Karlsruhe Institute of Technology

Evaluating Approaches to Resource Demand Estimation

Master Thesis of

Simon Spinner

At the Department of Informatics
Institute for Program Structures
and Data Organization (IPD)

Reviewer:	Prof. Dr. Ralf H. Reussner
Second reviewer:	Prof. Dr. Walter F. Tichy
Advisor:	Dipl.-Inform. Fabian Brosig
Second advisor:	Dr.-Ing. Samuel Kounev

Duration: 1 February 2011 – 31 July 2011

I declare that I have developed and written the enclosed Master Thesis completely by myself, and have not used sources or means without declaration in the text.

Karlsruhe, 31 July 2011

Abstract

Proactively managing the performance and resource efficiency of running software systems requires techniques to predict system performance and resource consumption. Typically, performance predictions are based on performance models that capture the performance-relevant aspects of the considered software system. Building performance models involves the estimation of resource demands, i.e., estimating the time a unit of work spends obtaining service from a resource.

A number of approaches to estimating the resource demands of a system already exist, e.g., based on regression analysis or stochastic filtering. These approaches differ in their accuracy, their robustness and their applicability. For instance, there are notable differences in the amount and type of measurement data that is required as input. However, to the best of our knowledge, a comprehensive evaluation and comparison of these approaches in a representative context does not exist.

In this thesis, we give an overview of the state-of-the-art in resource demand estimation and develop a classification scheme for approaches to resource demand estimation. We implement a subset of these estimation approaches and evaluate them in a representative environment. We analyze the influence of various factors of the environment on the estimation accuracy, considering the impact of current technologies, such as multi-core processors and virtualization.

Our work improves the comparability of existing estimation approaches and facilitates the selection of an approach in a given application scenario. Additionally, it shows possible directions for future research in the field of resource demand estimation.

Zusammenfassung

Um die Leistung und Ressourceneffizienz von Softwaresystemen zur Laufzeit vorausschauend steuern zu können, werden Methoden für die Vorhersage der Leistung und des Ressourcenbedarfs eines Systems benötigt. Vorhersagen zur Leistung basieren typischerweise auf Modellen, die relevante Leistungseigenschaften des betrachteten Softwaresystems erfassen. Das Erstellen eines solchen Modells beinhaltet die Schätzung von Ressourcenverbräuchen, d.h. die Schätzung der Zeit, die eine Ressource für die Bearbeitung eines Auftrags benötigt.

Es gibt bereits eine Reihe von Verfahren zur Schätzung der Ressourcenverbräuche eines Systems, die zum Beispiel auf Regressionsanalyse oder stochastischen Filtern basieren. Diese Verfahren unterscheiden sich in ihrer Genauigkeit, ihrer Robustheit und ihrer Anwendbarkeit. Es gibt zum Beispiel beträchtliche Unterschiede bei der Menge und der Art der Messdaten, die als Eingabe erwartet werden. Es gibt unserem Kenntnisstand zufolge keine umfassende Evaluation und kein Vergleich dieser Verfahren in einer repräsentativen Umgebung.

Diese Arbeit gibt einen Überblick über den aktuellen Stand der Technik im Bereich der Schätzung von Ressourcenverbräuchen und es wird ein Klassifikationsverfahren für Verfahren zur Schätzung von Ressourcenverbräuchen entwickelt. Wir implementieren eine Untermenge dieser Schätzverfahren und evaluieren sie in einer repräsentativen Umgebung. Wir analysieren den Einfluss von verschiedenen Umgebungseinflüssen auf die Genauigkeit der Schätzungen und betrachten dabei auch die Auswirkungen von aktuellen Technologien, wie zum Beispiel Multi-core Prozessoren und Virtualisierung.

Unsere Arbeit verbessert die Vergleichbarkeit von existierenden Schätzverfahren und vereinfacht die Auswahl eines Verfahrens in einem konkreten Anwendungsszenario. Außerdem verdeutlicht sie mögliche zukünftige Forschungsrichtungen auf dem Gebiet der Schätzung von Ressourcenverbräuchen auf.

Contents

List of Figures

List of Tables

1. Introduction

People and companies increasingly rely on IT services to automate tasks and processes in many application areas. These services are subject to a set of performance requirements. A failure to meet performance requirements can result in substantial financial losses [MDA04]. Therefore, performance plays a vital role in the development and operation of IT services.

Performance requirements define objectives for the timeliness and resource efficiency of a system [Smi02]. The performance of a system can be defined as the degree to which a system meets these objectives. This definition of performance is compatible with green computing [WC08], as it postulates the minimization of the resource consumption of a system while ensuring its timeliness objectives, such as response time and throughput.

Software projects are often focused primarily on functional correctness and postpone performance considerations to later phases of the system lifecycle [SW02]. There is a common belief that performance problems can be solved by simply adding additional hardware or tuning the software later. However, the time and cost required to fix the performance of a system at later stages outweighs the costs of built-in performance [Smi90]. Therefore, performance requirements need to be considered during the complete system lifecycle, starting with the requirements analysis, through design and development, and all the way up to operation and maintenance.

Software Performance Engineering (SPE) is a "systematic, quantitative approach to the cost-effective development of software systems to meet performance requirements" [SW02]. At each stage of the system lifecycle, SPE provides methods to predict the expected performance of a system and helps to achieve the optimal performance [MDA04]. At system design time, performance evaluation helps to compare different design alternatives regarding their performance characteristics. At deployment time, performance prediction allows to determine the required server capacity for current and future workloads. During system operation, the performance influence of workload and system configuration changes needs to be evaluated when proactively managing software systems to permanently ensure performance requirements.

Performance models are commonly used to predict the performance of a system [Kou05]. A performance model is an abstraction of a real system capturing its performance-relevant aspects. Performance models are usually less expensive to build and analyze than performing experiments on a real system and can also be used in early stages of the system lifecycle, when no running system is available.

1.1. Motivation

The construction of a performance model involves its parameterization. We need to determine these parameters based on available information about the system represented by the performance model. One type of parameters are resource demands. Resource demands describe to what extent a resource is used for serving a request from a client. Resource demands are of vital importance when analyzing a performance model quantitatively. Determining representative values of resource demands can be a challenging and time consuming process [LZGS84]. In most cases, resource demands are not directly observable at the system of interest. Then we have to estimate the resource demands from available performance measures. Therefore, efficient and reliable estimation approaches for resource demands are necessary.

Many approaches to resource demand estimation have been proposed over the years. These estimation approaches use varying mathematical methods, e.g., regression analysis or stochastic filtering, to infer resource demands from measurements at a real system. When selecting an appropriate approach to resource demand estimation, we have to consider different characteristics of the estimation approach, such as the expected input measurements, its accuracy or its robustness to measurement anomalies. Depending on the constraints of the application context, only a subset of the estimation approaches is applicable.

To the best of our knowledge there is no comprehensive survey of the state-of-the-art in resource demand estimation. Furthermore, the evaluation of existing approaches to resource demand estimation differs in its scope. Most of the work is focused on specific characteristics of one estimation approach. A comparative evaluation of different estimation approaches has not been performed yet. Therefore, the selection of an appropriate approach to resource demand estimation is a challenging task.

In this thesis, we develop a classification scheme for approaches to resource demand estimation. The classification scheme will define the dimensions along which estimation approaches can be described and compared. Additionally, we will carry out further evaluation of existing approaches in order to improve the comparability of estimation approaches. We see the following benefits of this work:

- An overview of the state-of-the-art in resource demand estimation is provided in this thesis. It can serve as an introduction to the topic of resource demand estimation.
- Each approach to resource demand estimation is subject to certain limitations. The classification scheme and the subsequent evaluation can help to identify and show the limitations of an estimation approach.
- The classification scheme can provide guidelines for a systematic selection of an approach to resource demand estimation. Performance engineers are enabled to readily compare a number of estimation approaches and to consider the pros and cons of each estimation approach.
- Both the classification scheme and the evaluation can help to identify future research directions in the field of research demand estimation. These include extensions of existing estimation approaches, combinations of two or more approaches, or the application in new contexts, e.g., CPUs with dynamic voltage and frequency scaling or in virtualized environments.

1.2. Aim of the Thesis

The aim of this thesis is the classification of existing approaches to resource demand estimation and their evaluation in a realistic experiment environment. Based on a comprehensive literature research, we describe the state-of-the-art in resource demand estimation

and compare the assumptions and characteristics of the different approaches to resource demand estimation. We build a classification scheme for these approaches to resource demand estimation showing commonalities and differences between estimation approaches. The evaluation comprises a set of experiments in a realistic experiment environment. The evaluation is focused on online estimation approaches, which are applicable during system operation.

In this thesis, we pursue the following goals:

- Identify existing approaches to resource demand estimation and determine their intrinsic assumptions and constraints.
- Develop a classification scheme for approaches to resource demand estimation. The idea of the classification scheme is to facilitate the selection of an estimation approach given a specific application scenario.
- Design a tool enabling the analysis of approaches to resource demand estimation.
- Design and conduct a series of experiments in a realistic experiment environment. The goal of the evaluation is to improve the comparability of different estimation approaches.
- Identify future research directions in the field of resource demand estimation.

We provide ready-to-use implementations of a selected subset of approaches to resource demand estimation as part of our tool. The tool uses non-intrusive, low overhead measurements from standard monitoring tools as input, prepares the measurement data for estimation and executes a configurable number of estimation approaches. The tool provides common functionality shared by many estimation approaches and can significantly reduce the effort of implementing additional estimation approaches.

1.3. Outline

Chapter 2 explains the foundations of the thesis. We then give an overview of the state-of-the-art in resource demand estimation in Chapter 3. In Chapter 4, we describe the classification scheme for approaches to resource demand estimation. Chapter 5 gives an overview of how we evaluate the approaches to resource demand estimation. In Chapter 6, we explain the implementation of our tool for resource demand estimation. Chapter 7 contains the evaluation results.

2. Foundations

In this chapter, we give a short introduction to methods, models and tools that are used later in the thesis. First we describe the core concepts of queueing theory. Then we shortly explain the mathematical methods used by different approaches to resource demand estimation. Afterwards, we show how to obtain measurements required for resource demand estimation. Finally, we give an overview of the Ginpex framework and the TPC-W benchmark, which will be used for evaluation purposes.

2.1. Queueing Theory Background

The core concept of queueing theory is a *queue* (also called service station). In general terms, a queue consists of a waiting line and one or more servers. Customers arrive at the queue and are served immediately at the server if at least one is currently not occupied. Otherwise, they have to wait in the waiting line until a server is free. After service completion the customers leave the queue. Figure 2.1 shows an example of a queue with a single server. The queueing paradigm can be used to create performance models of computer systems. Physical resources, such as CPUs or disks, are represented by queues in the performance model. In this context the server is also called *resource* and *requests* (or transactions or jobs) are processed by the resource. In the following description we will use these more specific terms.

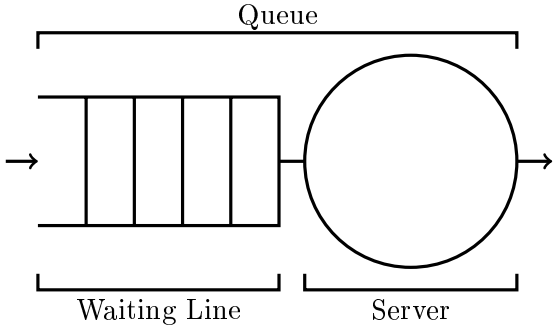


Figure 2.1.: Single-server queue.

A number of terms are commonly used when describing the timing behaviour of a queue. Requests arrive at the queue at arbitrary points in time. The number of requests per

time unit is called *arrival rate*. The time span between consecutive requests is called *interarrival time*. Each request requires a certain amount of processing at a resource. The time a resource is occupied with processing a request is called *service time*. The time a request resides in the waiting line waiting for service is called *queueing delay*. When one request is completed, the next is selected from the requests in the waiting line according to a *scheduling strategy*. Typical scheduling strategies are First-Come-First-Serve (FCFS), Processor-Sharing (PS) or Infinite-Server (IS). The sum of service time and queueing delay is called the *response time* of a request at a queue.

There is a standard notation for describing a queue, which is called *Kendall's notation*. It consists of six parameters $A/S/m/B/K/SD$ where [Kou05]:

- A stands for the interarrival time distribution,
- S stands for the service time distribution,
- m specifies the number of servers,
- B specifies the maximum number of requests that a queue can hold,
- K specifies the maximum number of requests that can arrive at the queue,
- and SD stands for a scheduling discipline.

Typical values for the distribution parameters are [Kou05]:

- M = Exponential (Markovian) distribution,
- D = Deterministic distribution, i.e. constant times without variance,
- E_k = Erlang distribution with parameter k,
- and G = General distribution.

Several queues can be connected together to form a Queueing Network (QN). Connections between queues describe the possible routes on which a request can move through the QN. If the requests come from outside sources to the QN and leave it on service completion, the QN is called *open*. If no requests enter or leave a QN, it is *closed*. If a QN is open for some requests and closed for others, it is called *mixed*.

If we want to analyze a QN quantitatively, we have to characterize its workload. The workload describes which and how many requests arrive at the system. Requests are grouped into workload classes according to selected criteria, such as involved applications, functions, resource consumption, etc. [Kou05, p. 133f]. For each workload class the *workload intensity* and the *resource demands* for each individual queue need to be specified. The workload intensity describes the number of requests arriving at the system. The resource demand is defined as following [LZGS84, MDA04]:

Definition (Resource demand or service demand). *The resource demand is the total average service time of a request at a resource over all visits. It can be defined as*

$$D_{i,r} = V_{i,r} \cdot S_{i,r}, \tag{2.1}$$

where $D_{i,r}$ is the service demand of requests of class r at resource i, $V_{i,r}$ is the number of visits of requests of class r at resource i and $S_{i,r}$ is the average service time of requests of class r at resource i.

The terms resource demand and service demand can be used interchangeably. In the following we will use the term resource demand.

Given a QN, we can determine performance quantities of this QN using basic relationships from operational analysis. In the following, two of those relationships, which are relevant for resource demand estimation, are described. The Utilization Law is defined as [MDA04, p. 64f]

$$U_{i,r} = S_{i,r} \cdot X_{i,r} \tag{2.2}$$

where $U_{i,r}$ is the utilization at resource i due to requests of class r, $S_{i,r}$ is the mean service time of a request at resource i and $X_{i,r}$ is the throughput of resource i.

The Service Demand Law is defined as [MDA04, p. 65f]:

$$D_{i,r} = \frac{U_{i,r}}{X_{0,r}}. \tag{2.3}$$

It relates the service demand of requests of class r at resource i to the utilization $U_{i,r}$ and the total system throughput of class r. $V_{i,r}$ specifies the number of visits of requests of class r at resource i.

If several assumptions hold for a QN, we can directly derive the mean response time of requests at a queue with load-independent resource demands with the relation [MDA04, p. 366]

$$R_{i,r} = \frac{D_{i,r}}{1 - U_i} \tag{2.4}$$

where $R_{i,r}$ is the mean response time of requests of class r at resource i, $D_{i,r}$ is the resource demand of requests of class r at resource i and U_i is the utilization of resource i. This equation is valid for open QNs. See [MDA04] for the corresponding equations of closed QNs. When using this equation, the interarrival times must have an exponential distribution. Furthermore, the QN must have a product-form solution. The BCMP theorem of Baskett, Chandy, Muntz and Palacios defines a combination of service time distributions and scheduling disciplines for which multi-class QNs have a product-form solution [MDA04]. If a queue has a FCFS scheduling strategy, the service time distribution must be exponential. PS queues can have any kind of service time distribution. For other scheduling strategies see [MDA04].

2.2. Mathematical Background

In this section, we explain mathematical methods that are commonly used for resource demand estimation. The methods are presented here in a general manner. In Section 3.1, the application of these methods in the context of resource demand estimation is described.

2.2.1. Linear Regression

Given a set of independent variables $x_1 \dots x_k$ and a dependent variable y, linear regression tries to describe the relationship between the dependent variable and the independent variables with the linear model

$$y = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \ldots + \beta_k x_k + \epsilon. \tag{2.5}$$

In regression analysis, y is known as *response variable* and x_j with $1 \leq j \leq k$ as *control variables*. The goal is to determine the parameters β_j with $0 \leq j \leq k$ in such a way that the residuals ϵ are minimized regarding a specific measure. Examples for such measures are the sum of squared residuals used in Least Squares (LSQ) regression or the sum of absolute differences used for Least Absolute Differences (LAD) regression. To be able to determine a unique solution for the parameters β_j, at least n sets of known values

$(y, x_1 \dots x_k)$ are required, where $n > k$. The above linear model can be written in matrix notation as [CP95, p. 96ff]

$$\mathbf{Y} = \mathbf{X}\boldsymbol{\beta} + \boldsymbol{\epsilon} \tag{2.6}$$

with following matrices

$$\mathbf{X} = \begin{pmatrix} 1 & x_{1,1} & x_{1,2} & \cdots & x_{1,k} \\ 1 & x_{2,1} & x_{2,2} & \cdots & x_{2,k} \\ \vdots & \vdots & \vdots & & \vdots \\ 1 & x_{n,1} & x_{n,2} & \cdots & x_{n,k} \end{pmatrix}, \mathbf{Y} = \begin{pmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{pmatrix}, \boldsymbol{\epsilon} = \begin{pmatrix} \epsilon_1 \\ \epsilon_2 \\ \vdots \\ \epsilon_n \end{pmatrix} \text{ and } \boldsymbol{\beta} = \begin{pmatrix} \beta_0 \\ \beta_1 \\ \vdots \\ \beta_k \end{pmatrix}.$$

$\mathbf{X}$ is called control matrix and $\mathbf{Y}$ is the response vector. We assume that the vector of error residuals $\boldsymbol{\epsilon}$ is independent and identically distributed with mean $E[\boldsymbol{\epsilon}] = 0$ and a constant variance. Then we can conclude that $E[\mathbf{Y}] = \mathbf{X}\boldsymbol{\beta}$ [CP95, p. 97]. The parameter vector $\boldsymbol{\beta}$ needs to be estimated.

LSQ regression estimates the parameter vector $\boldsymbol{\beta}$ by minimizing the sum of squared residuals. Hence, the following expression needs to be minimized

$$\boldsymbol{\epsilon}^T\boldsymbol{\epsilon} = (\mathbf{Y} - \mathbf{X}\boldsymbol{\beta})^T(\mathbf{Y} - \mathbf{X}\boldsymbol{\beta}). \tag{2.7}$$

The value $\hat{\boldsymbol{\beta}}$ that minimizes the previous expression can be calculated with following formula [CP95, p. 97]

$$\hat{\boldsymbol{\beta}} = (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbf{Y}. \tag{2.8}$$

2.2.2. Kalman Filtering

Statistical filtering is about the estimation of a hidden state of a dynamic system from known system inputs and incomplete and noisy measurements [KTZ09]. In this context, the term *state* is defined as follows:

> "The states of a system are those variables that provide a complete representation of the internal condition or status at a given instant of time." [Sim06, p. xxi]

The term *dynamic system* implies that the state of the system changes over time. Different statistical filtering methods have been proposed. We describe the Kalman filter here in more detail because it is often used to estimate resource demands.

Generally speaking, we can distinguish between discrete-time and continuous-time systems. Accordingly, different definitions of the Kalman filter exist for these types of system. Subsequently, we will concentrate on discrete-time Kalman filters. The notation used is based on the one used in [KTZ09] and [Sim06].

The system state $\mathbf{x}$ is a vector containing the variables that describe the internal state of a system. These variables cannot be directly observed at a system. The Kalman filter estimates the vector $\mathbf{x}$ from a series of measurements $\mathbf{z}$. The system is described by two equations. The first equation describes how the system state evolves over time according to [Sim06]

$$\mathbf{x}_k = \mathbf{F}_{k-1}\mathbf{x}_{k-1} + \mathbf{G}_{k-1}\mathbf{u}_{k-1} + \mathbf{w}_{k-1}. \tag{2.9}$$

The time advances in discrete steps, which are denoted by index k. $\mathbf{x}_k$ is the system state at time step k, which is calculated from the previous system state $\mathbf{x}_{k-1}$ and the control vector $\mathbf{u}_{k-1}$ containing the inputs of the system. The matrices $\mathbf{F}$ and $\mathbf{G}$ are called state transition model and control-input model. The process noise $\mathbf{w}_{k-1}$ is assumed to be normally distributed with zero mean and covariance $\mathbf{Q}_k$.

The second equation describes the relationship between the system state $\mathbf{x}_k$ and the measurements $\mathbf{z}_k$ at time step k according to [Sim06]

$$\mathbf{z}_k = \mathbf{H}_k \mathbf{x}_k + \mathbf{v}_k. \tag{2.10}$$

The matrix $\mathbf{H}_k$ is the observation model, which maps the state space to the observation space. $\mathbf{v}_k$ is the observation noise, which is assumed to be Gaussian white noise with zero mean and covariance $\mathbf{R}_k$.

If the relation $\mathbf{z} = h(\mathbf{x})$ between system state and measurements is non-linear, the Extended Kalman filter (EKF) can be used. The EKF approximates a linear model with the following output sensitivity matrix:

$$\mathbf{H}_k = \left[\frac{\partial h}{\partial x}\right]_{\hat{x}_{k|k-1}}. \tag{2.11}$$

The output sensitivity matrix is set to the Jacobian matrix of $h(\mathbf{x})$. The partial derivatives are evaluated with the current estimates of the system state. The vector $\hat{\mathbf{x}}_{n|m}$ represents the estimated system state $\hat{\mathbf{x}}$ at time step n given measurements $\mathbf{z}_1 \dots \mathbf{z}_m$.

The Kalman filter is a recursive estimator. It starts with an initial state and continuously updates its estimate as new measurements are obtained. At each time step k the calculations only depend on the previous estimate $\hat{\mathbf{x}}_{k-1|k-1}$ and the current measurements vector $\mathbf{z}_k$ [KTZ09]. The internal state of the filter is represented by two variables:

- the state estimate $\hat{\mathbf{x}}_{k|k}$,
- and the error covariance matrix $\mathbf{P}_{k|k}$.

The error covariance matrix is a measure for the estimated accuracy of the state estimates [KTZ09]. At the beginning the filter is initialized with given values for $\hat{\mathbf{x}}_{0|0}$ and $\mathbf{P}_{0|0}$.

The algorithm that calculates new state estimates consists of two phases [KZT09]: *Predict* and *Update*. In the predict phase a new state estimate $\hat{\mathbf{x}}_{k|k-1}$ is calculated with Equation 2.9. In the update phase the prediction error of $\hat{\mathbf{x}}_{k|k-1}$ is determined according to the current measurements $\mathbf{z}_k$. Then a corrected estimate $\hat{\mathbf{x}}_{k|k}$ is calculated. These two steps are carried out each time a new measurement sample vector gets available.

Assuming a linear relationship between the measurements and the system state, and uncorrelated and normally distributed noise with zero mean, the Kalman filter is an optimal estimator. Since most systems are inherently nonlinear, the EKF provides a linear approximation for cases with slightly nonlinear characteristics [ZWL08]. When applying the EKF to nonlinear problems, it is a Best Linear Unbiased Estimator (BLUE).

