goal is to obtain a useful approximation to each function using a set of training data. Therefore, 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 [10]. 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 [8]. 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.

Genetic Optimization (GO) If the functional dependency between multiple parameters is known, i.e., a valid hypothesis exists, then either non-linear regression or GO can be used to fit the function against measured data. Non-linear regressions allow various types of functional relationships (such as exponential, logarithmic, or Gaussian functions). Non-linear regression problems are solved by a series of iterative approximations. Based on an initial estimate of the value of each parameter, the non-linear regression method adjusts these values iteratively to improve the fit of the curve to the data. To determine the best-fitting parameters numerical optimization algorithms (such as Gauss-Newton or Levenberg-Marquardt) can be applied.

Another way of identifying a good-fitting curve for a non-linear problem is the use of GOs. In [14] the author describes the basic principals of GOs. GOs simulate processes of biological organisms that are essential to evolution. They combine two techniques at the same time in an optimal way: (i) exploration which is used to investigate new areas in the search space, and (ii) exploitation which uses knowledge found at points previously visited to help finding better points [5]. Compared to the non-linear regression techniques GO is more robust, but requires more time. In our case, the basis of the GO is an error function which has to be minimized. Errors represent the difference between the observations and the model's predictions. It must be taken into account that the definition of the error metric can influence the accuracy of fitting. For example, if error is expressed as absolute measure, the approximation is inaccurate for small values at large scattering. If error is expressed as relative measure, small values are approximated better while large values can show stronger deviations.

We use mean squared error (MSE) and relative mean squared error (RMSE) to measure the difference between predicted and observed value. The MSE is given by $MSE = \frac{1}{N}\sum_{i=1}^{N}(\widehat{y_i} - y_i)^2$ and the RMSE is given by $RMSE = \frac{1}{N}\sum_{i=1}^{N}(\frac{\widehat{y_i - y_i}}{y_i})^2$. In both cases N is the number of cases in the data set, y is defined as observed value, $\widehat{y_i}$ as the predicted value.

In the following section, we apply the methods for statistical model inference presented here to derive a performance model for message-oriented systems.

4.3 Analyzing Message-oriented Systems

In message-oriented systems, components communicate by exchanging messages using a message-oriented middleware. Such a loose coupling of communicating parties has several important advantages: i) message producers and consumers do not need to be aware of each other, ii) they do not need to be active at the same time to exchange information, iii) they are not blocked when sending or receiving messages. Most MOM platforms offer two types of communication patterns: (a) Point-to-Point (P2P), where each message is consumed by exactly one message receiver and (b) Publish/Subscribe, where each message can be received by multiple receivers [28]. A discussion of messaging patterns influencing the software performance is provided in [11].

MOM Benchmarks: SPECjms2007 & jms2009-PS *SPECjms2007* is the first industry standard benchmark for Java Message Services (JMS). It was developed by the Standard Performance Evaluation Corporation (SPEC) under the leadership of TU Darmstadt. The underlying application scenario models a supermarket's supply chain where RFID technology is used to track the flow of goods between different parties. Seven interactions such as order management are modeled in detail to stress different aspects of MOM performance.

jms2009-PS [24] is built on top of the SPECjms2007 framework and SPECjms2007 workload [25] using pub/sub communication for most interactions. Both benchmarks are focused on the influence of the MOM's implementation and configuration. The benchmarks minimize the impact of other components and services that are typically used in the chosen application scenario. For example, the database used to store business data and manage the application state could easily become the limiting factor and thus is not represented in the benchmark. This design allows us to focus our evaluation on the influences of MOM without disturbances.

Benchmark Application For our experiments, we selected *Interaction 4: Supermarket (SM) Inventory Management*. This interaction exercises P2P messaging inside the SMs. The interaction is triggered when goods leave the warehouse of a SM (e.g., to refill a shelf). Goods are registered by RFID readers and the local warehouse application is notified so that inventory can be updated. The size of such messages varies from very small (a single good) to very large (pallets). Therefore they can be used to test JMS performance for all message sizes.