2.2.3. Mathematical Optimization

The estimation of resource demands can be formulated as a mathematical optimization problem. Generally speaking, an optimization problem is described by a *cost (objective) function* f with a domain $D \subseteq \mathbb{R}^n$ and a *constraint set* $\Omega \subseteq D$ [Dos09]. The objective function can be either minimized or maximized. In the following, we assume that it should be minimized. Then the optimization problem is also called *minimization problem* and can be solved by finding a value $\bar{x} \in \Omega$, so that

$$f(\bar{x}) \leq f(x),\ x \in \Omega. \tag{2.12}$$

Solutions of a minimization problem are called *(global) minimizers* [Dos09]. In contrast to global minimizers, there are also *local minimizers*. A local minimizers $\bar{x}$ satifies the condition

$$f(\bar{x}) \leq f(x),\ x \in \Omega,\ \|x - \bar{x}\| \leq \delta \tag{2.13}$$

for $\delta > 0$ [Dos09].

Optimization problems can be classified into different categories. There are *constrained* and *unconstrained* optimization problems. In the case of unconstrained optimization problems, there are no additional constraints in the constraint set, i.e., $\Omega = D$. If additional equality and inequality constraints are given, we speak of constrained optimization. Depending on the degree of the objective function and the constraints the following types of optimization problems exist:

- *Linear programming* problems have a linear objective function and a set of linear equality and inequality constraints.
- *Quadratic programming* problems are optimization problems with a quadratic objective function and a set of linear equality and inequality constraints.
- *Non-linear programming* problems can have any kind of non-linear objective function and/or non-linear constraints.

Different solution algorithms exist for the different types of optimization problems. Descriptions of possible solution algorithms can be found in [Dos09] and [Nem89].

2.2.4. Independent Component Analysis

The original motivation for developing Independent Component Analysis (ICA) can be illustrated with the so-called cocktail-party problem [HO00]: If there are a number of people in a room talking simultaneously, how can we separate the speech signal of each person assuming that only a mix of all signals can be recorded? The procedure of separating the original signals from a set of mixed signals without deeper knowledge of the original signal is called *blind source separation*. ICA is one method for blind source separation. Today, ICA is used not only in signal processing but also in many other domains. As we will later see in Section 3.1.4, it can be used for resource demand estimation, too.

Assuming that the observed mixtures x_i are always a linear combination of the independent components s_j with some parameters a_{ij}, the relation can be written as [HO00]

$$\begin{aligned} x_1 &= a_{11}s_1 + a_{12}s_2 + \ldots + a_{1n}s_n \\ x_2 &= a_{21}s_1 + a_{22}s_2 + \ldots + a_{2n}s_n \\ &\vdots \\ x_m &= a_{m1}s_1 + a_{m2}s_2 + \ldots + a_{mn}s_n. \end{aligned}$$

x_i and s_j are both random variables. This linear equation can be also written using matrix notation as

$$\mathbf{x} = \mathbf{A}\mathbf{s} \tag{2.14}$$

where $\mathbf{x}$ is a random vector of the mixtures x_i, $\mathbf{s}$ a random vector of the independent components s_j and $\mathbf{A}$ is called mixing matrix. Equation 2.14 describes the ICA model. The general procedure of ICA is as follows: First ICA estimates the mixing matrix $\mathbf{A}$, then it calculates the inverse of $\mathbf{A}$ and derives the independent components $\mathbf{s}$ [HO00]. There are different methods to estimate the mixing matrix $\mathbf{A}$. These methods are based on the

assumption that the elements in vector **s** are statistically independent. See [HO00] for examples.

It is important to note here that ICA does only work if the independent components have a non-Gaussian distribution. If this is not the case, ICA fails to estimate the mixing matrix **A**. In the case of Gaussian variables, the matrix **A** is not identifiable [HO00]. In fact, it is possible to have one Gaussian variable in the model [HO00]. Then ICA still works.

2.3. System Monitoring Approaches

In order to determine the resource demands of a performance model correctly, we usually depend on measurements at a running system. We can use monitoring tools to obtain these measurements. Menascé [MDA04] distinguishes between three types of monitors:

Hardware. Hardware monitors provide measurements at the hardware level. They incur no additional overhead. However, software-related information cannot be captured.

Software. A software monitor uses *application* or *kernel instrumentation* to perform measurements. With these monitors it is possible to collect more fine-grained measurements, e.g., at process or even function-level. However, there are drawbacks [PSST08]: application instrumentation requires changes to the code, kernel instrumentation relies on middleware or kernel agents to collect the data. In both cases the monitoring activities can produce a significant overhead.

Hybrid. Hybrid monitors combine features of hardware and software monitors. However, these monitors need to be integrated into the system and are not available in common systems.

Additionally, there are two possible modes of monitoring [MDA04]: event-trace and sampling. Event-trace monitors collect and count single events, such as disk read operations. If the event rate is high, this approach is not practical anymore. Sampling monitors only collect data at discrete points in time. Thus, depending on the sampling interval, they can reduce the monitoring overhead, even if the event rate is very high. However, the precision also declines with longer sampling intervals.

Apart from monitoring tools, log files can be used as a source of information. These log files are often created automatically by servers. For instance, web servers usually keep track of all requests in a log. Performance measures, such as response time or queue length, can be extracted from these log files [KPSCD09].

2.4. Ginpex Framework

Ginpex [HKHR11] (Goal-oriented INfrastructure Performance EXperiments) is a framework that allows to determine infrastructure properties, such as OS scheduling parameters or virtualization overhead, with automated experiments. The framework can induce certain load patterns on a System under Test (SUT) during these experiments. We use this feature of the framework during evaluation in order to generate artificial workloads with specified resource demands.

A measurements model contains a set of tasks modeling the workload pattern and a set of probes defining the measurements of interest. There are different tasks available: control flow tasks, e.g., loop and parallel tasks, and resource demand tasks for different types of resources (CPU, network, I/O and disks). With these tasks it is possible to model custom workload patterns. The measurements model is transformed into a Java program that is deployed and run on the SUT. During program execution the specified measurements are obtained from the SUT.

2.5. TPC Benchmark W

TPC-W [TPC02] is a web application benchmark simulating the operation of an online bookstore. The web application is designed to be representative of typical internet commerce applications [TPC02]. The benchmark consists of a database, a set of dynamic web pages and a load driver that emulates user sessions, also called Emulated Browsers (EBs). The standard does not prescribe any implementation technologies. In our evaluation, we use an implementation[1] based on Java servlets and a MySQL database.

Browsing Type	**Ordering Type**
Home	Shopping Cart
New Products	Customer Registration
Best Sellers	Buy Request
Product Detail	Buy Confirm
Search Request	Order Inquiry
Search Result	Order Display
	Admin Request
	Admin Confirm

Table 2.1.: TPC-W transactions.

The TPC-W standard defines 14 transactions that the online bookstore supports. These transactions are listed in Table 2.1. The transactions can be roughly classified into two groups: *browsing* and *ordering* type. The former consists of read-only transactions, the latter contains transactions updating the database. Furthermore, the standard also defines three transaction mixes. These transaction mixes have different ratios between browsing and ordering transactions. The list of the standard transaction mixes is shown in Table 2.2.

Transaction Mix	**Browsing Type**	**Ordering Type**
Browsing	95 %	5 %
Shopping	80 %	20 %
Ordering	50 %	50 %

Table 2.2.: TPC-W transaction mixes.

The workload is generated as follows: Each EBs represents one user in the system. An EB starts a user session, in which it calls several of the transactions in Table 2.1. The standard describes the possible navigation paths within a user session with state transition tables including transition probabilities between the transactions. The user think times between two transactions and the minimum user session lengths are drawn from an exponential distribution [TPC02]. It is possible to scale the workload by configuring the mean user think time and by changing the database size. The database size is determined by the number of items and the number of customers.

[1] `http://jmob.ow2.org/tpcw.html`

3. State of the Art

This chapter describes the current state of resource demand estimation. We did a comprehensive literature research on the topic of resource demand estimation and identified a number of different approaches that have been proposed so far. We describe the results of this literature research in the following sections. In Section 3.1, we explain how the different approaches work and what their fundamental assumptions are. In Section 3.2, we give an overview of how these different approaches have been evaluated in literature.

3.1. Resource Demand Estimation

In this section, we first describe our procedure for the literature research. Then we introduce a common notation used for describing the approaches to resource demand estimation. Afterwards, the approaches to resource demand estimation are illustrated.

Literature Research

The goal of the literature research is the identification and collection of approaches to resource demand estimation that have been published until today. The results of this literature research are the foundation for the classification scheme described in Section 4. In a first step we used common scientific search engines, such as scholar.google.com, portal.acm.org and citeseerx.ist.psu.edu, to get a first set of candidate articles on the topic. We only considered scientific articles published in journals and conference or workshop proceedings. In a second step, we analyzed the bibliographies of the articles found in the previous step. We repeated this step until no additional articles were found. We then collected the authors of the articles found and systematically scanned their publication lists. Finally, we searched the proceedings of conferences well-known in the performance modeling community. The conferences and workshops we considered are listed in Table A.1 on page 97.

There are several terms that are commonly used in literature instead of resource demand estimation. Some authors speak of estimating service demands, service times or service requirements. Additionally, resource demand estimation is also used as part of more general activities, such as workload characterization, parameter estimation and model calibration. We used these terms as keywords when searching for articles on the topic of resource demand estimation.

We focused our literature research on approaches that depend only on measurements that are obtained from a running system without intrusive application instrumentation. The

reason for this decision is that we are mainly interested in estimation approaches that are also applicable to live production systems where the internals of the application of interest might be unknown. Therefore, we did not look at approaches that try to estimate resource demands with static code analysis of the application source code or bytecode, such as [Kap07], or that rely on profiling and application instrumentation, such as [BDIM04].

Notation

We use the following notation for the subsequent description of approaches to resource demand estimation. We denote resources with the indexes i, j and workload classes with the indexes c, d. The following parameters are used:

- $D_{i,c}$ is the average resource demand of requests of workload class c at resource i,
- $U_{i,c}$ is the average utilization of resource i due to requests of workload class c,
- U_i is the average total utilization of resource i,
- λ_c is the average arrival rate of workload class c,
- X_c is the average throughput of workload class c,
- R_c is the average response time of workload class c,
- $A_{i,c}$ is the average queue length of requests of workload class c seen on arrival of a new request at resource i,
- I is the number of resources,
- and C is the number of workload classes.

Matrices and vectors are written in bold letters. We use capital letters for matrices, e.g., $\mathbf{R}$, and small letters for vectors, e.g., $\mathbf{r}$.

We assume that the *Flow Equilibrium Assumption* [MG00, p. 280] holds, i.e., that over a sufficiently long period of time the number of completions is equal to the number of arrivals. As a result, the arrival rate λ_c is equal to the throughput X_c. Therefore, arrival rate and throughput are interchangeable in the following descriptions.

3.1.1. Service Demand Law

The Service Demand Law states that the resource demand of workload class c at a resource i can be derived from the utilization of resource i due to workload class c and the throughput of workload class c [MDA04, p. 65f]. See Equation 2.3 on page 7 for details. We can use this relationship to determine resource demands based on utilization and throughput data measured at a real system. However, we need to measure the utilization separately for each work class. In most cases we only get the total utilization of a resource and must estimate the per-class utilization then.

Kounev identifies two approaches to estimating resource demands using the Service Demand Law [Kou05, p. 135ff] :

- If only requests of one workload class arrive at the system of interest during a distinct time interval, we can approximate the utilization due to this workload class by the total utilization of the resource.
- In cases where requests of different workload classes arrive at the system of interest simultaneously, we need to apportion the utilization between the different workload classes appropriately.

The former approach is usually only applicable in dedicated experiment environments where we can control the workload. In most cases the workload in a production environment consists of a mix of requests of different workload classes. Then we can only use the latter approach.

Different methods for apportioning the total utilization between workload classes have been proposed. Lazowska et al. [LZGS84, p. 285ff] describe a method based on per-class utilization measurements from an accounting monitor. Since accounting monitors fail to capture the system overhead caused by requests of a workload class, they discuss different ways to distribute the unattributed CPU busy time between different workload classes. In [MAD97, MA98, MDA04], Menascé discusses further methods for apportioning based on additional metrics collected by the operating system. Brosig et al. use in [BKK09] an ad-hoc method to partition the total utilization U_i of resource i based on weighted response time ratios. They estimate the per-class utilization from the total utilization with the following relationship [BKK09]:

$$U_{i,c} = U_i \cdot \frac{R_c \cdot \lambda_c}{\sum_{d=1}^{C} R_d \cdot \lambda_d}. \tag{3.1}$$

It is implicitly assumed that the measured response time R_c of workload class c is approximately proportional to the corresponding resource demands of this workload class, i.e., the response time R_c does not include any significant time spent at other resources than the one considered.

3.1.2. Approximation with Response Times

As described in Section 2.1 the response time of a request at a queue is the sum of the queueing delay and the resource demand. If we assume that there are no queueing delays and the response times do not include significant time spent at other resources, the response time is equal to the resource demand. In general, if queueing delays are significantly smaller than the resource demands, we can approximate the resource demands with the observed response times. Obviously, this approximation only works if the resource utilization and respectively the queueing delays are low.

Two different variants of the response time approximation approach can be found in literature. Nou et al. [NKJT09] continuously measure the response time of each request and use the minimum observed response time in a period of time as an estimate of the resource demand. They assume that the actual resource demands have a small variance. In [UPS+07] and [BKK09] the average response time over a period of time is used as an estimate instead.

In cases where a system consists of multiple resources and we need to estimate resource demands for each resource separately, we need extensions to the previously described approach to resource demand estimation. Urgaonkar et al. [UPS+07] apply the response time approximation approach when modeling a multitier internet application. They start with the last tier, which does not call any external services on other tiers, and approximate the resource demands of this tier with the response times measured at the tier. Then they determine for each tier the time it waited for results from external services on other tiers. The resource demand of the tier is the observed response time minus the time waiting for other services. Nou et al. [NKJT09] try to break down the response times into the time waiting for external services and the time using the resource for calculations. They use utilization measurements to estimate the time needed to invoke external services.

3.1.3. Linear Regression

A classic way to infer resource demands is based on regression analysis [BS78, RV95, KPSCD09, PSST08, CCT07, ZCS07, KZ06, SKZ07]. It was used early in [BS78] to estimate the CPU utilization due to operating system supervisory calls on mainframe computers. Later, Rolia et al. [RV95] discussed its use to estimate the resource demands of application services.

The linear model of the linear regression is usually based on the Utilization Law (see Equation 2.2 on page 7). Given a workload consisting of multiple workload classes, the model can be described by the following relationship:

$$U_i = \sum_{c=1}^{C} \lambda_c D_{i,c} + U_{i,0}. \tag{3.2}$$

The intercept term $U_{i,0}$ can be used to estimate the utilization due to any kind of background work in the system that is not described by a workload class. It is optional and can be left out. Pacifici et al. [PSST08] encountered problems estimating $U_{i,0}$ in some cases. Therefore, they recommend to remove the intercept term from the model [PSST08].

In order to solve the model with linear regression, we need to obtain at least N simultaneous measurement samples of the aggregate utilization U and the arrival rate λ_c of each workload class from the system. N must be greater than the number of resource demands we want to estimate. With these measurements the matrices for Equation 2.6 on page 8 can be initialized in the following manner: The dependent response vector contains the measurements of the total utilization and the control matrix contains the arrival rates for each workload class.

Commonly, LSQ regression is used to get resource demand estimates from the previously described linear model. However, when using LSQ regression for resource demand estimation, several issues can arise:

- LSQ regression assumes that the resource demands $D_{i,c}$ are constant. However, in reality the resource demands are stochastically distributed. Depending on the actual form of the distribution of the resource demands the estimation errors differ significantly [RV95].
- It is implicitly assumed that there are no close correlations between control variables. If there is no linear relation between control variables, they are called orthogonal. However, this is often not the case. Slightly related variables do not influence the regression analysis significantly. If two or more control variables have a close linear relation, we speak of multicollinearity or collinear data [CP95, p. 183]. Multicollinearity causes non-unique and unstable solutions and should therefore be avoided [PSST08]. In [PSST08] the authors identify two common reasons for multicollinearity: (a) Two or more requests are related to each other, e.g., the number of login requests usually approximately equals the number of logout requests. (b) If there is only little variation in the control variables, the measurements usually suffer from collinearity. Therefore, we need to make sure that the observation period is long enough to capture significant variances [PSST08, RV95].
- Outliers may disproportionally influence the resulting parameter estimates. Outliers may provide valuable insight, but they can just result from some erroneous measurements, too. Therefore, outliers should be detected and specially analyzed.
- We need to make sure that the system of interest is not modified during the observation period. Modifications, such as hardware or software upgrades, might influence

the resource demands. Such changes can be easily excluded in controlled environments. However, in production systems these changes are often not avoidable. In such cases the analysis should be done separately for periods with a stable system.

- Ordinary LSQ regression can yield a solution with negative parameter values. Negative resource demands are not allowed. Therefore, we should add an additional constraint that all entries in the resulting parameters vector must be non-negative.

There are a number of methods proposed in literature to avoid some of these issues. Pacifici et al. [PSST08] use ordinary LSQ regression, but include several ad-hoc techniques to reduce the influence of multicollinearity and insignificant or low contributing flows. Zhang et al. [ZCS07] use non-negative LSQ regression to avoid negative estimates. Stewart et al. [SKZ07] come to the conclusion that the estimates provided by LAD regression produce more accurate models than their LSQ counterparts. Furthermore, LAD regression is more robust to outliers [SKZ07]. In [CCT07, CCT08] the authors use robust regression techniques, such as Least Trimmed Squares (LTS) regression, to cope with outliers, discontinuities due to software or hardware upgrades, and multicollinearity. Similarly, Cremonesi et al. [CDS10] consider the application of a clusterwise regression algorithm to estimate resource demands in presence of software or hardware upgrades.

All of the previously described approaches based on linear regression use a model derived from the Utilization Law. In contrast, Kraft et al. [KPSCD09] propose an approach based on an alternative model using measurements of response times and queue length on arrival. They assume a closed QN, where the system of interest is represented by *one* M/M/1 queue with FCFS scheduling strategy [KPSCD09]. Then the mean response time of requests can be described in the single workload case with the following relationship from Mean Value Analysis (MVA):

$$R_i = D_i(1 + A_i)$$

With this relation, generalized to multiple workload classes and arbitrary resource demand distributions, they define a linear model that can be solved by linear regression. They use non-negative LSQ regression to obtain estimates of the resource demands.

3.1.4. Kalman Filter

Kalman filters can be used to estimate the hidden state of dynamic systems. A general description of Kalman filtering can be found in Section 2.2.2. Zheng et al. [ZYW+05, ZWL08] consider the application of Kalman filters to estimate hidden performance parameters of a time-varying software system from a series of measurements. They chose the Kalman filter due to its ability to adapt quickly to changes in parameter values over time.

When using Kalman filters for resource demand estimation we need to define a state and a measurement model that describe the dynamics of the system of interest and the measurement process. Various formulations of these models are possible. In [ZYW+05], the authors give some general guidelines on the structure of such models when tracking time-varying performance parameters of a software system. They also provide an example in which they estimate the resource demands of the systems in a typical three-tier application with a single-class workload [ZYW+05]. Kumar et al. [KTZ09] extend this work to cases with several workload classes. The following descriptions of the state and measurement model are based on the filter design described in [KTZ09].

If the system consists of a single resource, the system state x is a vector of the resource demand of each workload class. In the case of a workload with of three classes the system state x is defined as [KTZ09]

$$\mathbf{x} = \begin{pmatrix} D_1 & D_2 & D_3 \end{pmatrix}^T.$$

Assuming that there is no a-priori knowledge about the dynamics of the system state, a constant state model is used, i.e, the state transition model F_k is the identity matrix. The Equation 2.9 on page 8 is reduced to

$$\mathbf{x}_k = \mathbf{x}_{k-1} + \mathbf{w}_k.$$

The measurement model can be based on different kinds of measurements. In [ZWL08], Zheng et al. propose to use a subset of the following components for the measurement model:

- throughput,
- mean response time,
- mean utilization,
- or mean queue length.

In general, the number of linearly independent measurements included in the measurement model should be greater than the number of estimated parameters in order to obtain reliable estimates [ZWL08]. Furthermore, Zheng et al. recommend to include one measurement that is related to all estimated parameters [ZWL08]. According to these recommendations Kumar et al. define their measurement model based on utilization and response time measurements [KTZ09]. They describe their measurement model with the relation [KTZ09]:

$$\mathbf{z} = h(\mathbf{x}) = \begin{pmatrix} R_1 \\ R_2 \\ R_3 \\ U \end{pmatrix} = \begin{pmatrix} \frac{D_1}{1-U} \\ \frac{D_2}{1-U} \\ \frac{D_3}{1-U} \\ \frac{1}{P}\left(\lambda_1 D_1 + \lambda_2 D_2 + \lambda_3 D_3\right) \end{pmatrix}. \tag{3.3}$$

Based on the relation $R = \frac{D}{1-U}$, it is assumed that the observed resource can be modeled by a M/M/1 queue. The symbol P denotes the number of processors in a system, the utilization U is the average utilization of all CPUs. The measurement model is obviously of a non-linear nature regarding the resource demand parameters because of the utilization in the denominator of the response time relation. Therefore, the extended Kalman filter [Sim06] can be used for this estimation problem, which depends on the Jacobian matrix of the measurement model. The Jacobian matrix of the measurement model in Equation 3.3 is

$$\mathbf{H} = \frac{\partial h}{\partial \mathbf{x}} = \begin{pmatrix} \frac{\lambda_1}{P} \cdot \frac{D_1}{(1-U)^2} + \frac{1}{1-U} & \frac{\lambda_2}{P} \cdot \frac{D_1}{(1-U)^2} & \frac{\lambda_3}{P} \cdot \frac{D_1}{(1-U)^2} \\ \frac{\lambda_1}{P} \cdot \frac{D_2}{(1-U)^2} & \frac{\lambda_2}{P} \cdot \frac{D_2}{(1-U)^2} + \frac{1}{1-U} & \frac{\lambda_3}{P} \cdot \frac{D_2}{(1-U)^2} \\ \frac{\lambda_1}{P} \cdot \frac{D_3}{(1-U)^2} & \frac{\lambda_2}{P} \cdot \frac{D_3}{(1-U)^2} & \frac{\lambda_3}{P} \cdot \frac{D_3}{(1-U)^2} + \frac{1}{1-U} \\ \frac{\lambda_1}{P} & \frac{\lambda_2}{P} & \frac{\lambda_3}{P} \end{pmatrix}.$$

When using Kalman filters for resource demand estimation, several parameters need to be tuned for good filter performance. Especially, the covariance matrices for the process noise $\mathbf{Q}$ and the observation noise $\mathbf{R}$ are required. Additionally, the Kalman filter expects an initial estimate of the state vector $\hat{\mathbf{x}}_{0|0}$ and of the error covariance matrix $\mathbf{P}_{0|0}$. In [ZYW+05, ZWL08] the authors give recommendations on how to choose these values.

3.1.5. Optimization

In this section, we describe estimation approaches that are defined as optimization problems and solved with mathematical programming methods. The linear regression approaches in Section 3.1.3 are a special type of optimization problem where the objective

function has a well-defined structure. However, as they are very common in resource demand estimation, they are described separately in Section 3.1.3. The estimation approaches described here are based on more general objective functions.

The definition of an optimization problem for resource demand estimation can be derived from queueing models with basic operational laws of queueing theory described in Section 2.1. Zhang et al. describe in [ZXSI02] how an optimization problem for resource demand estimation can be derived from an open queueing model consisting of one queue with PS scheduling and another queue with FCFS scheduling. They chose this example queueing model as they consider it to be representative of typical multi-tier server applications [ZXSI02]. However, they also state that their approach can be generalized to more complex models with various scheduling strategies [ZXSI02]. Their objective function minimizes the weighted sum of the response time errors [ZXSI02], i.e.

$$\min \sum_{c=1}^{C} p_c (R_c - \tilde{R}_c)^2 \tag{3.4}$$

$$\text{where } p_c = \frac{\lambda_c}{\sum_{d=1}^{C} \lambda_d}. \tag{3.5}$$

$\tilde{R}_c$ denotes the measured response time, R_c is calculated from the measured arrival rates and the utilization. Zhang et al. weigh the response time error with the proportion of requests of workload class c. The objective function is independent from the structure of the queueing model. Furthermore, a set of constraints are necessary. The shape of these constraints depends on the structure of the queueing model. In general, these constraints can be derived from basic queueing theory relationships, such as those described in Section 2.1. Zhang et al. provide examples for such constraints in [ZXSI02]. In their example the constraints are all linear. Therefore, they can use Quadratic Programming (QP) algorithms to obtain a solution of the optimization problem.

The previously described queueing network optimization approach to resource demand estimation lays the foundation for several following articles [LXMZ03, WXZ04, LWXZ06, KZT09], which extend this approach in different directions. In all of those subsequent articles the objective functions also minimizes the utilization errors in addition to the response time error. The new objective function is then defined as [LWXZ06]

$$\min(\sum_{c=1}^{C} p_c (R_c - \tilde{R}_c)^2 + \sum_{i=1}^{I} (U_i - \tilde{U}_i)^2). \tag{3.6}$$

$\tilde{R}_c$ denotes the measured response time, $\tilde{U}_i$ the measured utilization. The factor p_c is the same as defined in Equation 3.5. In [LXMZ03] and [KZT09] the authors use a variation of this objective function where the sums of absolute errors are replaced by sums of relative errors. The approach described by Liu et al. [LXMZ03] and Kumar et al. [KZT09] is called "Inferencing" and is implemented in the context of the AMBIENCE tool [LXMZ03]. "Inferencing" provides a set of constraints applicable to QNs consisting of several M/M/1 queues with PS scheduling strategy and supports multi-class workloads. Kumar et al. describe in [KZT09] an enhanced version of "Inferencing" that supports the estimation of load-dependent resource demands. The enhanced version requires a-priori knowledge of the type of function, e.g., polynomial, exponential or logarithmic, that best describes the relation between workload and resource demand. Then it returns estimates for the parameters of these functions. In [WXZ04] and [LWXZ06], the authors describe a method to improve robustness of approaches based on optimization problems with a similar structure as Equation 3.6. They call their method self-adjusting nested estimation procedure [LWXZ06]. It is based on a series of experiments, each producing a complete set of

measurements required to get a solution of the optimization problem. In each experiment, they consider not only the current optimal solution but the set of all optimal solutions since the first experiment [LWXZ06]. Therefore, it is more robust to noisy measurements and outliers [LWXZ06].

Menascé describes an alternative optimization problem for resource demand estimation in [Men08]. In contrast to the previously described approaches, this approach depends only on response time and arrival rate measurements. It does not require any utilization measurements. It minimizes the squared sum of response time errors. The optimization problem is described as follows [Men08]:

$$\min \sum_{c=1}^{C} (R_c - \tilde{R}_c)^2 \text{ with } R_c = \sum_{i=1}^{I} \frac{D_{i,c}}{1 - \sum_{d=1}^{C} \lambda_d D_{i,d}} \tag{3.7}$$
$$\text{subject to } D_{i,c} \geq 0 \quad \forall i, c \text{ and } \sum_{c=1}^{C} \lambda_c D_{i,c} < 1 \quad \forall i.$$

This formulation is based on the solution of an open multi-class queueing model. Since the resource demand parameter is in the numerator and denominator of the fraction, the objective function is non-linear. Therefore, non-linear programming algorithms are required to find solutions of this optimization problem. In [Men08], Menascé uses a numerical solver provided by Microsoft Excel for this task. Generally speaking, only local solution algorithms are available for this kind of problem, i.e., a starting point is given and the algorithm iteratively searches for a feasible solution in the proximity. Such algorithms might yield only a local optimizer of the problem.

In the most general formulation of the optimization problem, as used in [Nor09], there are only two constraints. The first constraint ensures that all resource demands are always non-negative, the second one states that the utilization of each resource is always below a hundred percent. If some of the resource demands are known a-priori, these can be included in the optimization problem as additional constraints [Men08]. This additional knowledge might improve the quality of the solutions.

Menascé describes in [Men08] a recursive estimator based on the optimization problem definition in Equation 3.7. The recursive estimator is defined as [Men08]

$$\mathbf{D}_k = f(\boldsymbol{\lambda}_k, \mathbf{r}_k, \mathbf{D}_{k-1}).$$

At each step k, the matrix $\mathbf{D}_k$ is calculated by solving the optimization problem with the current measurement vectors $\boldsymbol{\lambda}_k$ and $\mathbf{r}_k$. The starting point of the iterative solution algorithm is the matrix $\mathbf{D}_{k-1}$ containing the estimates from the previous step. Additionally, a smoothing filter might be required to lower short-term variation in the estimated resource demands. The recursive estimator can be used for online estimation of resource demands [Men08].

3.1.6. Maximum Likelihood Estimation

In [KPSCD09] Kraft el al. develop a resource demand estimation approach base on Maximum Likelihood Estimation (MLE) using measured response times and queue lengths that were seen on arrival of a request. In general, MLE allows us to infer statistics of a random variable by determining the probability of observing a sample path [KPSCD09]. In the context of resource demand estimation, we want to estimate the resource demand D_c of class c. We obtain N measurements of the response time and get a sequence $R^1, \ldots, R^N$. Then

we search for the resource demands $D_1, \ldots, D_C$ so that the probability of observing the measured response times is maximized. The *likelihood function* is defined as [KPSCD09]

$$\mathbb{L}(D_1, \ldots, D_C) = \sum_{k=1}^{N} \log \mathbb{P}[R^k \mid D_1, \ldots, D_C]. \tag{3.8}$$

The result of this function can be interpreted as the likelihood of observing a sample sequence with the given parameter values. The corresponding maximization problem is then formally defined as [KPSCD09]

$$\max_{D_1,\ldots,D_C} \mathbb{L}(D_1, \ldots, D_C). \tag{3.9}$$

In order to efficiently solve this problem, we need an expression of the likelihood function that is analytically differentiable. In [KPSCD09] Kraft et al. describe how to derive such a likelihood function. The actual representation of the likelihood function depends on the distribution of the resource demands. With the likelihood function we can determine the gobal maximum of the likelihood function and thus get values for the resource demands $D_1, \ldots, D_C$ that explains the measured response times best.

3.1.7. Independent Component Analysis (ICA)

As described in Section 2.2.4, ICA is a method to solve the "blind source separation" problem, i.e., to estimate the individual signals from a number of aggregate measurements. Sharma et al. describe in [SBC+08] a way to map the "blind source separation" problem to resource demand estimation. They use a linear model based on the Utilization Law (see Equation 2.2 on page 7). In matrix notation the model is [SBC+08]

$$\mathbf{D}\mathbf{\Lambda} + \mathbf{E} = \mathbf{U}. \tag{3.10}$$

If there are I resources, C workload classes and N measurement samples, then

- $\mathbf{D}$ is a $I \times C$ matrix containing the resource demands of each resource and each workload class,
- $\mathbf{\Lambda}$ is a $C \times N$ matrix containing the arrival rates of each workload class and each measurement interval,
- $\mathbf{E}$ is a $I \times N$ matrix consisting of constant terms capturing any non-linear components and noise,
- and $\mathbf{U}$ is a $I \times N$ matrix with the utilization measurements of each resource and each measurement interval.

Given this model, the arrival rates $\mathbf{\Lambda}$ can be interpreted as the sources, $\mathbf{D}$ as the mixing matrix and $\mathbf{U}$ as the aggregate observations [SBC+08]. Then ICA can provide estimates for $\mathbf{\Lambda}$ and $\mathbf{D}$ solely based on utilization measurements of I resources.

In order to be able to apply the ICA approach to resource demand estimation certain assumptions must hold [SBC+08]:

- The number of workload classes is limited by the number of observed resources, i.e., the constraint $C \leq I$ must hold.
- The columns of the arrival rate matrix $\mathbf{\Lambda}$ need to be statistically independent.
- The arrival rates are assumed to be non-Gaussian.
- The measurement noise is assumed zero-mean Gaussian.

ICA does not only provide estimates of resource demands, it also automatically categorizes requests into workload classes. This can significantly simplify the creation of performance models. However, the resulting workload classes are hard to interpret. It is not obvious which requests map to which of the workload classes automatically determined by ICA. This might hinder successive performance analysis based on the results from ICA.

3.2. Related Work

The approaches to resource demand estimation described in the previous section have been evaluated by various authors so far. The goals of these evaluations are diverse and their scope varies. In the following, we give an overview of how different approaches to resource demand estimation have been evaluated in literature so far.

Early work on the evaluation of linear regression in the context of resource demand estimation was done by Rolia and Vetland in [RV95] and [RV98]. They used a simulator modeling a single server to analyze the impact of various factors on the estimation error. They show that the number of workload classes and the resource demand distribution have the biggest influence on the estimation error [RV95]. They conclude that linear regression should only be used if the variation of the resource demands is low, because linear regression assumes constant parameters [RV98].

In [PSST08], Pacifici et al. consider the application of linear regression in cases where the resource demands are time-varying. They extracted traffic patterns from real-world systems and applied them to a micro-benchmarking servlet. They come to the conclusion that a number of practical issues, such as insignificant flows, collinear flows and background noise, need to be addressed before using linear regression [PSST08]. Furthermore, they show that in case of time-varying resource demands linear regression can only provide rough estimates.

The robustness of linear regression under various conditions was evaluated by Casale et al. in [CCT07, CCT08]. The authors come to the conclusion that robust regression techniques can help with many issues occurring in real environments, such as outliers, workload collinearity and changes in resource demands after software or hardware upgrades. In [SKZ07] the authors conclude that LAD regression provides better estimates in the presence of outliers than ordinary LSQ regression.

The applicability of Kalman filters to resource demand estimation was evaluated by Zheng et al. in [ZYW+05, ZWL08] with a single workload class. They ran experiments to assess the ability of the Kalman filter to dynamically adapt its estimates to deterministic and random changes in resource demands. They conclude that the Kalman filter can adequately track changes in resource demands if the changes are not very sensitive [ZYW+05]. Furthermore, they ran a set of experiments to determine the influence of the assignment of the process noise covariance matrix $\mathbf{Q}$ and the measurement covariance matrix $\mathbf{R}$, and the measurement interval length. They come to the conclusion that these parameters can be derived from information about the system with reasonable efforts [ZYW+05]. Kumar et al. extend the evaluation of Kalman filters to cases with multiple workload classes in [KTZ09]. They come to the conclusion that the Kalman filter has convergence problems in the cases with of multiple workload classes because of an underdetermined equation system [KTZ09].

There are also articles that compare the accuracy of different approaches to resource demand estimation. In [KPSCD09], Kraft et al. compare their linear regression and maximum likelihood approaches based on response times to LSQ regression based on utilization measurements. They used traces from a queueing network simulator and from experiments on a real system for their evaluation. They come to the conclusion that the maximum

likelihood approach provides the best accuracy in all their experiments under different utilization levels, number of workload classes and number of measurement samples [KP-SCD09]. In [SBC+08], the ICA approach is compared to LSQ regression. The experimental results show that the estimation accuracy of ICA is close to that of LSQ regression, but it is generally a bit lower [SBC+08].

3.3. Concluding Remarks

Resource demand estimation is an active field of research. Over the years a number of different approaches have been proposed. As shown in this chapter, these approaches differ in the models they are based on and in the mathematical methods they use for estimation. These diverse models and estimation methods imply different assumptions about the structure of the system of interest and its workload. When using one of the approaches to estimate resource demands, we must bear in mind that the underlying assumptions of the approach are fulfilled in the current context of application. We must choose an approach that fits the system of interest best in order to get reliable and accurate estimates of its resource demands.

The related work presented in the previous section provides an overview of the existing evaluation of the different approaches to resource demand estimation. In many cases the evaluation only comprises experiments with a single approach. A few authors also compare one approach to another. However, we could not find an evaluation that systematically compares several approaches to resource demand regarding different aspects. Such an evaluation would help to select an appropriate approach to resource demand estimation that fits best in a given application scenario.

4. Classification Scheme

In our literature research on resource demand estimation, we identified commonalities and variation points between different estimation approaches. In this chapter, we describe a classification scheme for approaches to resource demand estimation that incorporates the findings of the literature research. The goal of the classification scheme is to help

- to systematically choose an adequate estimation approach in a concrete application scenario,
- to characterize the state-of-the-art and identify possible future research directions in the field of resource demand estimation,
- and to determine starting points for the subsequent evaluation.

The classification scheme consists of a set of dimensions. Each dimension describes a major aspect of an approach to resource demand estimation. There are continuous and discrete dimensions. Discrete dimensions might be further divided into categories with several characteristics.

We describe each dimension in a separate section. Each section consists of a short introduction why the dimension is necessary. Then a general description of the dimension follows. Afterwards, if appropriate, we categorize the existing approaches to resource demand estimation in the described dimension.

Table 4.1 gives an overview of all estimation approaches we consider here. We usually reference these approaches by the authors' names of the publication in which we found the estimation approach. Sometimes we need to reference a group of several estimation approaches. To simplify matters, we then use the more generic terms listed in the other columns of the table.

Feature Diagrams

In the following, we use feature diagrams to describe the characteristics of a dimension in the classification scheme. Feature diagrams were first introduced by Kang et al. in [KCH+90] to model commonalities and variability points of software products in a domain. A *feature diagram* consists of a tree, in which nodes correspond to features and edges describe the compositions of these feature. A *feature* is defined as

> "a prominent or distinctive user-visible aspect, quality, or characteristic of a software system or systems" [KCH+90].

Features can be *mandatory*, which is expressed by a filled circle above the feature, or they can be *optional*, recognizable by a hollow circle. Each feature can be composed of a set of subfeatures. Decomposition edges define the relationship between subfeatures and the parent feature. *And-decomposition* edges indicate that all subfeatures should be present. *Alternatives (xor-decomposition)* indicate that only one of the subfeatures should be present. *Or-decomposition* edges are used to express that at least one of the subfeatures needs to be present. The graphical notation of these elements is shown in Figure 4.1.

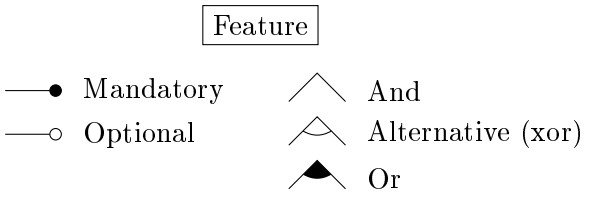


Figure 4.1.: Notation of feature diagrams.

In the classification scheme, a feature describes a characteristic or quality of an approach to resource demand estimation. The root feature corresponds to a dimension of the classification scheme. The subfeatures describe the characteristics of this dimension. Mandatory or optional elements indicate whether all or respectively only some of the estimation approaches possess this characteristic.

Class	**Variant**	**Approach (identified by authors)**
Response time approximation	Single resource	Brosig et al. [BKK09]
	Multiple resources	Nou et al. [NKJT09] Urgaonkar et al. [UPS+07]
Service Demand Law		Brosig et al. [BKK09] Lazowksa [LZGS84]
Linear regression	LSQ regression	Rolia et al. [RV95, RV98] Pacifici et al. [PSST08] Kraft et al. [KPSCD09]
	LAD regression	Zhang et al. [ZCS07, SKZ07]
	Robust regression	Casale et al. [CCT07, CCT08]
	Clusterwise regression	Cremonesi et al. [CDS10]
Kalman filter	Single workload class	Zheng et al. [ZYW+05, ZWL08]
	Multiple workload classes	Kumar et al. [KTZ09]
Optimization		Menascé [Men08] Zhang et al. [ZXSI02]
	"Inferencing"	Liu et al. [LXMZ03, WXZ04, LWXZ06]
	"Enhanced Inferencing"	Kumar et al. [KZT09]
MLE		Kraft et al. [KPSCD09]
ICA		Sharma et al. [SBC+08]
DEC		Rolia et al. [RKKD10, RKCD10]

Table 4.1.: List of approaches to resource demand estimation and corresponding references.

4.1. Input Parameters

All approaches to resource demand estimation require a set of input parameters. Obviously, the concrete set of input parameters differs depending on the underlying mathematical methods and the concrete implementation of an approach. However, we could identify a set of parameters that various approaches share with each other. The types of

parameters expected by an approach to resource demand estimation is one dimension in our classification scheme. We only look at types of input parameters that are directly related to the estimation approach. Input parameters specific to a concrete implementation of an estimation approach are not considered here.

Dimension Description

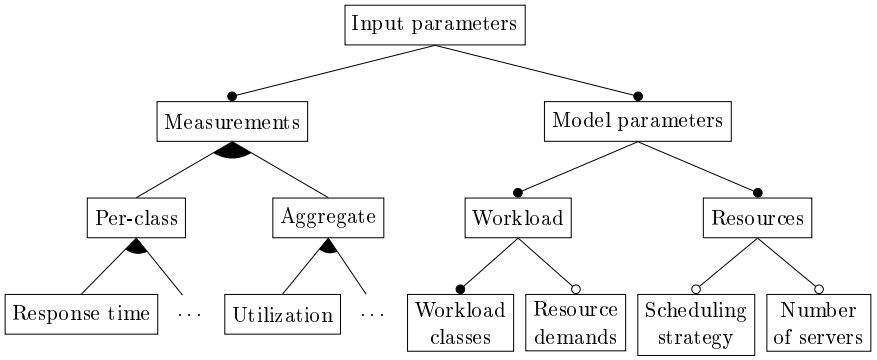


Figure 4.2.: Types of input parameters.

Figure 4.2 depicts the types of input parameters we identified as a feature diagram. The parameters can be put into two major classes: *model parameters* and *measurements*. In general, parameters of both classes are required. The model parameters provide information about the performance model for which we estimate resource demands. The values of these parameters need to be known beforehand. The measurements consist of a set of samples of different metrics. These samples need to be obtained from a running system, either a live production system or a dedicated and controlled test system. Depending on the situation only some specific metrics can be measured. For instance, the available monitoring tools might not support data collection of a metric or the measurement overhead of a metric might be unacceptable in some situations. Therefore, the required measurements are a criterion when selecting an estimation approach.

The measurements can be further grouped into *aggregate* and *per-class* measurements. Aggregate measurements are collected as totals over all workload classes that are processed at a resource. For instance, the operating system can usually report only the total utilization of a resource, and not the utilization due to requests of a specific workload class. Typical per-class measurements are

- response time (average or of individual requests),
- throughput,
- arrival rate,
- visit counts,
- and queue length on arrival.

Any combination of aggregate and per-class measurements is allowed. The response time needs to be further differentiated. It can either denote the response times of individual requests at a system or the average response time of all requests in specified time intervals. If a system is made up of several distinct resources, the response time, throughput and arrival rate metrics can be measured at different positions in the system. Then it is necessary to clearly define the scope of the metric, e.g., the response time of a request at a given resource or the response time of the whole system. If a system is in a steady state, the Flow Equilibrium Assumption [MG00, p. 280] holds and the throughput and arrival

rate are equal. However, we distinguish between these two metrics in the classification scheme, as this assumption might not always hold.

Since most approaches to resource demand estimation are based on statistical inference methods, the system of interest needs to be observed for some time, collecting several sets of measurement samples in order to obtain representative resource demand estimates. However, the measurement data collection can be time-consuming and costly regarding the monitoring overhead. Therefore, the number of measurements required by an estimation approach can be a crucial factor. Generally speaking, it is helpful if there is a method to determine bounds for the number of necessary measurement samples.

The model parameters describe the *workload* arriving at the system and the involved *resources*. The approaches to resource demand estimation commonly require at least information about the number of workload classes. Additionally, it is possible that we know some of the resource demands beforehand and can provide these values to the estimation approach in order to improve solution quality. In general, the estimation approaches also depend on some information about the resources for which we want to determine resource demands. We must at least provide the number of resources relevant to resource demand estimation. However, it might also be beneficial for estimation approaches to get additional information describing individual resources, e.g., the *scheduling strategy* and the *number of servers* of the queue representing the resource of interest.

Categorization of Existing Approaches

We analyzed all approaches to resource demand estimation that are listed in Table 4.1 and determined their input parameters. Table 4.2 contains an overview of the input parameters of each estimation approach. Parameters common to all estimation approaches, such as the number of workload classes and the number of resources, are not included in this table. Obviously, the required input parameters vary widely between different estimation approaches. The required measurements depend primarily on the statistical model behind the estimation approach. Estimation approaches expecting other combinations of input parameters might be possible as well. Especially, optimization based approaches give the freedom to adapt the underlying models, e.g., with additional constraints capturing knowledge about the system of interest.

Most linear regression approaches are based on the Utilization Law described in Section 2.1. They require the total utilization of a resource and the throughput of each workload class. In contrast, the linear regression approach Kraft et al. describe in [KPSCD09] depends on the measured response time and the queue length seen on arrival. For the Kalman filter, varying definitions are in use. Zheng et al. propose in [ZWL08] to use a subset of the following metrics to build a Kalman filter: throughput, end-to-end response time, total delay at a resource, total utilization of a resource or the mean number of requests at a resource. The ICA approach [SBC+08] only expects the total utilization of one or more resources. However, the number of workload classes for which it can estimate resource demands is limited by the number of observed resources. The DEC approach [RKKD10] is based on per-class throughput and visit counts for each resource. Additionally, it is assumed that a set of benchmarks stressing different parts of the system is available.

Some of the estimation approaches also depend partly on resource demands that are known beforehand. Menascé describes in [Men08] how to calculate one or more unknown resource demands from a set of given resource demands. The given resource demands can come from other estimation approaches or direct measurements. Another approach that expects resource demands for each workload class is described by Lazowska [LZGS84]. Lazowska assumes that the resource demands are measured with an accounting monitor. Such an accounting monitor, however, does not include the system overhead caused by a workload

Estimation approach	Measurements						Others	
	Utilization	Response time	Throughput	Arrival rate	Queue length	Visit counts	Resource demands	Scheduling strategy
Response time approximation								
- Brosig et al.		✗						
- Urgaonkar et al.		✗[a]				✗		
- Nou et al.	✗	✗						
Service Demand Law								
- Brosig et al.	✗	✗				✗		
- Lazowska			✗				✗[b]	
Linear regression								
- Kraft et al.		✗			✗			
- Other regression approaches	✗		✗					
Kalman filter								
- Zheng et al.	✗	✗						
- Kumar et al.	✗	✗		✗				
Optimization								
- Menascé		✗					✗[c]	
- Zhang et al.	✗		✗					✗
- "Inferencing"	✗	✗		✗		✗		✗
- "Enhanced Inferencing"	✗	✗		✗				✗
MLE		✗			✗			
ICA	✗							
DEC			✗			✗		

[a] Response time per resource.
[b] Measured with accounting monitor. System overhead is not included.
[c] A selected set of resource demands is known a priori.