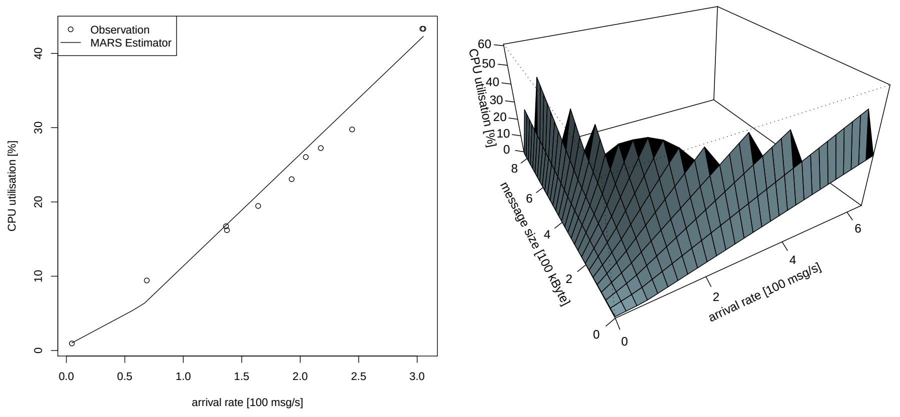
Experimental Environment We statistically inferred a performance model for Active MQ 5.3 running on a IBM x3850 Server with a 2-Core Intel Xeon 3.5 GHz, a RAID 10 with 6 SAS hard drives and 16 GByte of RAM running under Debian Linux 2.6.26. During the measurements, all satellites where hosted on a Windows Server 2003 System with 16 GByte of RAM and two 4-Core Intel Xeon 2.33 GHz. The utilization of the host for the satellites never exceeded 20%. The benchmark was executed 288 times. Each run lasted about 6 minutes leading to a total measurement time of approximately 31 hours. During each run, the inventory management send between 1800 and 216000 messages to the supermarket server. The actual number depends on the configured arrival rate of messages. For each run, we measured the utilization of CPUs and hard disk, network traffic and throughput as well as the delivery time of messages. In the following, we analyze the measurements collected by the jms2009-PS benchmark using the statistical inferencing techniques presented in Section 4.2.

Analysis We determine the functional dependency of performance metrics on the MOM's usage applying MARS and genetic optimization. For the analyses, the actual arrival rate and message size can be computed based on the benchmark's configuration. In case of interaction 4 "'Supermarket Inventory Management"', the size of a message v (in kilobyte) is given by the linear equation $v = m_1 * x + b$ [25], where $m_1 = 0.0970$ and $b = 0.5137$. Furthermore, the total arrival rate of messages ξ_4 per second is a multiple of the arrival rate for each supermarket (λ_4): $\xi_4 = \lambda_4 * |\Psi_{SM}|$, where $\Psi_{SM} = \{SM_1, SM_2, \ldots, SM_{|\Psi_{SM}|}\}$ is the set of all supermarkets. Since we only consider flow-balanced scenarios, the number of messages sent and received are equal. Furthermore, no other messages are sent through the channels of interaction 4.

In the first measurement phase, we determine the maximal throughput for persistent and non-persistent message delivery.The performance metrics collected in this setting represent the worst case performance of the MOM. The message size lies between 1 and 850 kBytes while the maximal throughput ranges from 6 to 900 messages per second. The maximal throughput decreases exponentially with an increasing message size for persistent and non-persistent delivery.

In the second phase, we analyze the influence of the message size without contention. The total arrival rate is set to 0.5 messages per second for this purpose. The results represent the best achievable performance for each message size. The mean delivery time of the messages ranges from less than a millisecond to approx. 85 ms. Here, we observed an almost linear growth of delivery time for increasing message sizes.

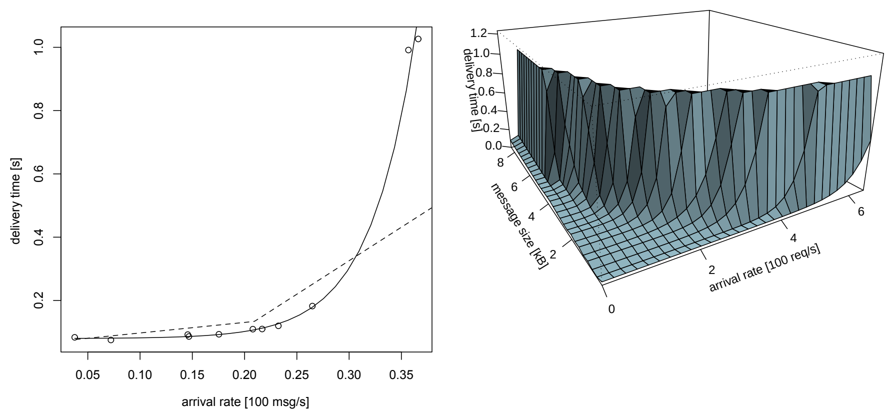
In the final phase, we analyze the intermediate performance between the best case and worst case observed. Figure 3 shows the utilization of the MOM's host machine as a function of message size and arrival rate. Figure 3(a) suggests that, for a fixed message