Table 4.2.: Input measurements of estimation approaches.

class. The system overhead is defined as the work done by the operating system and that is caused by the processing of a request. Lazowska [LZGS84] describes a way to distribute unattributed computing time between the workload classes. Thus we can obtain improved resource demand estimates, which also include the system overhead.

Some of the optimization approaches require information about the scheduling strategy of the involved resources. These approaches are based on response time equations used in MVA. These response time equations depend on the scheduling strategy. There is a separate response time equation for each scheduling strategy. The optimization approaches described by Zhang et al. [ZXSI02], Liu et al. [LXMZ03, WXZ04, LWXZ06] and Kumar et al. [KZT09] automatically select the response time equation corresponding to the scheduling strategy of the resource.

In addition to the types of input parameters required by an estimation approach, some approaches also provide a rule of thumb regarding the number of required measurement samples. Approaches based on linear regression [RV95, KPSCD09, PSST08] need at least $K + 1$ linear independent equations to estimate K resource demands. When using robust

regression methods, significantly more measurements might be necessary [CCT07]. In [KZT09] Kumar et al. give a formula to calculate the number of measurements required by the "Enhanced Inferencing" approach. Yet, these are only minimum bounds for the number of measurements. A lot more measurements are typically required to obtain good estimates [SKZ07].

4.2. Output Metrics

Approaches to resource demand estimation are usually used to determine the mean resource demand of requests of a workload class at a resource. However, the estimated mean value is not sufficient in some situations. We may require more information about the confidence of the estimates and the distribution of the resource demands. The output metrics an estimation approach can provide might influence the decision for or against it.

Dimension Description

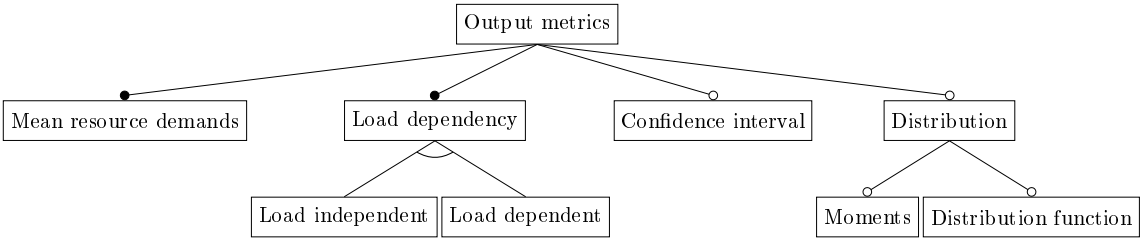


Figure 4.3.: Output parameter dimension.

Figure 4.3 depicts possible output metrics in a feature diagram. All approaches we considered provide point estimators of the mean resource demands. If the resource demand is load-independent, this is a constant value. Otherwise, it is a function that depends, e.g., on the arrival rates of the workload classes [KZT09].

However, we cannot generally rely on these mean values because they are only estimates of the real resource demands. Confidence intervals can help us to determine the reliability of the resource demand estimates. If a confidence interval is relatively wide, it might be necessary to obtain more measurements or to vary the workload in order to gain more confidence in the estimates [CP95]. A wide confidence interval can also indicate inappropriate model abstractions. Then a more fine-grained model might yield better results. Yet, a confidence interval cannot be used to determine the validity of a statistical model, i.e., if fundamental assumptions of an approach to resource demand estimation are not fulfilled, the resulting confidence intervals might be incorrect or misleading [RKKD10].

So far, we have assumed that the resource demands are deterministic and constant. However, this is often not the case in real computer systems [RKKD10]. Resource demands might depend on the data processed by an application or on the current state of the system or application [RV95]. In order to capture these non-deterministic characteristics in a performance model, we must also consider the distribution of resource demands. Estimates of higher moments of the resource demands or ideally the distribution function help in cases where resource demands cannot be assumed to be deterministic.

Categorization of Existing Approaches

All of the estimation approaches in Table 4.1 can estimate load-independent mean resource demands. The "Enhanced Inferencing" approach [KZT09] also supports the estimation of load-dependent resource demands: The resource demand can be either a polynomial,

exponential or logarithmic function of the arrival rate. The degree of the function is predefined. Then the estimation approach estimates the parameters of the load-dependent function.

There are standard formulas to compute confidence intervals for linear regression. For details see [CP95]. When using these formulas for LSQ regression, it is assumed that resource demands are deterministic and the errors are normally distributed [RKKD10]. If these assumptions do not hold, resulting confidence intervals might not be reliable. The DEC approach [RKKD10] also includes the calculation of confidence intervals, providing more robust intervals than linear regression [RKKD10].

None of the estimation approaches in Table 4.1 supports the estimation of distribution functions of resource demands. However, there are estimation approaches that can calculate higher moments of the resource demand estimation, e.g., the variance. Both the MLE [KPSCD09] approach and the optimization approach described by Zhang et al. [ZXSI02] are capable of provide estimates of higher moments. Obviously, this additional information comes at the cost of a higher number of required measurements.

4.3. Robustness

It is usually not possible to control every aspect of a system while collecting measurements due to the inherent complexity of modern computer systems. This is true both for controlled test environments and especially for production systems. This can lead to anomalous behavior in the measurements [CCT07]. In general, approaches to resource demand estimation should be as robust as possible regarding such anomalous behavior in the measurement process.

Dimension Description

The authors in [CCT07, CCT08] and [PSST08] identified the following issues with real measurement data:

- presence of outliers,
- background noise,
- configuration discontinuities,
- collinear workload,
- and insignificant flows.

Background and secondary activities can have two effects on measurements: outliers and background noise [CCT07]. Background noise is created by secondary activities that utilize a resource only lightly over a long period of time. Outliers result from secondary activities that stress a resource highly for a short period of time. Outliers can have a significant impact on the parameter estimation resulting in biased estimates [CCT07]. Different strategies are possible to cope with outliers. It is possible to use special filter techniques in an upstream processing step or to use parameter estimation techniques that are inherently robust to outliers. However, outliers may provide significant information, e.g., they may result from exceptionally complex requests. In this case the measurement sample should not be ignored. It might indicate that the performance model needs to be refined further.

Changes to the software and hardware configuration can be avoided in a controlled environment, but in production environments such changes may occur, possibly resulting in discontinuous changes of some resource demands [CCT07]. For instance, a software

update of the operating system might significantly influence the demands of certain functions of an application if it uses updated operating system procedures. If we attempt to describe measurement data with discontinuous changes with one linear model, the estimated resource demands might have significant errors. Then it is better to describe the measurements with two separate linear models, one before the discontinuity and one after it [CCT07, CCT08].

Another challenge for estimation approaches are collinearities between the throughput of workload classes. There are two possible reasons for collinearities in the workload: low variation in the throughput of a workload class or dependencies between workload classes [PSST08]. For example, if we model *login* and *logout* requests each with a separate workload class, these classes are probably dependent. The number of logins usually approximately matches the number of logouts. Collinearities in the workload may have negative effects on the estimates. A way to avoid these problems is to detect and combine workload classes that are dependent [PSST08].

Insignificant flows are caused by workload classes with very small arrival rates. Pacifici et al. [PSST08] experience numerical stability problems with their linear regression approach when insignificant flows exist. In order to avoid numerical instabilities, it might be necessary to remove insignificant flows before the measurement data is used for resource demand estimation [PSST08].

Categorization of Existing Approaches

In the following, we describe how various authors addressed the issues described above. However, we must keep in mind that some approaches to resource demand estimation are inherently not prone to some of these issues. For instance, response time approximation is evidently not influenced by collinear workload because it does not depend on throughput or arrival rate measurements.

Generally speaking, ordinary LSQ regression is prone to outliers. Stewart et al. come to the conclusion that LAD regression is more robust to outliers than LSQ regression [SKZ07]. Robust regression techniques as described by Casale et al. [CCT07, CCT08] try to detect outliers and ignore measurement samples that cannot be explained by the regression model. Liu et al. [LWXZ06] also include an outlier detection mechanism in their estimation approach based on optimization.

Robust and clusterwise regression approaches are proposed in [CCT07, CCT08, CDS10] to detect software and hardware configuration discontinuities. If such discontinuities are detected, the resource demands are estimated separately before and after the configuration change. Approaches based on Kalman filters [ZYW+05, ZWL08, KTZ09] are designed to estimate time-varying parameters. Therefore, they automatically adapt to the new resource demands after a software or hardware discontinuity.

Collinearities are one of the major issues when using linear regression [CP95]. A common method to cope with this issue is to check the workload classes for collinear dependencies before applying linear regression. If collinearities are detected, the involved workload classes are merged into one class. This is proposed in [PSST08, CCT07]. The DEC approach in [RKKD10] mitigates these dependencies, since it only estimates the resource demands for mixes of workload classes.

Pacifici et al. also consider insignificant flows in [PSST08]. They call a workload class insignificant, if the ratio between the throughput of the workload class and the throughput of all workload classes is below a given threshold. They completely remove insignificant workload classes in order to avoid numerical instabilities [PSST08].

4.4. Accuracy

When we use a model to predict the performance of a system, the accuracy of the resource demands has a crucial influence on the accuracy of the predictions. Different estimation approaches might yield varying accuracy in a given environment. A performance engineer must be able to determine the accuracy of candidate estimation approaches. If no past experiences are available, the performance engineer must devise experiments for his environment. Ideally, guidelines on the general accuracy of an approach under different conditions are generally available.

4.5. Resources

A performance model can capture the performance characteristics of a system at various levels of abstraction. Depending on the level of abstraction, different types of resources need to be considered. Each resource type has different characteristics. An approach to resource demand estimation needs to consider the characteristics of the resources for which resource demands are estimated.

Dimension Description

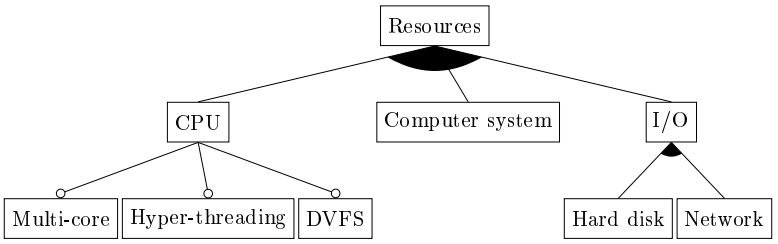


Figure 4.4.: Resource dimension.

The feature diagram in Figure 4.4 depicts the resource types we distinguish in this classification scheme. At a high level, *computer systems* as a whole, e.g., servers in a multi-tier infrastructure, are modeled with a single queue. If a more detailed performance model is required, *CPUs* and *I/O devices* are explicitly modeled as distinct queues.

If we model a computer system with a single queue, the resource demand of this queue must account for all kinds of delays, including CPU processing and I/O contention. The end-to-end response time of requests comprises all of these delays. Therefore, estimation approaches based on response times are usually better suited for the estimation of the resource demands of complete computer systems. Assuming that the response time is dominated by the CPU and delays incurred by other resources are insignificant, estimation approaches based on the total CPU utilization are also applicable to complete computer systems. In contrast, at the level of CPUs and I/O devices, we usually cannot determine response times for these resources. We know only the end-to-end response time of the whole computer system consisting of several CPUs and I/O devices. Estimation approaches that depend on response times directly measured at a resource are hard to use at this level of abstraction.

Modern CPUs have a number of features that can influence their performance characteristics significantly, such as multiple cores, hyper-threading and Dynamic Voltage and Frequency Scaling (DVFS). Multiple cores and hyper-threading are introduced to improve parallelism. Multi-core and hyper-threading CPUs may be represented by queues with several servers. However, this assumption does not always hold. For example, hyper-threading can have effects on utilization measurements and the factor by which the computing power

of a machine is increased can be significantly smaller than the number of cores [PSST08]. DVFS is used to improve the energy efficiency of CPUs by throttling their speed during phases of low utilization. CPUs with DVFS are commonly modeled by a queue with load-dependent service times.

Regarding I/O devices, we distinguish between *hard disks* and *network* devices. There are fine-grained analytical disk models available, which explicitly include fine-grained device characteristics, such as seek times, latency, caching and command queueing. See [Sch08] for an overview of fine-grained analytic disk models. Here we are interested in higher-level abstractions that consider the hard disk as a black box and are solely based on readily available measurements. If network delays are not negligible, we must estimate these delays, too, so that a performance model provides accurate predictions.

Categorization of Existing Approaches

Most estimation approaches we consider here are focused on the estimation of resource demands of CPUs or CPU-bound servers. Linear regression based on the Utilization Law can also be used to estimate I/O demands of hard disks [RV95]. However, corresponding evaluation in the context of I/O devices is missing. There is work to estimate network delays. However, network resources are not part of the scope of this thesis.

4.6. Virtualization

The current trend of virtualization adds additional complexity to the problem of resource demand estimation. The Virtual Machine Monitor (VMM) now resides between the physical hardware and a VM, in which the operating systems, middleware platforms and applications run. This new layer has significant influences on the measurement process and the estimation of resource demands.

Dimension Description

The VMM is responsible for mapping the different logical devices to physical ones in virtual environments. This mapping affects resource demand estimation in several ways:

- The VMM needs to translate between the logical resources and the physical ones. For instance, it may be necessary to translate certain guest CPU instructions into instructions of the host system [Nor09] or to channel data between virtual and real I/O devices [CG05]. This extra work causes an overhead at the VMM layer. The overhead needs to be charged to the VMs that caused it [CG05].
- The physical resources are shared between VMs. A scheduler is distributing the processing time between all VMs that require the resource. Therefore, a system might not have instant access to a resource because another VM is using it and must wait until the scheduler assigns the resource. That causes additional contention at the VMM layer, which needs to be taken into account when estimating resource demands [Nor09].
- Measurements in a VM are often less reliable. Clock synchronization issues and granularity may distort measurements [Nor09]. If the operating system does not know about the virtualization layer, utilization statistics can be wrong because the operating system might include the time a physical resource was dispatched to other VMs in the utilization calculation [Nor09].

When applying an approach to resource demand estimation in virtualized environments, we must therefore ensure that the measurements the estimation approach depends on are correct and reliable. Furthermore, we must consider how to charge the overhead at the VMM layer to each VM and how we cope with the additional contention at the VMM layer.

Categorization of Existing Approaches

There are only few documented experiences of resource demand estimation in virtualized environments. Norton [Nor09] evaluates the applicability of the Menascé optimization approach [Men08] in virtualized environments. He starts with utilization measurements, which might be unreliable due to influences of the virtualization layer, and calculates first estimates with the Utilization Law (see Equation 2.2 on page 7. Afterwards, the Menascé optimization approach is used to refine these initial estimates. The Menascé optimization approach only depends on response time and throughput measurements, which the author regards as more reliable than utilization measurements. The author comes to the conclusion that such a combination can improve the accuracy of resulting models in the presence of virtualized environments [Nor09].

4.7. Applicability

The applicability dimension comprises special constraints and characteristics of an approach to resource demand estimation limiting its field of application. We must pay attention to these applicability aspects when choosing an estimation approach.

Dimension Description

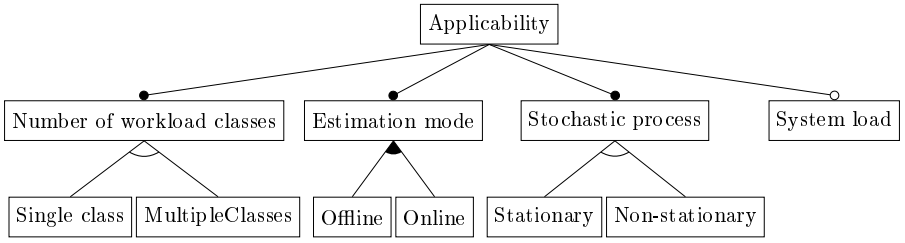


Figure 4.5.: Applicability dimension.

Figure 4.5 shows four categories of the applicability dimension:

Number of workload classes. Some estimation approaches can only provide estimates of a single workload class at a time. These approaches cannot cope with mixes of different workload classes. Either we merge all workload classes into a single class or we ensure that during a measurement period only requests from one workload class arrive at the system. However, if the workload consists of several workload classes, it is usually better to take an estimation approach that supports multiple classes.

Estimation mode. We can distinguish between two modes of resource demand estimation: *offline* and *online*. In the offline mode, measurement and estimation are temporally separated processes. First, we collect all necessary measurements and persist them. Afterwards, we estimate the resource demands based on all collected measurements. In contrast, online estimators collect regular measurements from the monitored system and instantaneously update their estimates as new measurement samples become available. This mode of estimation is usually used in production environments in order to obtain current estimates of resource demands, e.g., as input for capacity and performance management tasks.

Stochastic process. There are situations where resource demands cannot be assumed to follow a stationary process. For instance, adaptive software systems often reconfigure themselves in order to adapt to changes in the workload or the execution environment. In this case the resource demands are time-varying. Older measurements might not explain the current system state anymore. Accordingly, an approach to

resource demand estimation must attempt to adapt to these changes in the resource demands within an acceptable time interval.

System load. Depending on internal assumptions some estimation approaches yield a poor performance at certain utilization levels. This can result in very inaccurate or unstable estimates.

Categorization of Existing Approaches

The following approaches to resource demand estimation are only defined for a single-class workload. The Kalman filter definition described by Zheng et al. [WZL05, ZYW+05] is based on single-class equations. Kumar et al. extend this definition to multiple workload classes in [KTZ09]. However, they also come to the conclusion that if the system consists of several resources, the multi-class Kalman filter might be underdetermined and therefore might have a slow convergence. The robust regression method described by Casale et al. [CCT07, CCT08] is also defined only for single-class workloads, but should be generalizable to multi-class cases [CCT07].

Evidently, all of the estimation approaches can be used for offline estimation of resource demands. If we want to perform online estimation, recursive or incremental estimation approaches are generally better suited, as they can gradually improve estimates with additional measurements. A typical example of recursive estimators are Kalman filters [WZL05, ZYW+05, KTZ09], but optimization approaches are considered for online estimation as well [Men08].

In order to be able to adapt to non-stationary resource demands, we can usually adopt estimation approaches to include a sliding window or age discounting mechanism discarding older measurements. For instance, Pacifici et al. [PSST08] propose to extend linear regression with such a mechanism. However, they come to the conclusion that it can only provide rough estimates of time-varying resource demands. Kalman filters as proposed by Zheng et al. [ZYW+05] are a promising alternative for time-varying parameters [ZYW+05]. In general, a trade-off is required between how fast an estimator adapts to changes in resource demands and how strongly measurement errors influence the estimates [ZYW+05].

If we approximate resource demands with response times, e.g., described in [BKK09, UPS+07], it is important that the resource of interest is only lightly utilized. Otherwise, queueing delays become significant and the approximation is therefore inaccurate. If we have a system where phases of light load alternate with phases of higher loads, we might approximate the resource demands with the minimal observed response time as used by Nou et al. [NKJT09]. Estimation approaches using the response time equation (see Equation 2.4 on page 7) have problems if utilization converges to a hundred per cent, since the response time then tends to infinity. This is the case with most Kalman filter and optimization approaches.

4.8. Kind of Existing Evaluation

Existing evaluation results of approaches to resource demand estimation are a great help when selecting and applying an estimation approach. This dimension of the classification scheme describes how the estimation approaches have been evaluated in literature so far. Additionally, the categorization of existing approaches in this dimension provides support to identify gaps in the existing evaluation and to plan new evaluation scenarios closing those gaps.

Dimension Description

The resource demand values of a real system are usually not known. This makes it difficult to determine the accuracy of an estimation approach. We must address this challenge when creating evaluation scenarios. We identified the following common evaluation procedures to determine the accuracy of an estimation approach:

Simulation. We create a performance model of a real or hypothetical system. Common modeling notations are QNs, Layered Queueing Networks (LQNs) or QPNs. We parameterize the model with fixed values for the resource demands. These values are arbitrary and need not correspond to the resource demands at a real system. Then we solve the performance model analytically or with simulation in order to obtain traces of the metrics required by the estimation approach, e.g., response time and throughput. Afterwards we use these traces as input to our estimation approach and compare the resulting estimates to the actual values.

Artificial benchmarks. We can use artificial benchmarks that generate a defined load on a resource. For instance, the benchmark may perform time-consuming mathematical calculations to stress the CPU of a system. Each run of the benchmark corresponds to a request to the system. Furthermore, we must define an arrival process that drives the execution of the artificial benchmark. We can generate a synthetic arrival process, e.g. a Markovian process, or use arrival patterns of request observed at a real system. While running the benchmark, we monitor the system of interest. The resulting measurements are used for resource demand estimation. Then we can compare the estimated resource demands with the real ones.

Real applications. We can use measurement traces from real production systems or from experiments with application benchmarks, such as SPECjEnterprise2010, TPC-W or RUBiS. In this case we do not have any knowledge about the real resource demands and cannot determine the estimation error. Therefore, we build a performance model of the system and parameterize it with the estimated resource demands. We analyze it analytically or with simulation and compare the results, such as response time, utilization or throughput, with measurement data from the real system. We distinguish between two different ways of comparing measurements with predictions [SKZ07]:

- If we compare the results of the model with the measurement traces used for estimation, we get the *explanatory accuracy.*
- If we compare the results of the model with other measurement traces than those used for estimation, we get the *predictive accuracy.*

Obviously, if we only consider the explanatory accuracy, we cannot discover overfitted estimates, which might lead to wrong predictions.

Each of the mentioned evaluation procedures has its pros and cons. Simulation is easy to use and it is possible to clearly determine the influence of different factors on the estimates. However, it fails to capture the complexity and unforeseeable behavior of real environments and applications. Artificial benchmarks allow us to use real measurements for resource demand estimation and to know the real resource demands. Yet, the generated workload might not always be representative of real applications. Obviously, evaluation scenarios with real applications are desirable. However, the experiment setup for such evaluation scenarios is complex and it is not always possible to determine which factor has an influence on estimation accuracy.

Categorization of Existing Approaches

Early evaluation of linear regression models was done by Rolia and Vetland [RV95, RV98] with a simulator. The predictive accuracy of LSQ regression is considered by Zhang et

al. in [ZCS07] with experiments based on the TPC-W suite. Stewart et al. [SKZ07] compare LSQ regression with LAD regression by using measurement traces from production systems and from an application benchmark. They evaluate both the explanatory and the predictive accuracy. Pacifici et al. [PSST08] examine the accuracy of LSQ regression when estimating time-varying resource demands. They use a micro-benchmark experiment, where the resource demands can be controlled by parameters and use an arrival process based on arrival patterns at a real system. The accuracy of the response time based LSQ regression is studied by Kraft et al. [KPSCD09] with simulation. Casale et al. [CCT07, CCT08] evaluate robust regression approaches with measurement data from a real system. However, they only consider the explanatory accuracy.

The Kalman filter approach is evaluated by Zheng et al. in [ZYW+05, ZWL08] with a single workload class. The focus of this evaluation is on the tracking performance of the filter. The authors use a LQN simulator and simulate random and deterministic changes to measured parameters and assess how well the Kalman filter can cope with these variations. Kumar et al. [KTZ09] extend this work to scenarios with multiple classes. They use a micro-benchmarking servlet to determine the estimation accuracy while varying the real resource demands.

In [LWXZ06], Liu et al. evaluate their optimization based approach with data obtained from a real server environment. They consider the explanatory as well as the predictive accuracy of their model. Similarly, the optimization approach of Menascé [Men08] is also validated against data of real systems. The evaluation there focuses on the predictive accuracy. The approach called "Enhanced Inferencing" in [KZT09] is evaluated with data from a real system with synthetically generated workloads. The evaluation focuses on the explanatory accuracy of the approach.

Kraft et al. [KPSCD09] run simulation experiments to compare their MLE approach with several regression approaches. The ICA approach [SBC+08] is evaluated with an application benchmark and synthetically generated workload. The resulting estimates are compared to estimates obtained with linear regression. Rolia et al. use the TPC-W application benchmark to study their DEC approach [RKKD10] and compare the predicted demands with measurements and estimates obtained with LSQ and LAD regression.

4.9. Concluding Remarks

In this chapter, we described a classification scheme for approaches to resource demand estimation. The classification scheme shows that there are many possible variation points and differences between the estimation approaches. As a performance engineer it is important to choose an estimation approach that fits best in a given application scenario. The classification scheme can help in such situations as it enables the systematic comparison of the estimation approaches regarding specific dimensions.

The classification scheme also points out a number of possible future research directions. For instance, the high-level resource demand estimation of I/O devices might be worthwhile to investigate. Furthermore, existing approaches to resource demand estimation might need to be extended to reflect modern features of CPUs, such as DVFS and multi-core processors. Another major research topic is the estimation of resource demands in virtualized environments.

5. Evaluation Strategy

In this chapter, we describe our evaluation approach. We first state our evaluation goals and then derive a number of specific questions for the evaluation. Afterwards, we select a set of approaches to resource demand estimation regarding defined criteria. Finally, we provide an in-depth description of our evaluation scenarios.

5.1. Goals and Questions

The major goal of the evaluation is the *comparison of the accuracy of different estimation approaches under varying environmental conditions*. Environmental conditions are determined by the *workload* and the *resources* in a concrete application scenario. Starting with this statement of the evaluation goal, we derive a number of questions, which are a refinement of the evaluation goal considering specific aspects of it.

A set of questions aims at the evaluation of the influence of specific workload characteristics on the accuracy of the estimation approaches. In particular, we consider the following questions:

Q1: How does an increasing number of workload classes influence the accuracy of the estimation approaches?

Q2: In what ways are the estimation approaches influenced by the workload intensity?

Q3: Are estimation approaches negatively affected by multicollinearities in the arrival process?

Q4: In what ways are estimation approaches influenced by background jobs?

Q5: Are estimation approaches negatively affected by delays due to software or hardware contention?

The following question considers the applicability of the estimation approaches in real environments:

Q6: What is the accuracy of the estimation with realistic workloads?

The last two questions are aimed at specific resource characteristics of. Here, we consider the current trend towards multi-core processors and virtualized servers. The questions are:

Q7: In what ways are the estimation approaches affected by multi-core processors?

Q8: Are the estimation approaches applicable in virtualized environments?

These questions need to be answered in our evaluation. In the following, we design a set of evaluation scenarios addressing each of these questions.

5.2. Evaluated Approaches to Resource Demand Estimation

We will evaluate a subset of the approaches to resource demand estimation in Table 4.1 on page 26 regarding the questions mentioned in Section 5.1. The decision which estimation approaches we will evaluate is based on the following criteria:

Input measurements. The preferred types of input measurements are CPU utilization, average or individual response times and throughput. These measurements can be obtained easily with standard monitoring tools. Visit counts and queue lengths are not observed by the monitoring tools we plan to use. Additionally, the estimation approach must not depend on any known resource demands.

Applicability. In the evaluation, we want to estimate resource demands for several workload classes. The estimation approaches need to support multi-class workloads. Furthermore, they should be applicable to resources with PS scheduling strategy.

Support for online estimation. We are primarily interested in estimation approaches that support the online estimation of resource demands.

Implementation complexity. The implementation effort for the estimation approaches should be within reasonable limits regarding the time frame of this thesis. Ideally, there are ready-to-use libraries providing implementations of complete estimation approaches or parts of them.

Additionally, we ensured that at least one representative of each major mathematical method, namely linear regression, Kalman filtering and mathematical optimization, is included in the evaluation.

Estimation approach	References
Response time approximation	Nou et al. [NKJT09] and Urgaonkar et al. [UPS+07]
Service Demand Law	Menascé [MDA04] with the apportionment scheme of Brosig et al. [BKK09]
Linear regression	Rolia and Vetland [RV95]
Kalman filter	Kumar et al. [KTZ09]
Menascé optimization	Menascé [Men08]

Table 5.1.: List of evaluated estimation approaches including their main references.

Table 5.1 lists the estimation approaches we consider in the evaluation. We do not consider the Service Demand Law approach by Lazowska [LZGS84] because it requires resource demands as inputs. The regression approach based on response times and MLE proposed by Kraft et al. [KPSCD09] are not included as they are targeted for resources with FCFS scheduling. The implementation effort for the optimization approaches by Liu et al. [LXMZ03, LWXZ06], Zhang et al. [ZXSI02] and Kumar et al. [KZT09] is considered too high within this thesis. This can be done in the context of future work.

5.3. Experiment Environment

In this section, we describe the experiment environment we use for the evaluation. We require two separate environments for our experiments: a native and a virtualized environment.

5.3.1. Native Execution Environment

A pool of identical computers is available for the experiments. Depending on the scenario, we use a different number of computers. Table 5.2 lists the hardware configuration of the computers. The cores of the CPU can be individually activated and deactivated as required. All computers are connected by a network.

CPU model	Intel Core 2 Quad Q6600
Number of cores	4
CPU frequency	2.4 GHz
Memory	8 GB
Harddisk	2 x 500 GB SATA2 Disk
Network	1 Gbit/s

Table 5.2.: Hardware configuration of the experiment computers.

An Ubuntu 10.04 64-bit operating system is installed on each computer. We use the standard server kernel provided by Ubuntu in version 2.6.32. We removed all programs and services that are not needed for the experiments in order to avoid unexpected influences on the measurements. For measurement purposes the sysstat package, which includes the `sar` tool, is installed in version 9.0.6.

5.3.2. Virtualized Execution Environment

We conduct the experiments with virtualization on a SunFire X4440 x64 server. It has four processors with six cores each and 128 GB main memory. We chose the Citrix XenServer 5.5 as virtualization platform because of its significant market share [IDC10] and its free availability. The virtualization platform hosts two full-virtualized VMs with the hardware configuration listed in Table 5.3.

CPU model	Logical CPU
Number of cores	1
CPU frequency	2.4 GHz
Memory	4 GB
Harddisk	30 GB

Table 5.3.: Hardware configuration of the experiment VMs.

The VMs are connected by a virtualized network. We installed the same operating system and software in the VMs as in the native execution environment.

5.4. Evaluation Scenarios

This section contains a description of four evaluation scenarios. The scenarios are designed to provide answers to the questions in Section 5.1.

5.4.1. Scenario A: Artificial Workload

This scenario addresses questions **Q1** to **Q5**. To answer these questions, we require a controllable and predictable workload. This is usually not the case with real workloads, where the actual values of the resource demands are unknown and unwanted interactions with the environment cannot be excluded completely. Therefore, we use an artificial workload generated with the Ginpex framework [HKHR11].

First, we carry out a base experiment excluding any influences that might affect the estimation approaches negatively. We do several experiment runs while varying the utilization level between 20%, 40%, 60% and 80%, as well as the number of workload classes between 1, 2, 3, 4, 6, 8 and 16. This experiment is targeted at questions **Q1** and **Q2**.

Afterwards, we conduct another three experiments using variations of the base workload. These experiments are targeted at questions **Q3**, **Q4** and **Q5**. The utilization level and the number of workload classes is fixed at 40% and respectively 4 classes in these experiments .

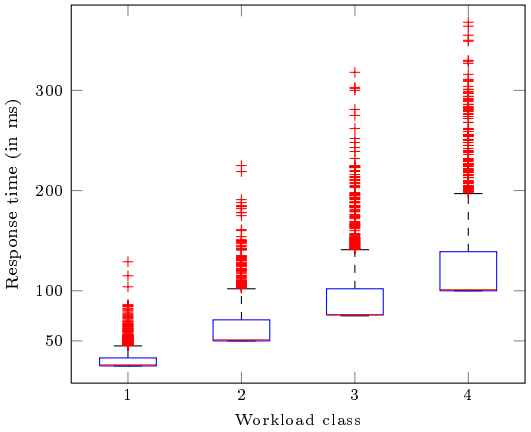


Figure 5.1.: Box plot of the response times of individual requests.

The estimation accuracy is determined by comparing the resource demands the Ginpex framework generated with the estimated resource demands of each estimation approach. We assume the generated resource demands of Ginpex to be of high accuracy. We validated this assumption by measuring the response time of individual requests. Figure 5.1 shows a box plot of the response times of individual requests at a utilization of 20% with four workload classes. The configured resource demands were 25 ms, 50 ms, 75 ms, 100 ms for workload class 1, 2, 3 and 4, respectively. The mean response time is in all cases close to the configured resource demand indicating that the generated resource demands are of high accuracy.

5.4.1.1. Workload

Ginpex measurement models describe the workload that is generated on the system of interest. A measurement model consists of different types of tasks. Tasks of the type `ResourceStrategyMeasurementTask` incur a certain load with a specified demand on a resource. We use these resource demanding tasks to model the processing of a request. The resource demand is generated by the calculating a Mandelbrot set. In this scenario, we use a base workload and several variations of it. These workloads are described in the following paragraphs.

Base Workload

The base workload models a system that processes requests of different workload classes. Each request requires a certain calculation time at the CPU of the system. The requests arrive continuously at the system with varying interarrival times.

Figure 5.2 illustrates the timing of the generated workload. Resource demanding tasks representing the processing of requests at the system alternate with pauses in which no calculations are done. The duration of the resource demanding tasks is constant over time,

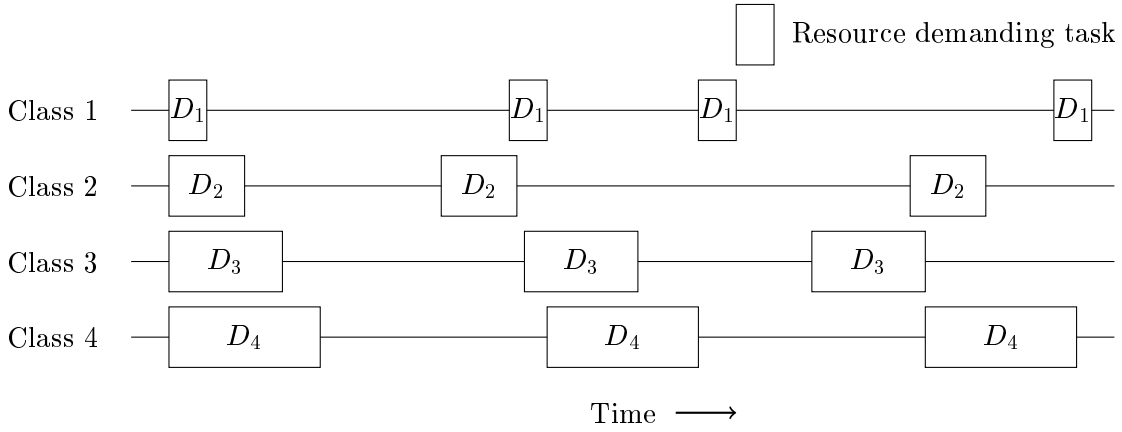
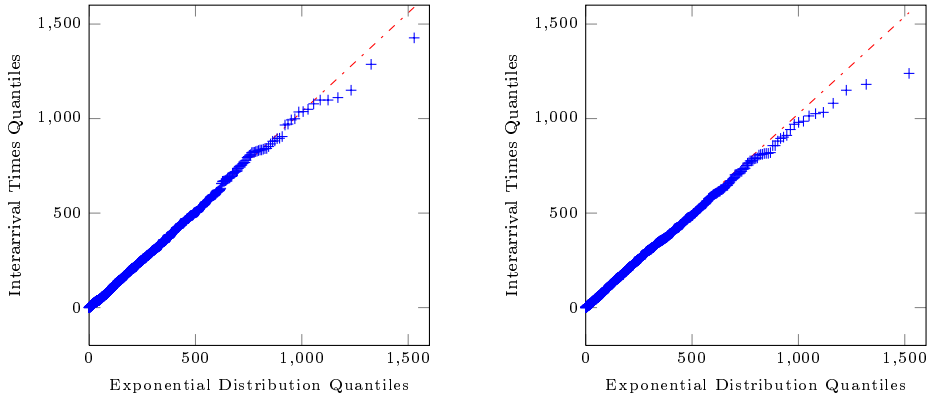


Figure 5.2.: Timing of the artificial workload.

whereas the duration of the pauses is randomly chosen from an exponential distribution. There is a separate loop of resource demanding tasks and pauses for each workload class. These loops run in parallel in order to avoid dependencies between the tasks of different workload classes resulting in collinearities in the throughput.

The mean interarrival time of each workload class is determined by the configured CPU utilization level. The durations of the generated resource demands depend on the workload class. The resource demand of workload class c is determined by the c-th element of the array $D = (25\text{ ms}, 50\text{ ms}, 75\text{ ms}, \ldots, 300\text{ ms}, 30\text{ ms}, 55\text{ ms}, \ldots, 105\text{ ms})$. The Ginpex framework provides sensors to measure the response times of each task. We use these sensors to measure the actual response times of the resource demanding tasks.

Since some of the estimation approaches assume that the interarrival times follow an exponential distribution, we have to ensure that the generated workload fulfills this assumption. However, this is not the case with the workload described before. The interarrival times of the requests of one workload class have a shifted exponential distribution because the minimum possible interarrival time is equal to the resource demand of the workload class. The next task can only be started if the previous one has finished.



(a) Workload class 1 (b) Workload class 2

Figure 5.3.: Q-Q plots of the interarrival time distributions.

We solved this issue by starting several threads for each workload class. Each thread executes its own loop alternating between resource demanding tasks and pauses. With ten threads per workload class, the interarrival times have approximately an exponential distribution. To check this, we carried out 28 experiment runs with varying numbers

of workload classes and at different utilization levels. We determined the coefficient of variation for each experiment run and each workload class. The observed coefficients of variation had a mean value of 1.0000 with a standard deviation of 0.0433. The minimum observed coefficient was 0.8703, and the maximum one was 1.1623. A coefficient of variation of 1 is a strong indicator for an exponential distribution. Additionally, we created a Q-Q plot comparing the quantiles of the observed interarrival time distributions with the quantiles of an ideal exponential distribution. If these plots are diagonal, the compared distributions are equal, i.e., the interarrival times follow an exponential distribution. We checked this visually for all experiment runs. Figure 5.3 shows a representative example in the case of two workload classes. In general, we observed a small deviation from the diagonal at higher quantiles, but the deviation is limited. Therefore, we consider it as an adequate approximation of an exponential interarrival time distribution.

Collinear Workload

This is a variation of the base workload where we deliberately included dependencies between tasks of different workload classes. The workload can be configured to run tasks of two or more workload classes always in sequence. As a result, we expect strong collinearities in the throughput of different workload classes.

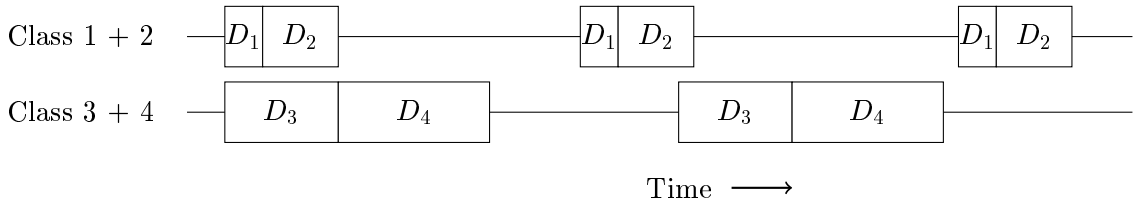


Figure 5.4.: Timing of the collinear workload.

Figure 5.4 shows the timing of the tasks if always two of four workload classes are dependent on each other. The two dependent classes always come directly after each other without a pause. We recognize that this is an extreme type of dependency. In real applications, one would expect that a certain time elapses between two dependent requests, e.g. the think time of a user. However, we chose this extreme case of dependency to get clear collinearities in the throughput so that the influence on the estimation approaches can be determined unambiguously. The correlation coefficient between the throughput of dependent workload classes is close to 1 as several validation runs showed.

Workload with Additional Wait Phases

This workload includes additional wait phases within the resource demanding tasks that represent delays due to software or hardware contention, e.g., getting a database connection from a pool or waiting for an I/O device to finish. The length of the additional wait phase is constant and can be configured.

Figure 5.5 illustrates the timing of the tasks including the additional wait phase. The wait phase is always in the middle of a resource demanding task, splitting it into two parts of equal duration. The resource demands used in this workload are 100 ms, 150 ms, 200 ms and 250 ms. We subtracted the duration of the additional wait phase from the interarrival times so that the utilization is not reduced by longer wait phases. The response time measurements are determined for the complete resource demanding task including the additional wait phase.

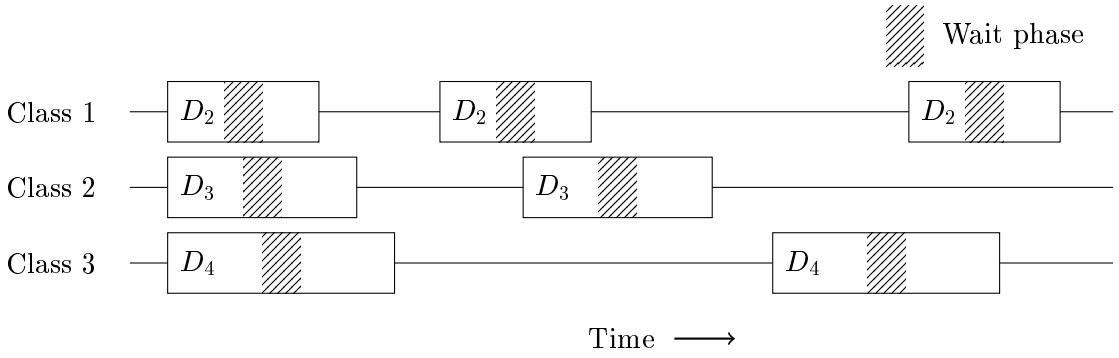


Figure 5.5.: Timing of the workload with additional wait phases.

Workload with Background Jobs

This workload contains an additional background job compared to the base workload. The background job is realized as an extra thread. The other threads of the base workload are not changed. The background job thread executes a loop that alternates between a resource demanding task of 20 ms and a constant wait phase whose duration is determined by the desired load level of the background job.

The duration of the background job is limited to five minutes. Then it pauses for another five minutes. This is repeated continuously. The thread mimics the behavior of a secondary activity that runs regularly in certain intervals in the system and causes a limited load over a longer period of time, e.g., system updates or hard disk checks.

5.4.1.2. Experiment Setup

The Ginpex framework requires two machines to execute measurement models:

- The artificial workload is generated on the *experiment machine*. We run the Ginpex load driver on this machine and perform the measurements.
- The *experiment controller* coordinates the experiment execution and collects the measurements after an experiment run.

Only one of the processor cores of the experiment machine is enabled in this scenario. The response times are measured with the sensors facility of Ginpex. The utilization is measured with the Linux monitoring tool `sar` with a sampling interval of 15 seconds.

Since the experiments are repeated for varying utilization levels and varying numbers of workload classes, we automated the experiment execution. Ginpex allows us to define custom experiments for this automation purposes by implementing a new experiment controller. We implemented custom experiment controllers for different experiments. These experiment controllers configure the experiment machine, start the load driver, generate a measurement model with one of the workloads in Section 5.4.1.1 and then run the generated model.

5.4.2. Scenario B: TPC-W Benchmark

In this scenario, we want to evaluate the estimation accuracy of different estimation approaches in a realistic environment. We use the TPC-W application benchmark in this scenario in order to analyze question **Q6**.

Since we do not know the actual resource demands of our application, we cannot directly compare the estimation accuracy of the estimation approaches. Instead, we create a performance model of our application and parameterize it with the estimated resource demands.

Then we analyze the performance model and compare the predicted average utilization and average response time with measurements from the real system.

The exact procedure is as following: We collect measurements at the real system when applying the workload of the FIFA 98 world cup web site to the TPC-W benchmark application. With these measurements, we estimate the resource demands and create a performance model of the system. We analyze this performance model with 400, 600, 800 and 1000 EBs and then compare the results with corresponding measurements when running the TPC-W benchmark with these number of EBs. Thus, we can evaluate the accuracy of the predictions obtained with the estimated resource demands of an estimation approach.

5.4.2.1. Application Benchmark

Due to the lack of traces from real production systems, we use an application benchmark for this scenario instead. We identified a number of requirements such an application benchmark needs to fulfill in order to obtain representative results. The requirements are:

1. The application benchmark must be representative for typical multi-tier applications, which typically consist of an application server and a database. Preferably, the benchmark should be based on the Java Enterprise Edition (Java EE) standard.
2. The benchmark must come with a ready-to-use load driver including standard workloads. The workload intensity must be configurable in order to be able to inject different loads on the system of interest.
3. The benchmark should primarily stress the CPUs of the systems.
4. It must be possible to run the benchmark distributed over several systems so that resource demands of several resources need to be determined.
5. The workload must consist of several different transactions.
6. The application benchmark should be accepted by the industry and academia to be representative of real applications. Ideally, the benchmark is standardized.

We searched for application benchmarks that fulfill these requirements and built a list of possible candidates:

SPECjEnterprise2010. This is the newest version of the Java EE industry standard benchmarks published by the SPEC consortium. It is modeled after an application of an automobile manufacturer. It is a highly complex application using the complete technology stack of Java EE including web services and asynchronous communication.

RUBiS. This application benchmark is modeled after an internet auction system. It implements the core transactions of an auction system: selling, browsing and bidding. There are different implementations of the system using different technologies: Java EE, java servlets only or PHP. This benchmark is not standardized by an industry consortium.

TPC-W. This application benchmark is standardized by the TPC-W consortium. There are implementations based on Java EE and PHP. The application is modeled after an online book store.

The SPECjEnterprise2010 benchmark has a high inherent complexity due to the multitude of different technologies it depends on. Therefore, we decided against its usage in this scenario in view of the limited time frame of this thesis. As a compromise, we use the TPC-W benchmark instead. This benchmark is standardized and used in many scientific

papers [RKKD10, ZCS07, GBL+11]. We did not consider the RUBiS benchmark due to a lack of current documentation about its setup and internal functioning.

We use the TPC-W implementation available from the OW2 consortium[1]. This implementation relies on Java servlets and Java Database Connectivity (JDBC) to access a MySQL database. During setup and experiment runs we identified two issues with the current code provided by the OW2 consortium. Therefore, we changed the following parts of the benchmark:

- The benchmark uses a custom implementation for database connection pooling. The pool showed some reliability and performance problems when scaling the workload. Therefore, we replaced it with the standard mechanism for pooling database connections provided by Java EE.
- When inserting rows in the database, the benchmark used an old-fashioned way to determine the value of an auto-increment field. In case of high concurrency, this caused consistency issues. We replaced the involved database queries with the `LAST_INSERT_ID()` function provided by MySQL.