(a) Message size of 100kByte. (b) Combined influence.

Fig. 3. CPU utilization as a function of message size and arrival rate.

size, the dependency is almost linear. However, the gradient increases significantly for larger messages (cf. Figure 3(b)). In our measurements, the CPU utilization never exceeded 50%. This observation is unexpected, especially for large messages. However, the physical resources were not the limiting factor in our experiments, but in the implementation of the MOM. Active MQ 5.3 uses only a single thread to process its I/O. This thread can become the bottleneck in multiprocessing environments. Statistically inferred models can cover this effect without knowledge about the internal cause.



(a) Messages size of 850 kByte. (b) Combined influence.

Fig. 4. Functional dependency of delivery times on arrival rate and message size.

So far, MARS provided good approximations of the measurements. However, it fails to accurately reflect the effect of arrival rates and messages sizes on delivery times. Figure 4(a) shows the averages of the measured delivery time (circles) compared to predictions of MARS (dashed line). In this case, regression splines do not capture the steep increase in delivery times for an arrival rate of 35 msg/s (with messages of 850 kByte). Exponential distributions, which are fitted to the measurements using genetic optimization (cf. Section 4.2), provide much better results. In Figure 4(a), the solid line depicts an exponential function fitted to the delivery time of messages with a size of 850 kByte. For an arrival λ of messages with 850 kByte, the exponential function

$$f_{850}^{dt}(\lambda) = \exp(24.692 * (\lambda - 0.412)) + 0.079$$

accurately reflects the changes of the delivery time in dependence of the arrival rate of messages. The relative mean squared error (RMSE) of measurements and predictions is 0.061 compared to 3.47 for MARS. Figure 4(b) illustrates the result of the genetic optimization for different message sizes and different arrival rates. For each message size the exponential function is determined separately. The resulting terms are combined by linear interpolation. Let v be the size of the message whose delivery time is to be determined and m and n message sizes with $n \leq v < m$ that are close to v and for which $f_n^{dt}(\lambda)$ and $f_m^{dt}(\lambda)$ are known, then:

$$f^{dt}(v, \lambda) = f_n^{dt}(\lambda) + \frac{f_m^{dt}(\lambda) - f_n^{dt}(\lambda)}{m - n}(v - n).$$

This function reduces the sum of the relative mean squared error (RMSE) from 341.4 (MARS) to 10.7. The additional knowledge about the type of the function significantly decreases the error of our statistical estimator. However, at this point, it is still unclear whether linear interpolation is appropriate to estimate the delivery time for message sizes whose exponential functions have not been approximated explicitly. In the following section, we address this question by comparing measurements for message sizes and arrival rates that are not part of the training set to the predictions of the statistical models.

4.4 Evaluation of the Statistical Models

In order to validate the prediction model for MOM developed in Section 4.3, we compare the predicted delivery times and resource utilization to observations that are not part of the training set. The results indicate whether the approach introduced in the paper yields performance models with the desired prediction accuracy. More specifically, we address the following questions: i) ”Is the training set appropriate for statistical inferencing?” and ii) ”Are the chosen statistical methods for model inference sufficient to accurately reflect the systems performance?”. To answer both questions, we set up a series of experiments where the arrival rate λ is 200 messages per second and the message size is varied between 0.6 kByte and 165.4 kByte in steps of 9.7 kByte. The experiments have been repeated three times.