These are small changes of the TPC-W benchmark, which are not expected to influence its performance characteristics significantly.

5.4.2.2. FIFA 98 World Cup Workload

The TPC-W benchmark provides a ready-to-use load driver that can simulate a closed workload with a configurable number of users. The navigation behavior of each user is described by a first-order Markov chain. This results in a very steady workload with only few variations. However, real workloads are usually varying over time [SKZ07]. Time-varying workloads can positively influence some estimation approaches as already shown by Stewart et al. [SKZ07]. To improve the representativeness of our evaluation, we use workload traces from a real application instead.

Arlitt and Jin [MJ99] carried out a workload characterization of the access logs of the official web site of the FIFA 98 world cup. The access logs collected at the web servers of the site between April 26 and July 26, 1998 are publicly available. The access logs contain over 1.35 billion requests and can be regarded as being representative of a very busy web site [MJ99].

We adapted the load driver of the TPC-W benchmark to use the access logs of the FIFA 98 world cup web site. Before, the think times between individual requests were drawn from an exponential distribution with a configurable, but constant mean value. Now the load driver uses the timestamps in the access logs instead to determine the think times. Each time a client finishes a request at the server, the load driver determines the next request in the access log. Then it calculates the think time TT for this client in the following way:

$$TT = (LT_{next} - LT_{start}) - (T_{cur} - T_{start})$$

where LT_{next} denotes the timestamp of the next request in the log file, LT_{start} the timestamp of the first request in the log file, T_{cur} the current system time and T_{start} the system time when the load driver was started. If the think time is negative the client proceeds immediately with the next request. The load driver can be configured to use only every N-th request in the log file, so that the workload can be scaled down.

We use a part of the access logs from day 66 in this scenario. The log starts at 15:00 and lasts for two hours. In view of our smaller deployment, we use only every seventh

[1] `http://jmob.ow2.org/tpcw.html`

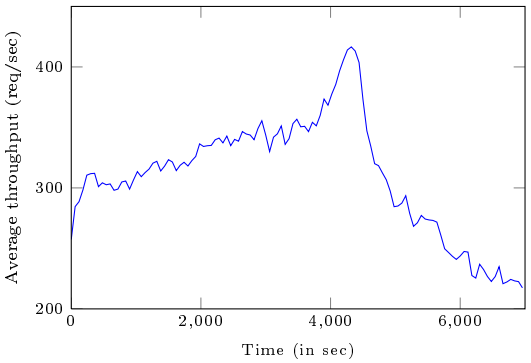


Figure 5.6.: Average throughput of FIFA 98 world cup workload.

request in the access log. Figure 5.6 shows the temporal development of the throughput in this time window. The workload varies at around 300 requests per second. It has a peak with almost 400 requests per second and levels out at slightly more than 200 requests per second.

5.4.2.3. Experiment Setup

In this scenario, we use the native execution environment described in Section 5.3.1. Enabling only one core of the processors, we avoid negative influences on the resource demand estimation due to multi-core processors. The TPC-W benchmark is deployed on two servers. The front-end server runs an Apache HTTP server and a Tomcat servet container. The back-end server contains a MySQL database server. The Apache web server and the Tomcat servlet container are integrated with the mod_jk module. The Apache web server serves requests for static resources and delegates requests for servlets to the Tomcat servlet container. The MySQL database is accessed through the standard JDBC driver provided by MySQL. The MySQL database uses the standard configuration, i.e. tables are managed by the default storage engine MyISAM. The load driver is executed on a separate machine. Table 5.4 gives details about the installed software.

Program	**Version**
Apache HTTP server	2.2.14
mod_jk	1.2.28
Java SE 64-bit	1.6.0_24
Tomcat	6.0.24
MySQL	5.1.41
MySQL JDBC ConnectorJ	5.1.14

Table 5.4.: Software configuration for the TPC-W benchmark.

During experiment runs, we obtain measurements of the utilization on the front-end and back-end server as well as the response times of individual request. We use the `sar` tool provided as part of the Ubuntu Linux distribution to monitor the utilization of the servers. We use a sampling interval of one second. The response times are collected with the built-in logging facilities of the Apache web server. Its access logs contain the response time of each request. We defined a custom logging format to print out the response time of each request with microsecond accuracy.

5.4.2.4. TPC-W Performance Model

The performance model needs to capture the major performance-relevant aspects of the TPC-W benchmark application. We regard the application as a black box without any

knowledge about its internal structures. Observations at the running system showed that the processing resources with the greatest influence on the performance are

- the CPU of the web server,
- the CPU of the database server,
- and the hard disk of the database server.

Since our focus lies on estimation approaches for CPUs and servers, we exclude the influence of the hard disk by keeping the complete database in memory.

With a high number of users, software contention plays a non-negligible role when predicting the performance of a web server or a database server. We identified the following sources for software contention:

- The Apache web server uses a process pool to process incoming requests.
- The Tomcat servlet container uses two process pools. One for parsing forwarded requests from the Apache web server and another one for executing servlets to generate the response.
- The Tomcat servlet container maintains a pool of database connections.
- The MySQL database server has a limited number of threads to process queries from clients.

We ensured that these pools do not incur any additional delays by setting their maximum size high enough so that requests never have to wait because of empty pools.

The workload is divided into two classes. One class for *browse* transactions and another one for *order* transactions. Table 2.1 on page 12 describes the assignment of the TPC-W transactions to the corresponding workload classes. The transactions of the browse workload class mainly cause `SELECT` queries at the database server, whereas transactions of the order workload class result in a mixture of `INSERT`, `UPDATE` and `SELECT` requests.

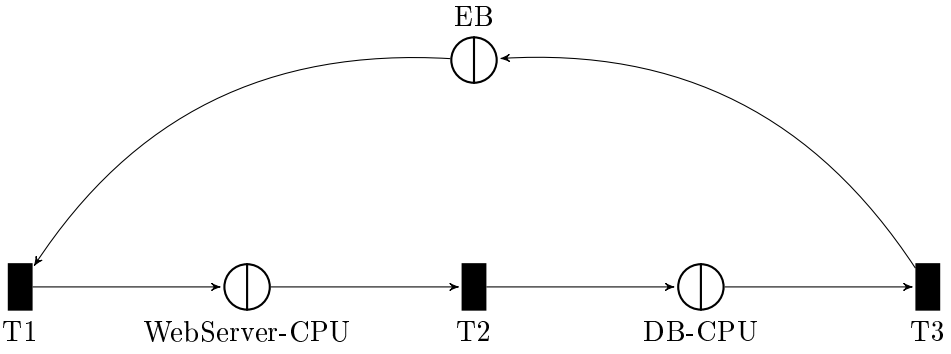


Figure 5.7.: QPN model of the TPC-W benchmark.

We use the QPN modeling formalism [Bau93] for our performance model. Figure 5.7 depicts the QPN model of the TPC-W benchmark application. It consists of three queueing places: `EB`, `WebServer-CPU` and `DB-CPU`. `WebServer-CPU` and `DB-CPU` both contain a PS queue, the place `EB` contains an IS queue. All queues have an exponential service time distribution. The mean service time of the queue of `EB` is equal to the think time configured in the load driver, i.e., in our case 1.75 seconds. It contains an initial population of tokens of the color `client` corresponding to the configured number of EBs in the load driver. Transition `T1` contains two firing modes with equal firing weights. Both modes consumes tokens of color `client` and produce one token of color `browse` or respectively `order`. The

equal firing weights ensure that we get a mix of 50% browse and 50% order requests. Two colors are defined in the place `WebServer-CPU`, one for each workload class. The mean service time is set to the estimated resource demand of each workload class. The transition `T2` just takes a token from place `WebServer-CPU` and puts a token of the same color in place `DB-CPU`. Place `DB-CPU` also contains a color definitions for each workload class. The service times are again set to the estimated resource demands of the database server. Finally, transition `T3` consumes a token of color `browse` or `order` and puts a token of color `client` in place `EB`.

The workload of the TPC-W benchmark is session-based in fact. However, Zhang et al. [ZCS07] show that the TPC-W application can be adequately modeled with the simplified transaction-based equivalent. Therefore, we do not explicitly model sessions in our performance model.

To analyze the QPN model, we use the well-proven and efficient simulator SimQPN in version 2.0 [KB06, KSM10]. This simulator provides so-called probes that can measure the response time of individual tokens over several places [KSM10]. The response time measurements start when a token enters place `WebServer-CPU` and finish when it leaves place `DB-CPU`.

5.4.3. Scenario C: Multi-core Processors

In the previous scenarios, we always enabled only one processor core. Now, we use two and more cores for the experiments. The multi-core scenario comprises two separate experiments. In the first experiment, we use the base workload from Scenario A in Section 5.4.1.1 and apply it on a computer with multiple processor cores. We carry out several experiment runs while increasing the number of enabled processor cores from one to four.

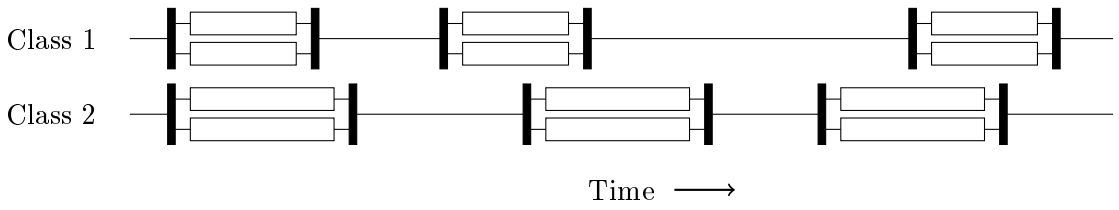


Figure 5.8.: Timing of the parallel workload.

In the second experiment, we use a workload with parallel tasks instead. Figure 5.8 illustrates the timing of the parallel workload. The resource demanding tasks fork a configurable number of threads, where each thread generates a share of the total resource demands. The total resource demands are equally distributed between the threads. To ensure that the duration of the generated resource demand of each thread is not too short for Ginpex, we increased the resource demands to 100 ms, 125 ms, 150 ms and 175 ms in this experiment. We carry out several experiment runs while varying the number of threads between 1 and 8.

5.4.4. Scenario D: Virtualization

This scenario is based on the one described in Scenario B in Section 5.4.2. Here, we deploy the TPC-W benchmark in the virtualized experiment environment instead. See Section 5.3.2 for details about the virtualized execution environment. The database server and the application server each run in their own VM. The load driver runs externally on a native computer. The load driver is connected with the VMs through a 1 Gbit/s network.

The general experiment procedure and setup is the same as in Scenario B in Section 5.4.2. We also use the same workloads and performance models for performance prediction. Measurements of the utilization and response time are done within the VMs.

6. Implementation of Estimation Approaches

In this chapter, we describe the tool for resource demand estimation we implemented as part of this thesis. We start with a description of the requirements that we expect from this tool. Then we develop an architecture for the tool that fulfills these requirements. Afterwards, we explain the design of the major components of the tool. Finally, we give an overview of the implementation of the tool including a description how we realized the estimation approaches.

6.1. Requirements

The goal of the tool for resource demand estimation is to enable the evaluation of different estimation approaches. For this purpose, we identified the following requirements:

Support for different estimation approaches: In Section 5.2, we selected a couple of approaches to resource demand estimation for further evaluation. The tool must support for these estimation approaches. The correctness and reliability of their implementations is important. Therefore, the implementations should reuse proven and commonly used components if available.

Support for inhomogeneous measurement data sources: Depending on the evaluation scenarios described in Section 5.4, different measurement tools are used. The tools store the measurement data in different formats. The data needs to be transformed into a common data format in order to simplify the implementation of the estimation approaches. For our purposes, it is necessary to import and transform measurement data from Ginpex and `sar` as well as access logs from Apache HTTP servers.

Data preparation: The raw measurement data from the monitoring tools requires further preprocessing before it can be used for resource demand estimation. Since the data might come from independent tools the observation periods might differ, e.g., if the tool for monitoring the utilization is started before response times are measured. In such cases, only measurements from the overlapping parts of the observations periods should be used for resource demand estimation. Furthermore, it is necessary to derive additional metrics from the existing observations, such as average response times or throughput, if monitoring tools do not provide the desired metrics.

Analyzable results: After resource demand estimation, the results need to be further analyzed and interpreted. Therefore, the resulting estimates and the input data after preparation should be stored in CSV files to simplify their subsequent processing in analysis tools, e.g., gnuplot, R or Matlab. The results of each approach to resource demand estimation should be stored separately. Additionally, the estimation approaches should output not only the final estimates but also their temporal evolution as new observations become available.

Configurability: The tool should provide variation points that can be configured separately in order to adapt the estimation process to specific scenarios and to evaluate the impacts of different parameters on resource demand estimation. Especially, we should be able to configure the following parameters:

- The estimation approaches that are used.
- The input files containing the measurement data.
- The output directory where the results are stored.
- Parameters specific to each estimation approach.

Scriptability: It should be possible to start the estimation tool from the command line in order to allow the simple integration in scripts.

6.2. Architecture

With the requirements defined in Section 6.1, we developed an architecture for our resource demand estimation tool. In this section, we describe the resulting architecture. A core entity in this architecture are *traces*. A trace is a table where each row contains observations of a specific metric at a certain time. The first column contains timestamps, each following column contains the actual observations. Traces are used to store the measurement data and the resulting estimates.

Figure 6.1 depicts the architecture of the tool. We structured the program into a number of components in accordance with the separation of concerns principle. Each component is responsible for a subset of the requirements. We implemented the following six components:

- The *estimation controller* is the central component that controls the execution of resource demand estimation. It expects a configuration file as input. The user must provide this file. It is used to adapt the estimation process to the field of application. The configuration defines which implementations of the data importer, data preprocessor, estimation strategy and data exporter components are used.
- The *trace repository* is the central place where traces are stored. All other components can read and write traces in the repository. It is currently implemented as an in-memory, non-persistent data store.
- The *data importer* component loads measurement traces produced by the monitoring tools. It translates the traces to a canonical format and stores them in the trace repository. Different implementations of this component exist for each supported measurement trace format. There are currently implementations for utilization traces from `sar`, response time traces from `Ginpex` and access log files from Apache web servers.
- The *data preprocessor* component prepares the measurement traces for resource demand estimation. For example, it may be necessary to filter out observations or to derive additional metrics from existing measurement traces. A data preprocessor loads, changes and stores traces in the repository.

- The *estimation strategy* component implements the actual resource demand estimation. There are implementations of this component for each approach to resource demand estimation. An estimation strategy gets the required measurement traces from the trace repository and stores the resulting estimates in the repository.
- The *data exporter* component is used to export selected traces from the trace repository in a data format that can be used for subsequent analyses. There are currently component implementations that write traces to CSV files or print them to the console.

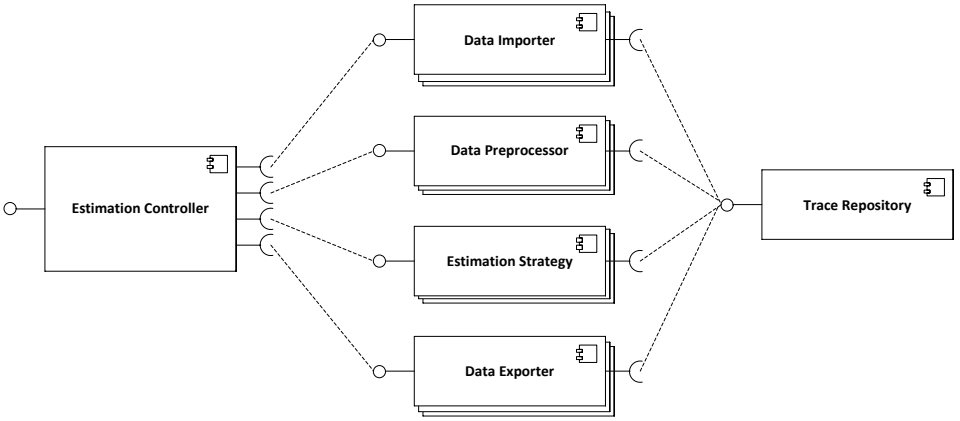


Figure 6.1.: Architecture overview.

The tool is based on the Eclipse platform and divided into set of plug-ins. To ensure the easy scriptability, we realized it as a headless Eclipse application, i.e., it has no graphical user interface. The user provides all information necessary for the estimation tool either in the configuration file or with program arguments on the command line. Since the structure of the configuration file is relatively complex, we decided to use the Eclipse Modeling Framework (EMF) for this purpose. Therefore, we created a metamodel that describes the configuration model.

6.3. Design

We describe the central design decisions of the estimation tool in this section. In particular, we explain how the measurement data and estimation results are managed in the trace repository and we give an overview of the metamodel defining the estimation configuration model.

6.3.1. Trace Repository

The trace repository is the central data storage for measurement data and estimation results. Figure 6.2 shows the data model of the repository. Traces are the core entity of the repository data model. A trace contains a set of tuples $t = (T, D_1, D_2, \ldots, D_N)$ where T is a timestamp and D_i are observations made at time T.

Each trace is associated with exactly one resource, e.g., a utilization trace references the monitored CPU or a response time trace is linked with the group of systems for which the response time is measured. There are two types of traces: measurement and result traces. Measurement traces contain the input data for estimation approaches. A measurement trace contains only observations of one metric. Possible metrics are: utilization, response times, throughput and average response time. Result traces are used to store the results

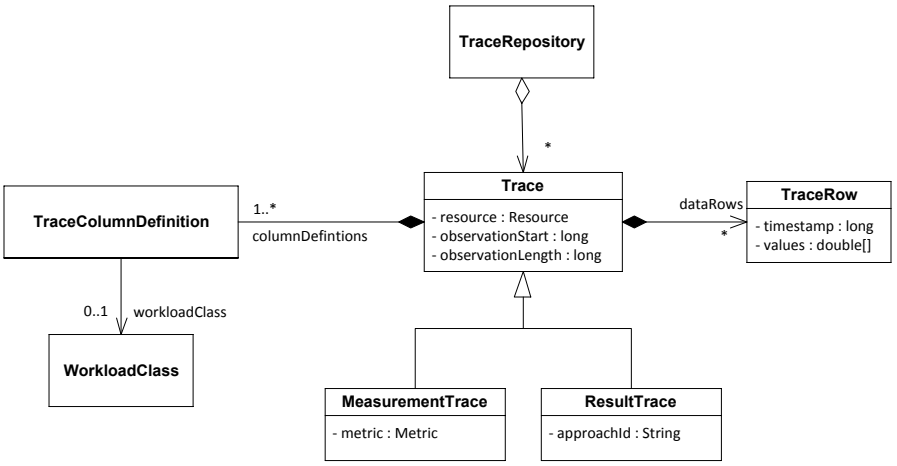


Figure 6.2.: Data model of the trace repository.

of an estimation approach. They always contain an identifier of the estimation approach in order to distinguish between the results of different approaches.

A trace consists of a set of rows. Each row contains a timestamp and a set of values. The number of values in each row must be equal to the number of columns defined in the trace. A column definition contains additional metadata about the observations stored in the corresponding column. It can be optionally associated with a workload class. For instance, a throughput trace usually contains several columns, one for each workload class. If no workload class is associated with a column, the observations in this column are for all workload classes, e.g., the total utilization of a resource. Only one such column is allowed in a trace.

The trace repository offers a number of operations to load and store traces. The methods `queryMeasurementTraces()` and `putMeasurementTrace()` are for measurement traces. Correspondingly, the methods `queryResultTraces()` and `putResultTrace()` are for result traces.

6.3.2. Estimation Configuration Metamodel

An estimation configuration model contains two types of information:

1. a description of the workload and the resource environment for which resource demands should be esimated.

2. a configuration of the estimation process including concrete values for the configuration parameters of the components.

Figure 6.3 gives an overview of the top-level elements of the metamodel. The root element of a configuration model is always an instance of the `EstimationConfiguration` class. Elements of the classes `Workload` and `ResourceEnvironment` contain information of type 1. Elements of the classes `DataImportSpecification`, `PreprocessingSpecification`, `EstimationSpecification` and `DataExportSpecification` contain information of type 2. We describe each of these six classes and their children in the following.

Figure 6.4 depicts the elements for describing the workload and the resource environment. Each element of the class `WorkloadClass` introduces a workload class with a given name. The name is used as a human-readable identifier of the workload class. The subclass

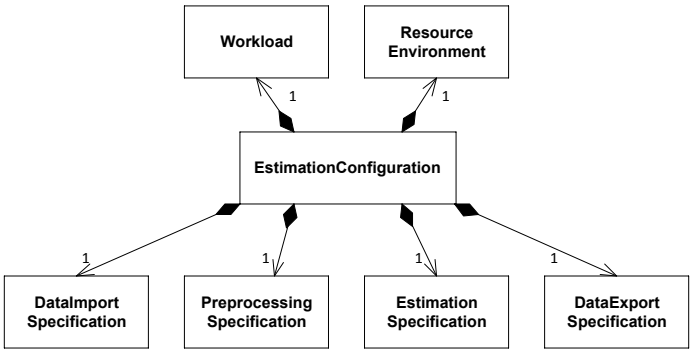


Figure 6.3.: Overview of the metamodel.

`HttpWorkloadClass` provides the additional attribute `urlPrefix`. All requests in an access log from a web server whose Uniform Resource Locators (URLs) starts with this prefix are attributed to that workload class.

The resource environment is modeled with different types of resources. All types of resources are subclasses of the common base class `Resource`. A system group represents a set of computer systems, e.g., web server, application server and database server of a multi-tier application architecture. Each system of a system group is represented by a system node including its CPUs and disks. This information of the resource hierarchy is required to attribute measurement traces to the correct resource.

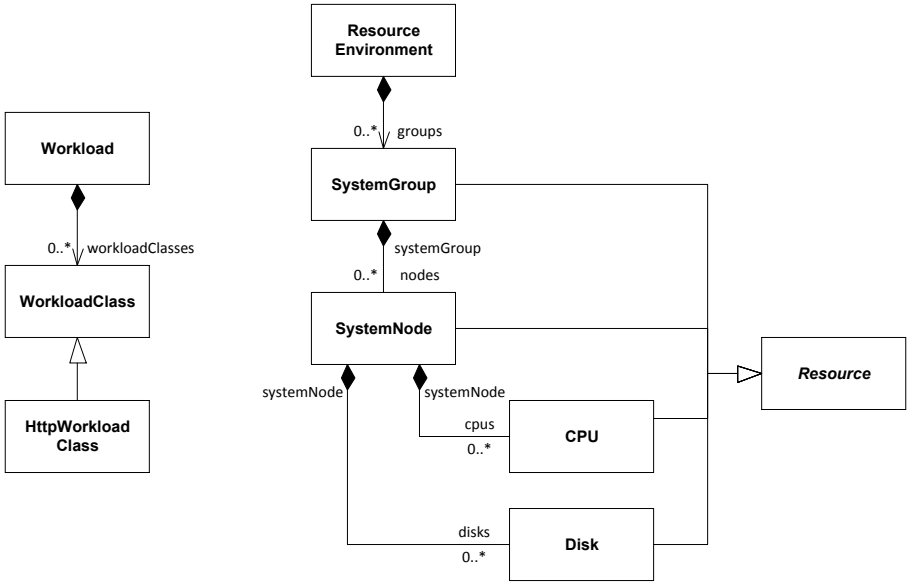


Figure 6.4.: Workload and resource environment description.

The components DataImporter and DataExporter are configured with the elements shown in Figure 6.5. A data import configuration contains the attribute `fileName`, which lets the user specify the measurement trace file that should be imported. Additionally, each data importer configuration must be associated with one resource. This association states that the imported measurement trace were collected for the specified resource. There are specialized configuration elements for each supported input and output format.

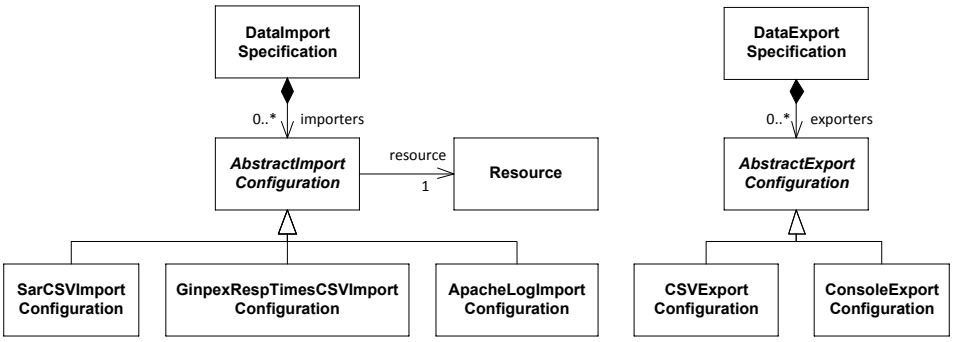


Figure 6.5.: Data import and export configuration.

Figure 6.6 shows the elements of a preprocessing specification. The data preprocessing configuration is used to configure a DataPreprocessor component. There are different types of data preprocessing configurations for different purposes:

- An *observation period standardizer* ensures that all observation periods of different measurement traces have the same start time and length. It determines the maximum overlap of the observation periods and discards all measurements that lie outside.
- A *transient period cutter* removes a configurable number of measurement samples at the beginning and the end of each measurement trace.
- The *throughput* and *average response time calculators* are applied to response time traces to calculate throughput and average response times at specified intervals.

If several data preprocessing configurations are specified, several data preprocessor component instances are created and executed in the same order as the corresponding data preprocessing configurations. Therefore, it is possible that a data preprocessor works on the output of a previous one.

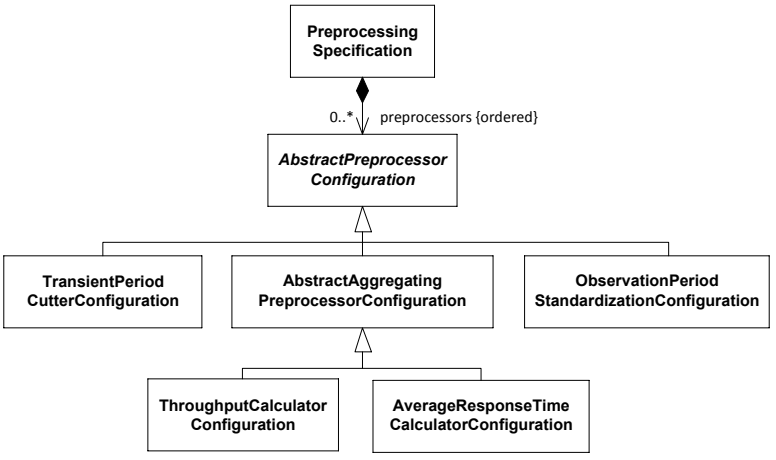


Figure 6.6.: Data preprocessing configuration.

Finally, the estimation specification, as shown in Figure 6.7, is used to configure each estimation strategy. The estimation strategies might provide additional configuration parameters that are specific for each strategy.

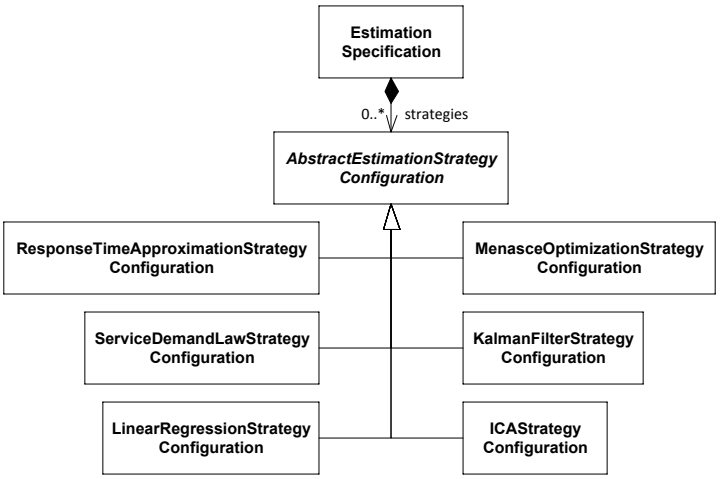


Figure 6.7.: Resource demand estimation configuration.

6.4. Implementation

In this section, we discuss major implementation issues and show how we solved them. We start with an in-depth description of the data preprocessors. Afterwards, we give an overview of the tools and libraries we considered to use for the implementation of the estimation approaches including a consideration of the pros and cons of each identified alternative. Subsequently, we provide details about the concrete realization of each estimation approach. Finally, we describe the plug-in structure of the tool.

6.4.1. Data Preprocessing

We implemented four data preprocessors that prepare the input measurement traces for resource demand estimation:

- The `ObservationPeriodStandardizer` is used to adapt the observation periods of a set of measurement traces to a common period, in cases where traces originate from separate, not synchronized monitoring tools. In a first step, the `ObservationPeriodStandardizer` fetches all utilization and response time traces of a system group and determines the maximum overlapping part of the observation periods of all traces. In a second step, the overlapping part is used as the new standard observation period for all measurement traces. Measurements outside of the standard observation period are discarded. If the measurement traces originate from different systems, the `ObservationPeriodStandardizer` assumes that the time on these systems is synchronized, e.g., with Network Time Protocol (NTP).
- The `ThroughputCalculator` depends on response time measurements of individual requests. It determines the number of requests during given time intervals based on the arrival timestamps of the observed requests. Then it uses this information to calculate the throughput during these time intervals. These time intervals are taken from the utilization measurement traces available for the current machine. Thus, we can ensure that throughput and utilization measurements have the same time intervals.
- The `AverageResponseTimeCalculator` works similarly. It also calculates the average response times for specified time intervals. We use the time intervals of the utilization measurement traces here as well.

Estimation approach	Languages and libraries
Response time approximation	Java, Apache Commons Math
Service Demand Law	Java, Apache Commons Math
Kalman filter	C++, Bayes++ [Bay]
Linear regression	Matlab
Menasce optimization	Matlab
ICA	Matlab, FastICA [Fas]

Table 6.1.: Programming languages and libraries used for the implementation.

The `ObservationPeriodStandardizer` should be applied to the measurement traces before all other preprocessors. Thus, the traces produced by the `ThroughputCalculator` and the `AverageResponseTimeCalculator` automatically have the correct observation period. These two preprocessors are independent of each other and can be applied in any order.

6.4.2. Tools and Libraries

If possible, the implementations of the approaches to resource demand estimation are based on existing tools and libraries in order to ensure their reliability and correctness and to reduce the implementation effort. Table 6.1 gives an overview of the programming languages and libraries used for implementing the estimation approaches.

We realized the response time approximation and Service Demand Law estimation approaches directly in Java. Both approaches require only basic statistical functions, such as the mean or minimum functions. We used the Commons Math library of the Apache Foundation for these functions.

There is already an implementation of an extended Kalman filter for resource demand estimation in our research group. We integrated and extended this implementation in our tool. The implementation is realized in C++ and is based on the Bayes++ library, which provides various Bayesian filtering algorithms including Kalman filters [Bay]. The library contains a set of base classes that can be extended to define custom state and observation models that describe the system of interest. The filters included in the library work with the given models to derive estimates of the real state. The Kalman filter implementation of Bayes++ already addresses common issues that occur in the practical application of Kalman filters, such as ensuring the symmetry of matrices. The existing implementation of our research group provides complete implementations of state and observation models for resource demand estimation according to the filter design described by Kumar et al. [KTZ09].

As the existing Kalman filter implementation is written in C++, an adapter to call it from Java code is necessary. We considered the following alternatives to realize this integration:

Separate program. We invoke the Kalman filter as *an external program* and pass on the required information through program arguments or temporary files. This can be done with standard Java classes. However, it requires additional logic to write the measurements to temporary files and read the resulting estimates.

Java Native Interface (JNI). The Java Development Kit (JDK) includes the JNI technology for developers who want to access C functions from Java code. However, the Application Programming Interface (API) of JNI is very complex and data types need to be converted manually between C and Java. Therefore, a lot of glue code is required to access C functions in Java.

Java Native Access (JNA). The third-party library JNA hides the complexity of JNI from the developer. The developer must only provide plain Java interfaces containing

Java method declarations for each C function that should be accessible in Java. JNA automatically generates proxies which invoke the corresponding C functions through JNI and convert the input and output parameters to the expected type.

We chose to use JNA for our purpose. It allows us to tightly integrate the Kalman filter implementation without the need for temporary data transfer files. At the same time, the integration can be done without much coding efforts.

The remaining estimation approaches, namely linear regression, Menasce optimization and ICA, rely on more complex mathematical methods, such as non-negative LSQ regression and non-linear, constrained optimization. Therefore, we looked for existing implementations of these methods we can build upon. In particular, we considered the following alternatives:

Third-party Java libraries. There are a number of third-party Java libraries providing mathematical and statistical functions. Examples for such libraries are Apache Commons Math 2.2, Colt 1.2 or OR-Objects 1.2.4. The advantage of these libraries is their simple integration in Java programs. However, they primarily support linear programming algorithms in the current versions. Solution algorithms for non-linear, constrained optimization problems are missing.

The R statistical language. R is a programming language speficially targeted to statistical computations. It provides tools to solve linear regression problems as well as algorithms for linear and quadratic optimization. However, it also lacks algorithms for non-linear, constrained algorithms. Furthermore, the invocation of R in Java is time-consuming because the commands need to be put together manually in Java code.

Matlab. Matlab is a proprietary program for numerical mathematical computations. It includes a number of toolboxes for different domains. There are also toolboxes for statistics and optimization, including algorithms for ordinary and non-negative LSQ regression, and for non-linear, constrained optimization. Functions and scripts for mathematical computations are written in a proprietary programming language. However, Matlab allows to automatically generate Java classes for selected functions. This simplifies the integration of Matlab functions in Java programs.

GNU Octave. Octave is a free tool for numerical mathematical computations similar to Matlab. Its script language is to a large extent compatible to Matlab. However, it only supports a subset of Matlab functions. Especially, it only supports one algorithm for the solution of non-linear optimization problems.

After considering the pros and cons, we decided to use Matlab for the further implementation because it provides the best support for the required mathematical methods. Additionally, the possibility to generate Java classes encapsulating Matlab functions simplifies its integration in Java programs. Obviously, the dependency on Matlab limits the distribution of our tool for resource demand estimation. However, as part of this thesis it is not regarded as a major issue.

We used the FastICA library [Fas] for the ICA estimation approach. Packages are available for different environments, including Matlab. We used the current version 2.5 for Matlab in our implementation.

6.4.3. Implementation of the Estimation Approaches

Response time approximation

This estimation approach is either applied to a whole system group or to each system node separately. The behavior is controlled with the `resourceLevel` attribute in the

estimation configuration. This attribute can be set to `SystemGroup` or to `SystemNode`. It is uncommon to use it for separate resources within a system, such as CPUs and disks, because corresponding response time measurements on this level of granularity are usually hard to obtain. Our implementation starts with a conservative estimate, which is either provided by the user with the attribute `initialEstimate` in the configuration file or set to `Double.MAX_VALUE`. Then the estimate is updated for each new response time measurement if the new value is smaller than the current estimate.

The attribute `windowSize` in the estimation configuration can be used to specify that only measurements in the specified time window are considered when determining the minimum. For example, if the window size is set to ten minutes, the minimum response time is only determined for the measurements that were made in the last ten minutes. If this value is not set, all measurements are considered when determining the minimum response time.

Service Demand Law approach

The Service Demand Law approach is implemented as described by Brosig et al. [BKK09] including the partitioning scheme for the total utilization with weighted response time ratios. For details see Section 3.1.1 on page 14. This approach also supports the attribute `windowSize` in the estimation configuration. Here, it controls the size of the time window for which the average response time and average utilization is determined. If this attribute is missing, the averages are calculated over all available measurements.

Kalman filter approach

There are several attributes that control the behavior of the filter. The attributes `processNoise` and `observationNoise` in the estimation configuration correspond to the process noise $\mathbf{Q}$ and the observation noise $\mathbf{R}$. The attribute `initialEstimate` provides a way to specify an initial estimate $\mathbf{x}_{0|0}$ for all workload classes. If the attribute `initialEstimate` is missing, the filter sets the starting point to one by default. The initial estimate of the error covariance $\mathbf{P}_{0|0}$ is automatically derived as recommended by Zheng et al. [ZYW+05]: The diagonal elements of the matrix $\mathbf{P}_{0|0}$ are initialized with the square of the initial estimates $\mathbf{x}_{0|0}$.

The filter design is based on the one proposed by Kumar et al. in [KTZ09], except that we only estimate the resource demands for systems and no network delays. The network delays are assumed to be zero. The state and the observation model have the same structure as described in [KTZ09]. We have generalized the models accordingly so that they can be used for an arbitrary number of workload classes and resources.

We included the truncation method proposed by Zheng et al. in [ZWL08] in order to limit the estimates to non-negative resource demands. The lower bound is chosen to be zero, the upper bound is set to infinity.

Linear regression

The implementation of the linear regression approach uses non-negative LSQ regression by default. We use the Matlab function `lsqnonneg` for this task. It is also possible to use ordinary LSQ regression, which allows negative resource demand estimates. This kind of regression can be activated by setting the attribute `allowNegativeEstimates` to `true`. Then the Matlab function `regress` is called instead. Furthermore, it is also possible to specify the attribute `windowSize` to configure a sliding time window.

Menasce optimization

The Menasce optimization is based on non-linear, constrained optimization. Matlab provides the `fmincon` function for such optimization problems. The `fmincon` function searches for an allocation of the variable vector x, that minimizes an objective function $f(x)$ fulfilling a set of constraints on x [Mat]:

$$\min_x f(x) \text{ such that } \begin{cases} c(x) & \leq 0 \\ ceq(x) & = 0 \\ A \cdot x & \leq b \\ Aeq \cdot x & = beq \\ lb & \leq x \leq ub \end{cases} \tag{6.1}$$

The caller can specify functions for $c(x)$ and $ceq(x)$ to define nonlinear constraints. Matrices A and Aeq with corresponding vectors b and beq are used to define linear constraints. Furthermore, vectors lb and ub limit the range of possible values of x.

Menascé provides a general description of the optimization problem in [Men08]. We had to translate it according to the input format expected by the `fmincon` function of Matlab. We solved it in the following way:

- In fact, the service demands $D_{i,c}$ with $1 \leq i \leq I$ and $1 \leq c \leq C$ for I resources and C workload classes constitute a $I \times C$ matrix. This matrix is reshaped into a vector $\mathbf{x} = (D_{1,1}, \ldots, D_{1,C}, D_{2,1}, \ldots, D_{2,C}, \ldots, D_{I,1}, \ldots, D_{I,C})$ because `fmincon` assumes $\mathbf{x}$ to be a vector.
- We implemented the objective function as a Matlab function that takes vector $\mathbf{x}$ containing the resource demands and returns a scalar value. This function is passed to `fmincon` as a function reference.
- The constraint that all resource demand parameters must be non-negative, is realized by setting the parameter `lb` (lower bound) of `fmincon` to a vector of zeros.
- The second constraints states that for each resource the total utilization is always less than one. It is passed to the `fmincon` function as a linear constraint of the following form:

  $$\mathbf{A} \cdot \mathbf{x} \leq \mathbf{b}$$

  $$\text{where } \mathbf{A} = \begin{pmatrix} \lambda_1 & \ldots & \lambda_C & 0 & \ldots & 0 & \ldots & 0 & \ldots & 0 \\ 0 & \ldots & 0 & \lambda_1 & \ldots & \lambda_C & \ldots & 0 & \ldots & 0 \\ \vdots & \ddots & \vdots & \vdots & \ddots & \vdots & \ddots & \vdots & \ddots & \vdots \\ 0 & \ldots & 0 & 0 & \ldots & 0 & \ldots & \lambda_1 & \ldots & \lambda_C \end{pmatrix}, \mathbf{b} = \begin{pmatrix} 1 \\ 1 \\ \vdots \\ 1 \end{pmatrix}.$$

  $\mathbf{A}$ is a matrix with I rows and $C \cdot I$ columns that contains throughput measurements λ_i for all workload classes. The vector $\mathbf{b}$ is of size I.

Since the `fmincon` function uses gradient-based algorithms to find a solution, it depends on the first-order partial gradients and the Hessian matrix of the objective function. We rely on the possibility to let `fmincon` determine the partial gradients and the Hessian matrix. Specifying this information as additional input to `fmincon` might improve estimation speed and quality.

Matlab provides different gradient-based algorithms that can be used with the `fmincon` function. By default, we use the `interior-point` algorithm [BGN00] because it is applicable to small dense problems as well as large sparse ones and supports bounds and linear constraints. Other algorithms can be specified with the `solutionAlgorithm` attribute.

As all algorithms are gradient-based, they iteratively search for a solution. The maximum number of iterations before stopping if no solution is found the search is configured with the attribute `maxIterations`.

The search for a solution starts with an initial estimate. An initial estimate for all workload classes can be specified with the `initialEstimate` attribute in the estimation configuration. The `fmincon` function is iteratively invoked for each measurement. The solution of one iteration is used as the initial estimate for the next one. In order to compensate temporary fluctuations we apply a smoothing method and calculate averages over the last N solutions. The number N can be specified with the `windowSize` attribute.

Independent Component Analysis

The ICA estimation approach expects the number of independent components as input. Since we do not know the exact number of independent components in advance, we set it to the number of resources for which we have utilization measurements as described by Sharma et al. in [SBC+08]. This might result in superfluous independent components if the number of workload classes is less than the number of resources. In this case the user has to identify and eliminate the superfluous independent components manually.

The estimates of the ICA cannot be directly mapped to the actual workload classes. Therefore, it creates new workload classes for each independent component. The new workload classes are labeled IC_x where x is a number.

6.4.4. Plug-in Overview

We implemented the estimation tool as a set of Eclipse plug-ins. Figure 6.8 gives an overview of all plug-ins. The plug-ins in the upper half contain the actual code of the estimation tool we have written. The wrapper plug-ins for external, third-party libraries used by the estimation tool are depicted in the lower half.

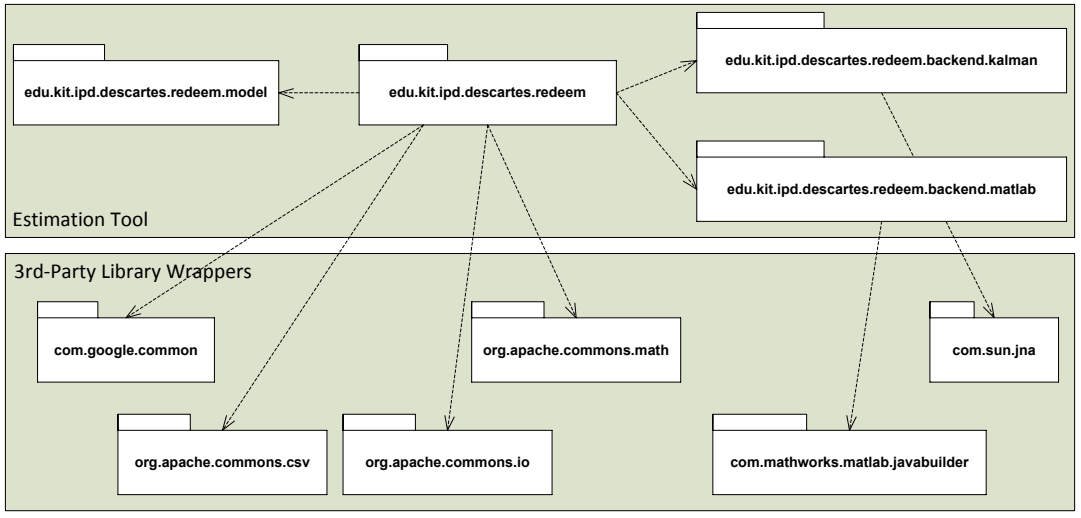


Figure 6.8.: Overview of plug-ins.

The plug-in `edu.kit.ipd.descartes.redeem.model` contains the EMF metamodel of the estimation configuration model including the Java source code generated by EMF. The plug-ins `edu.kit.ipd.descartes.redeem.backend.kalman` and `edu.kit.ipd.descartes.redeem.backend.matlab` contain the C++ functions of the Kalman filter

and the Matlab functions for linear regression, Menasce optimization and ICA. Additionally, they contain the wrapper code to call these functions from Java code. The plug-in `edu.kit.ipd.descartes.redeem` contains the core components of the estimation tool, which are the trace repository, the data importers and exporters, the data preprocessors and the estimation strategies.

The plug-ins depend only on the core runtime of Eclipse. The estimation tool is realized as a headless application, i.e., it does not depend on a graphical user interface. The user controls the behavior of the estimation tool with program arguments and the estimation configuration file.

6.5. Future Enhancements

For the future we plan to extend the estimation tool in the following directions:

- The current implementation does not support real online estimation of resource demands. This would require a direct connection between the monitoring tools and the estimation tool. The monitoring tools must continuously send updated measurements to the estimation tool. This could be realized with standard management protocols, such as Simple Network Management Protocol (SNMP), Java Management Extensions (JMX) or Windows Management Instrumentation (WMI), which are supported by many servers and operating systems.
- The expressiveness of the resource environment description in the estimation configuration model is currently limited. For instance, it is not possible to describe communication links between servers. In combination with detailed visit counts, this information can be beneficial for resource demand estimation.
- The results of the resource demand estimation can be only stored in flat files in the current version. Future versions might support to automatically parameterize performance models with the estimated resource demands. This can be realized by providing specialized data exporters for the performance models.
- The implementations for the data importer, preprocessor, estimation strategy and data exporter components are hard-wired in the current version. It would be useful to use the extension mechanism provided by the Eclipse plug-in system instead. This would greatly improve the extensibility of our tool.
- A graphical user interface including an editor for the estimation configuration model would improve the usability of the tool. Furthermore, a specialized launcher for the estimation tool could be developed that allows to start the estimation from within Eclipse.

6.6. Concluding Remarks

We provide a ready-to-use tool for estimating resource demands based on non-intrusive, low-overhead measurements, such as the CPU utilization and average response times. The tool comes with implementations of different estimation approaches. The tool executes the following major steps when estimating resource demands: It starts with importing the raw measurement traces as produced by the monitoring tools. Then the raw measurement data is prepared for resource demand estimation. Afterwards, the tool invokes a set of estimation approaches with the imported measurement data. Finally, the results from the resource demand estimation are stored in files and the user can analyze the results. Each of these steps is configurable and can be adapted to the specific requirements of a concrete situation. The tool can also be extended with custom implementations of the core components.