Figure 5 illustrates the results of the experiments as well as the corresponding predictions for delivery time (Figure 5(a)) and resource utilizations (Figure 5(b)–5(d)). In general, predictions and observations largely overlap. The interpolation used to estimate the delivery time captures the influence of messages sizes accurately. The relative mean squared error (RMSE) for predicted delivery times is 0.10. The partly unsteady shape of the prediction curve is a consequence of the interpolation. As a variant of the nearest neighbor estimator, it is quite sensitive to deviations in the training set and passes them

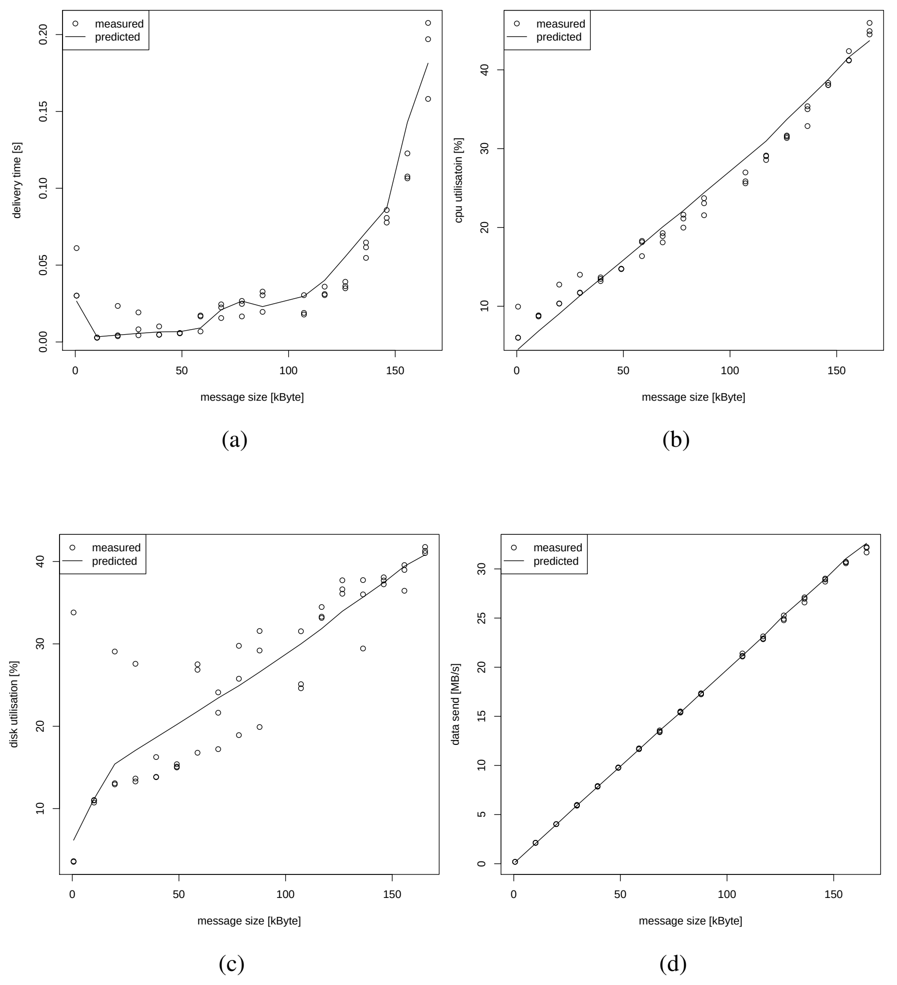


Fig. 5. Evaluation results.

down to the predictions. Furthermore, the almost linear growth of the utilization of all resources is accurately reflected in the corresponding MARS functions. The measured utilization of the hard disk shows a much larger variance than the utilization of other resources. This effect occurs because after each run the files used by Active MQ to store the messages are deleted. The deletion and recreation affects the fragmentation of the files which in turn can influence the disk utilization by a factor of three.

To answer the questions posed in the beginning, the training set is appropriate to capture the performance of the system under study. Furthermore, the chosen statistical inferencing techniques were able to extract accurate performance models from the training data. However, the interpolation for different message sizes might become instable if the training data has a high variance. In the following, we discuss the benefits and drawbacks of the approach for statistical model inferencing presented in this paper.

5 Discussion

The evaluation in the previous section demonstrates the prediction accuracy of performance models inferred using the approach proposed in this paper. For such models, no knowledge about the internals of the system under study is needed. In the case of Active MQ 5.3, its internal I/O thread became the limiting factor in our experiments while no physical resource was fully utilized. Modeling such behavior with queueing networks is challenging and requires extended queueing networks such as Queueing Petri Nets. The method for statistical inferencing proposed in this paper is based on functional dependencies only and thus allows to capture such effects without drilling down into the details of the middleware's implementation. However, observations like for Active MQ 5.3 provide valuable feedback for middleware developers, but are of minor interest for the application developers. They are mainly interested in how such internal bottlenecks will influence the performance of the overall system.

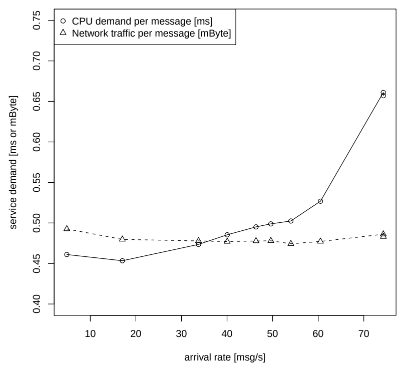


Fig. 6. Load dependent resource demand.

Load dependent resource demands are typically problematic in software performance prediction. Figure 6 shows the resource demand of a single message delivery for the network (in megabyte) and the CPU (in milliseconds). The data send over the network per message stays constant, independent of the arrival rate. By contrast, the resource demand for the CPU increases by 44% from 0.45 ms to 0.65 ms. The additional demand for processing time is caused by increasing I/O waits for a larger number of messages. Reflecting such a behaviour in queueing networks requires advanced inferencing techniques. By contrast, it plays a minor role for statistically inferred models proposed in this paper.

The drawback of such measurement-based models is the large amount of measurements necessary for their derivation. To derive the performance model for Active MQ 5.3, we conducted 288 benchmark runs that lasted approximately 31 hours. In this scenario, the measurements where fully automated and thus still feasible. However, as soon as the number of parameters that can be varied increases, the number of measurements needed increases rapidly. One approach to reduce the number of measurements is to add mare assumptions about the data dependencies to the inferencing technique. For example, given the exponential relationship of arrival rate and delivery time, only a very few measurements may be sufficient to characterize the function. An extensive discussion on the challenges of statistical inference for software performance models can be found in [12].

6 Conclusions

In this paper, we proposed statistical inferencing of performance models based on measurements only. Our approach focuses on the observable data and does not consider the structure of the system. We applied our approach to model the performance of Active MQ 5.3. The performance metrics considered include the delivery time of messages, throughput, and utilization of different resources. We used MARS as well as genetic optimization to infer their dependency on message size and arrival rates. The comparison of predictions and measurements demonstrated that the model can accurately predict the performance outside the original training set.

The method allows software architects to create performance models for middleware platforms without knowing or understanding all performance relevant internals. The models remain on high level of abstraction but still provide enough information for accurate performance analyses. Software architects can include a wide range of different implementations for the same middleware standard in their performance prediction without additional modeling effort. Having a simple means to include the performance influence of complex software systems into prediction models is a further, important step towards the application of software performance engineering in practice.

Based on the results presented in this paper, we plan the following steps. First, we need to reduce the number of measurements necessary to create performance models. This might be achieved by adding assumptions about the functional dependencies between input and output variables to the model. Furthermore, the results for one execution environment might be transferable to other environments with a significantly lower number of measurements. Second, we plan to apply our approach to other middleware platforms including common application servers. Finally, we will fully integrate the resulting models into software performance engineering approaches (namely the Palladio Component Model) to allow a direct usage of the models for the performance analysis of enterprise applications.

References

1. M. Arlitt, D. Krishnamurthy, and J. Rolia. Characterizing the scalability of a large web-based shopping system. *ACM Trans. on Internet Technology (TOIT)*, pages 44–69, 2001.
2. A. Avritzer, J. Kondek, D. Liu, and E. J. Weyuker. Software performance testing based on workload characterization. In *WOSP*, pages 17–24, 2002.
3. S. Balsamo, A. Di Marco, P. Inverardi, and M. Simeoni. Model-Based Performance Prediction in Software Development: A Survey. *IEEE Trans. on Software Engineering*, pages 295–310, 2004.
4. S. Barber. Creating effective load models for performance testing with incomplete empirical data. In *WSE*, pages 51–59, 2004.
5. David Beasley, David R. Bull, and Ralph R. Martin. An overview of genetic algorithms: Part 1, fundamentals, 1993.

6. S. Becker, H. Koziolek, and R. Reussner. The Palladio component model for model-driven performance prediction. *Journal of Systems and Software*, pages 3–22, 2009.
7. F. Brosig, S. Kounev, and K. Krogmann. Automated Extraction of Palladio Component Models from Running Enterprise Java Applications. In *ROSSA*, 2009.
8. M. Courtois and M. Woodside. Using regression splines for software performance analysis and software characterization. In *WOSP*, pages 105–114, 2000.
9. G. Denaro, A. Polini, and W. Emmerich. Early performance testing of distributed software applications. *SIGSOFT Software Engineering Notes*, pages 94–103, 2004.
10. J. H. Friedman. Multivariate adaptive regression splines. *Annals of Statistics*, pages 1–141, 1991.
11. J. Happe, S. Becker, Ch. Rathfelder, H. Friedrich, and R. H. Reussner. Parametric Performance Completions for Model-Driven Performance Prediction. *Performance Evaluation*, 2009.
12. J. Happe, H. Li, and W. Theilmann. Black-box Performance Models: Prediction based on Observation. In *QUASOSS*, pages 19–24, 2009.
13. T. Hastie, R. Tibshirani, and J. Friedman. *The Elements of Statistical Learning: Data mining, Inference ,and Prediction*. Springer Series in Statistics. 2009.
14. J. H. Holland. *Adaptation in Natural and Artificial Systems*. University of Michigan Press, 1975.
15. L. Kapova and T. Goldschmidt. Automated feature model-based generation of refinement transformations. In *SEAA-EUROMICRO*, pages 141–148, 2009.
16. L. Kapova, B. Zimmerova, A. Martens, J. Happe, and R. H. Reussner. State dependence in performance evaluation of component-based software systems. In *WOSP/SIPEW*, 2010.
17. S. Kounev and K. Sachs. Benchmarking and performance modeling of event-based systems. *it - Information Technology*, pages 262–269, 2009.
18. H. Koziolek. Performance evaluation of component-based software systems: A survey. *Performance Evaluation*, 2009.
19. S. Kraft, S. Pacheco-Sanchez, G. Casale, and S. Dawson. Estimating service resource consumption from response time measurements. In *Valuetools*, 2006.
20. K. Krogmann, M. Kuperberg, and R. Reussner. Using Genetic Search for Reverse Engineering of Parametric Behaviour Models for Performance Prediction. *IEEE Trans. on Software Engineering*, 2010.
21. D. Kumar, L. Zhang, and A. Tantawi. Enhanced inferencing: Estimation of a workload dependent performance model. In *Valuetools*, 2009.
22. Yan Liu, Alan Fekete, and Ian Gorton. Design-level performance prediction of component-based applications. *IEEE Trans. Software Eng.*, 31(11):928–941, 2005.
23. G. Pacifici, W. Segmuller, M. Spreitzer, and A. Tantawi. Dynamic estimation of cpu demand of web traffic. In *Valuetools*, page 26, 2006.
24. K. Sachs, S. Appel, S. Kounev, and A. Buchmann. Benchmarking publish/subscribe-based messaging systems. In *DASFAA Workshops: Benchmar'X10*, 2010.
25. K. Sachs, S. Kounev, J. Bacon, and A. Buchmann. Performance evaluation of message-oriented middleware using the SPECjms2007 benchmark. *Performance Evaluation*, 2009.
26. A. Sharma, R. Bhagwan, M. Choudhury, L. Golubchik, R. Govindan, and G. M. Voelker. Automatic request categorization in internet services, 2008.
27. C. U. Smith. *Performance Engineering of Software Systems*. Addison-Wesley, USA, 1990.
28. Sun Microsystems, Inc. Java Message Service (JMS) Specification - Version 1.1, 2002.
29. M. Woodside, Gr. Franks, and D. C. Petriu. The Future of Software Performance Engineering. In *ICSE*, pages 171–187, 2007.
30. T. Zheng, C. M. Woodside, and M. Litoiu. Performance model estimation and tracking using optimal filters. *IEEE Trans. Software Engineering*, pages 391–406, 2008.