7. Evaluation Results

In this chapter, we describe the results of the evaluation of different approaches to resource demand estimation. We evaluated the estimation approaches listed in Section 5.2. These estimation approaches were applied to the scenarios described in Section 5.4. In the following sections we explain the results of each scenario.

7.1. Scenario A: Artificial Workload

In this section, we describe the results of a number of experiments with artificial workloads generated with the Ginpex framework [HKHR11]. We compare the estimation accuracy of different approaches to resource demand estimation and evaluate the influence of certain workload characteristics.

The estimation accuracy is quantified with the mean relative error E_{rel} defined as

$$E_{rel} = \frac{1}{C} \sum_{c=1}^{C} \left| \frac{D^c_{est} - D^c_{real}}{D^c_{real}} \right|$$

where C denotes the number of workload classes, D^c_{est} the estimated resource demand of workload class c and D^c_{real} the real resource demand of workload class c. We use the average of the estimates in the last ten minutes of the observation period as D^c_{est} in order to compensate for short-term fluctuations of the estimates. If we aggregate the results from several experiment runs, we use the total mean relative error E_{total} defined as

$$E_{total} = \sum_{r=1}^{R} E^r_{rel}$$

where R denotes the number of experiment runs and E^r_{rel} the mean relative error of experiment run r.

This section is structured as follows: We first mention major issues that need to be considered when using the estimation approaches. Then we present the results from the base experiment. Afterwards, we show three sensitivity analysis different variations of the workload of the base experiment. At the end, we discuss the results of this scenario.

7.1.1. Preliminary Remarks

During evaluation, we identified the following issues that need to be considered when using the estimation approaches. Using the Kalman filter approach as desribed by Kumar et al. [KTZ09], we experienced numerical stability and convergence problems. Furthermore, the Kalman filter and Menascé optimization approaches both depend on a sensible initial estimate to converge to a solution. Finally, the size of the monitoring window significantly influences the estimation accuracy of some of the estimation approaches. In the following, we describe these issues in-depth.

7.1.1.1. Numerical Stability

The error covariance matrix $P_{k|k}$, which is part of the state of a Kalman filter, must always be symmetric. In theory, the Kalman filter equations are designed in a way that updated values of $P_{k|k}$ are always symmetric matrices. However, given the finite-precession arithmetic of computers, $P_{k|k}$ might become asymmetric after some filter steps. Then the estimation accuracy of the Kalman filter might suffer or with our implementation, the estimation fails prematurely because a sanity check fails. We solved this issue by explicitly making the matrix $P_{k|k}$ symmetric after each update step, as proposed by Simon [Sim06]. We used the following equation to keep $P_{k|k}$ symmetric:

$$P_{k|k} = \frac{1}{2}\left(P_{k|k} + P_{k|k}^T\right) \tag{7.1}$$

We added this additional step in the Bayes++ library before the updated matrix is assigned to the variable containing the error covariance matrix $P_{k|k}$.

The calculation of the gain matrix in the update phase of the Kalman filter includes an inversion operation on an intermediary result matrix. In some experiments, we experienced failures because the Kalman filter implementation could not calculate the inverse of this matrix. An in-depth analysis of the problem revealed that the inversion fails if the calculated utilization in the measurement model of the Kalman filter (see Equation 3.3 on page 18) is close to 100%. Then elements of the observation vector and the Jacobian matrix of the measurement model might be close to infinity causing numerical stability problems. As a result, some matrices cannot be inverted. We solved this issue by limiting the calculated utilization to 99%. All values above are truncated.

7.1.1.2. Initial Estimates

The Kalman filter and Menascé optimization approaches rely on an initial estimate of the resource demands. A performance engineer must provide sensible values as initial estimates. We observed a dependency between the given initial estimate and the estimation accuracy.

Figure 7.1 shows the total mean relative error of the Kalman filter and Menascé optimization approaches when varying the initial estimate. The measurement data came from 28 experiment runs with varying utilization levels (between 20% and 80%) and varying number of workload classes (between 1 and 16). The initial estimate is the same for all workload classes. The Kalman filter reveals significant convergence issues if the initial estimate is chosen too high. In this case, the calculated utilization based on the measured throughput and the initial estimate is close to or above 100% in the transient phase before the Kalman filter can reach better estimates. The measurement model in Equation 3.3 on page 18 has a highly non-linear character with utilization close to 100% because the response time asymptotically tends to infinity. We use the EKF here to cope with the nonlinear measurement model. However, EKF is only expected to work well with almost

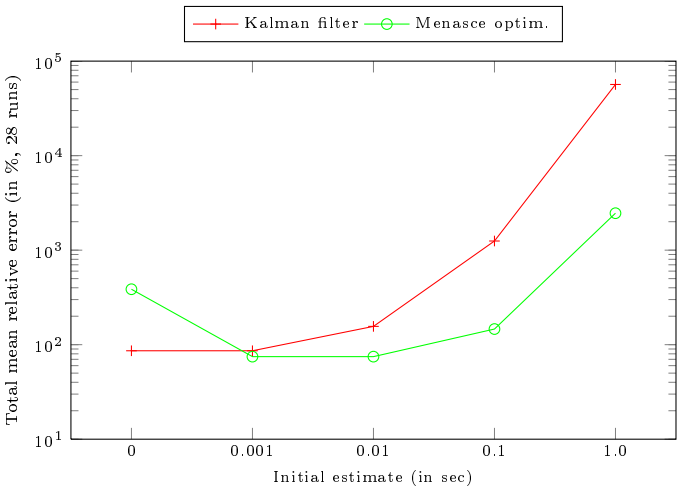


Figure 7.1.: Total relative error depending on the initial estimate.

linear models [ZWL08]. If the initial estimate is too high, the Kalman filter starts in the critical region where the measurement model is highly non-linear, and therefore has convergence issues. Otherwise, the influence of the initial estimate is insignificant as already shown by Zheng et al. [ZWL08].

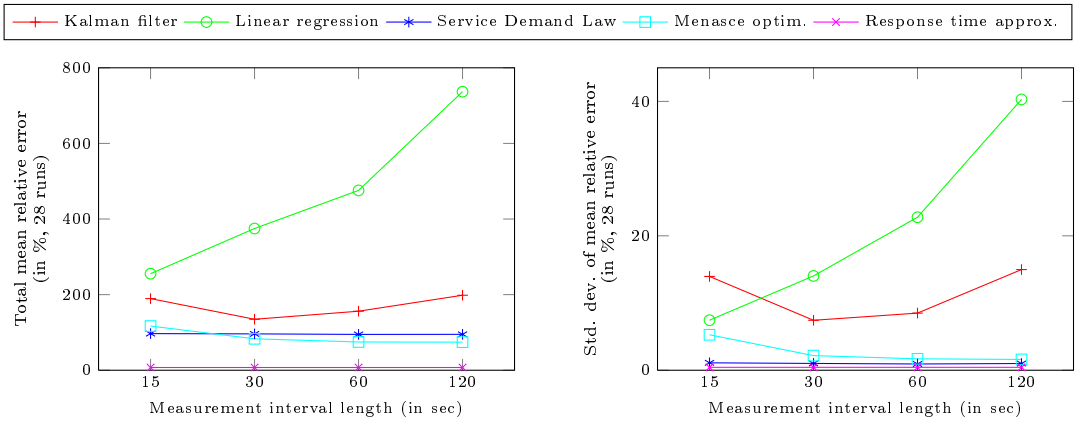
The Menascé optimization approach has significant convergence issues if we set the initial estimate to zero or to a high value where the calculated utilization is close to or above 100 per cent. In both cases, the solution algorithm we used to solve the non-linear optimization problem could not find a solution within a reasonable number of iterations. We limited the maximum number of iterations to 1000. The solution algorithm might be able to find a solution to the optimization problem when raising this limit. However, the solution algorithm can require significant computing time the more iterations it requires. This might not be feasible when using it for online estimation in a production environment.

As a general rule of thumb, we recommend to use initial estimates that underestimate the real resource demands. If we have no previous knowledge of the order of magnitude of the real resource demands, it might be worthwhile to obtain a rough estimate with the Service Demand Law or response time approximation before applying the Kalman filter or Menascé optimization. We used one millisecond as an initial estimate in the following experiments because it is always significantly smaller than the real resource demands, which are greater or equal to 25 milliseconds.

7.1.1.3. Monitoring Window Size

We obtained utilization measurements every fifteen seconds on the experiment machine. The throughput and average response time measures are determined for the same intervals. These measurement traces are used to examine the sensitivity of various estimation approaches under different monitoring window sizes. The results of this analysis help us to determine optimal monitoring window sizes for the following experiments.

Figure 7.2 depicts the estimation accuracy of each approach with window sizes of 15 sec, 30 sec, 60 sec and 120 sec. We obtained the measurement data during a number of experiment runs with varying utilization levels (between 20% and 80%) and varying number of workload classes (between 1 and 16). All in all, we carried out 28 experiment runs. Figure 7.2(a) shows the total mean relative error over all experiments and Figure 7.2(b) depicts the standard deviation of the mean relative errors of each experiment run.



(a) Total mean relative error

(b) Standard deviation of mean relative error

Figure 7.2.: Estimation accuracy under different monitoring window sizes.

The response time approximation and the Service Demand Law approaches are obviously not dependent on the size of the monitoring window. The response time approximation approach works on the response times of individual requests and thus is not influenced by the size of the monitoring window. The Service Demand Law approach calculates the average utilization and throughput over the whole observation period. Therefore, the size of the monitoring window is irrelevant. The Menascé optimization approach is only slightly influenced by the size of the monitoring window. It provides better estimates with longer monitoring windows. Especially, the variation of the mean relative errors is reduced. The Kalman filter yields the best results with a window size between 30 and 60 seconds. With shorter or longer monitoring windows the estimates become less stable and the mean relative error increases.

In our experiments, the size of the monitoring window had the biggest influence on the estimation accuracy of the linear regression approach. The mean relative error is significantly reduced when using shorter monitoring windows. At first sight, this result contradicts the observations done by Zhang et al. [ZCS07]: They come to the conclusion that larger monitoring windows work better in their case. However, the generated workload in this experiment has only a low variation in the throughput. If utilization and throughput are averaged over long intervals, the variation is even more reduced. That is a problem for linear regression as it requires a certain level of variation in the control variables in order to obtain accurate estimates.

This analysis shows that there is no window size that fits best for all approaches to resource demand estimation considered. Therefore, we used different monitoring window sizes in the following experiments. We used a monitoring window size of 15 sec for linear regression, one of 30 sec for the Kalman filter, and one of 60 sec for the other approaches.

7.1.2. Base Experiment

In this section, we present the results from the base experiment. We attempted to exclude any external influences that could have a negative effect on the estimation approaches. This experiment lays the foundation for a number of experiments described in the following section where we analyze the effects of different workload characteristics on the estimation accuracy of the considered estimation approaches. We carried out a number of experiment runs at utilization levels of 20%, 40%, 60% and 80%, and with 1, 2, 3, 4, 6, 8 and 16 workload classes, all in all 28 experiment runs. Each experiment run lasted approximately 25 minutes.

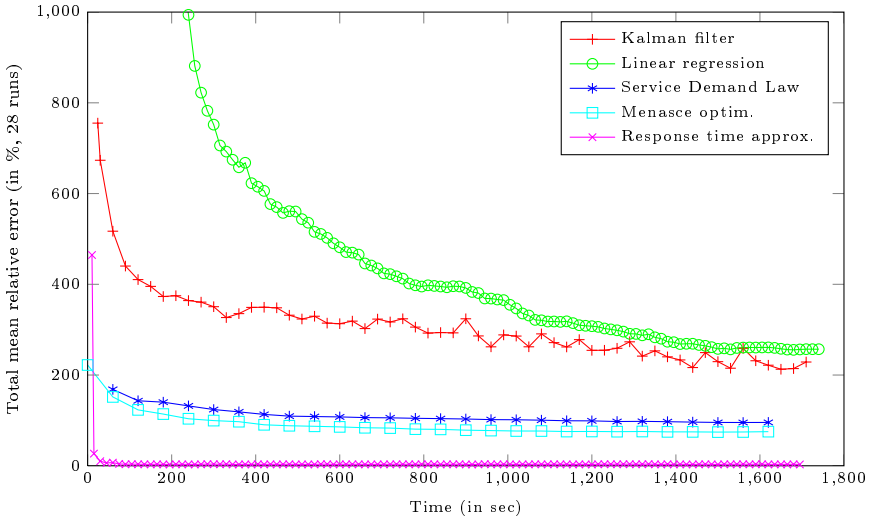
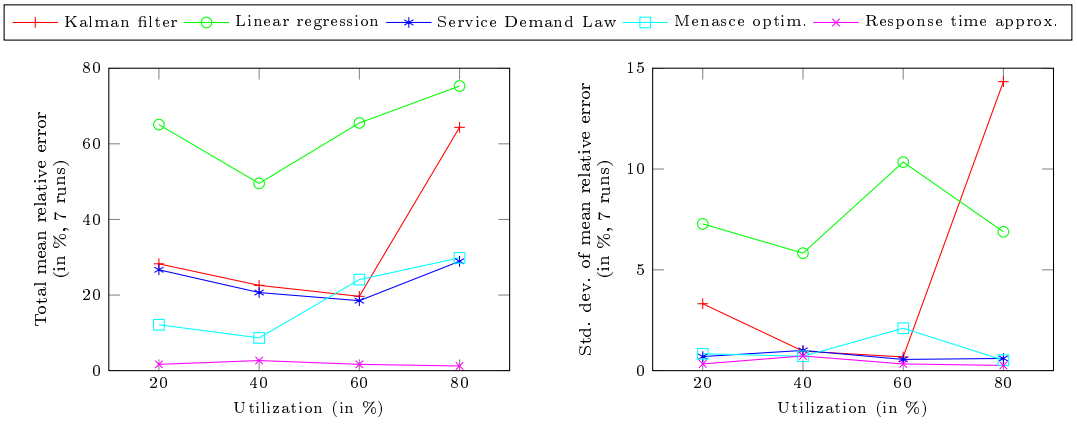


Figure 7.3.: Temporal development of the total mean relative error.

We start with an analysis of the convergence behavior of the estimation approaches. Figure 7.3 depicts the development of the total mean relative error over time. In this experiment, the response time approximation approach has by far the best convergence of all estimation approaches. It provides almost instantly very accurate estimates of the resource demands. However, we should keep in mind that this experiment offers ideal conditions for the response time approximation approach. Furthermore, we should assume that the resource demands generated by Ginpex are slightly scattered so that the response time approximation approach might underestimate the actual resource demands because it relies on the minimum observed response time. Kalman filter, linear regression and Service Demand Law have a similar convergence behavior. In under five minutes, they provide estimates that are close to their final estimate in. Afterwards, they still improve their resource demand estimates, but the improvements are only small. The convergence of the linear regression approach is significantly slower. The mean relative error of linear regression is very high in the first five minutes. The mean relative error then slowly converges to a level comparable to the Kalman filter approach.



(a) Sum of mean relative error

(b) Standard deviation of mean relative error

Figure 7.4.: Total mean relative error at varying utilization levels.

Figure 7.4 shows the total mean relative error for each level of utilization separately. The total mean relative error significantly varies with different utilization levels. Only the re-

sponse time approximation approach has a constantly low relative error over all utilization levels. Since we approximate the resource demand with the minimum observed response time, one request that experiences no delays due to resource contention is sufficient to obtain an accurate resource demand estimate for a workload class. Such requests that are not delayed are even observed at high utilization levels of around 80%. If we approximate the resource demands with average response times, the estimation error in this experiment is expected to be significantly higher at high utilization levels.

The linear regression approach has the highest total mean relative error. At the same time it has also the highest standard deviation of the relative error. These high errors are caused by a very slow convergence of the linear regression in some of the experiment runs. If we extend the observation period, the resource demand estimates of linear regression might be better. We assume that the slow convergence is caused by collinearities in the throughput measurements. In some experiments the coefficient of correlation between the throughput measurements of two workload classes were up to 0.6. However, we could not determine a clear relationship between correlation coefficients and the mean relative error.

The Kalman filter has a peak at a utilization of 80%. This peak is primarily caused by one experiment run where the Kalman filter has convergence issues, as we can see in Figure 7.5(d). In this case the estimation approach highly overestimates the resource demands at the start although we provided very low initial estimates. Then the convergence issues already described Section 7.1.1.2 apply because the calculated utilization is close to 100%.

We conclude that response time approximation, Service Demand Law and Menascé optimization provide the best and most reliable estimates in the base experiments with a mean relative error of below ten percent in all experiment runs. The Kalman filter approach can have convergence problems if the utilization is high (above 80%). The estimation accuracy of linear regression has a high variance compared to the other approaches and its resource demand estimates are therefore less reliable.

7.1.3. Sensitivity Analyses

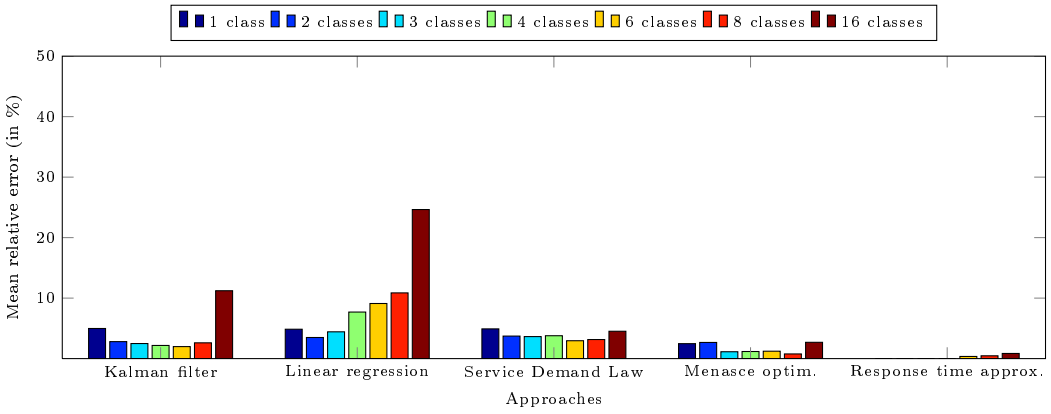
7.1.3.1. Collinear Workload

Linear regression is known to be prone to multicollinearities between the control variables. We are primarily interested in how the Kalman filter approach can cope with collinearities in the throughput measurements. We do not consider other estimation approaches in this experiment because it can be argued that these estimation approaches are not influenced by multicollinearities.

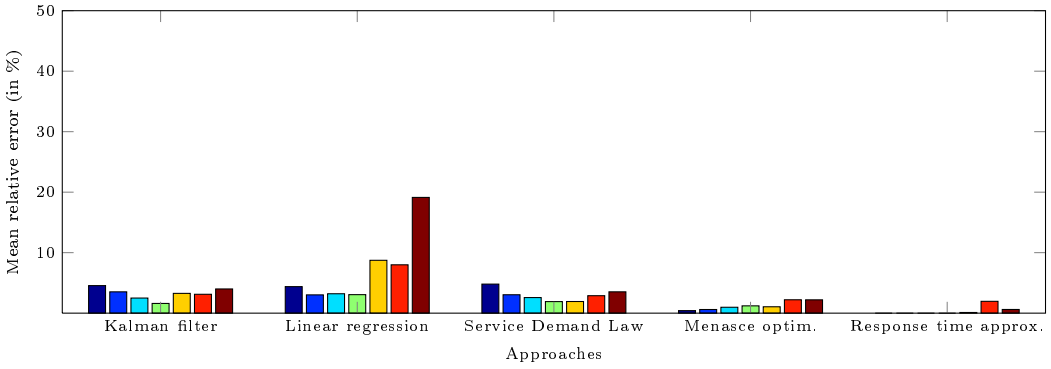
We used the collinear workload described in Section 5.4.1.1. The workload consisted of six classes causing a utilization of 40% on average. We repeated this experiment while varying the number of collinear workload classes between 2, 3 and 6 classes.

Figure 7.6 shows the mean relative error for each experiment run. As expected, the linear regression is significantly influenced by the collinear throughput measurements. It fails to distribute the observed utilization between the workload classes. The resource demand estimates have a high variance over time. The coefficient of covariation of the estimates of one workload class over time is up to 600% and the estimates never settle down to a stable value. The resource demand estimates of some workload classes temporarily collapse to zero.

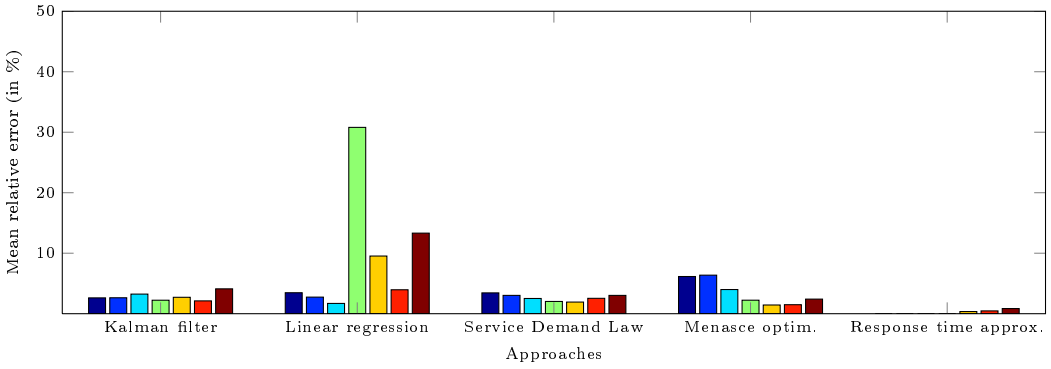
Compared to linear regression, the collinear throughput measurements do not clearly influence the resource demand estimates of the Kalman filter approach. The mean relative error is always below 3% and does not change when varying the level of collinearity in the throughput measurements. Although the Kalman filter also uses the Utilization Law in



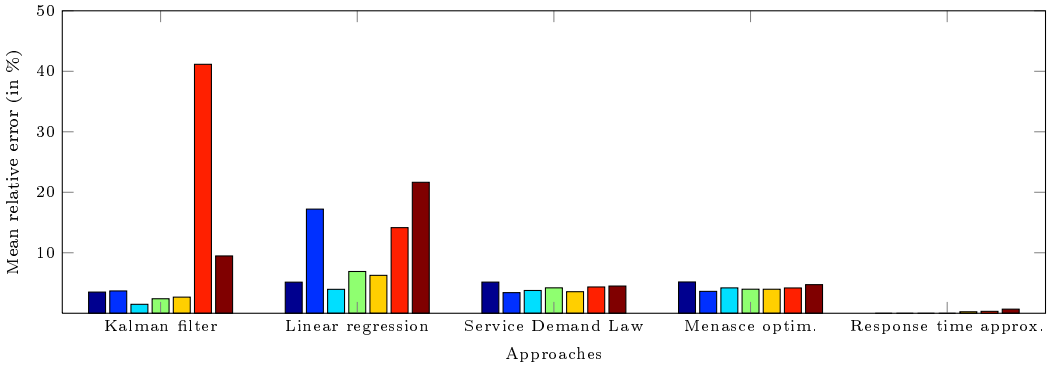
(a) 20% utilization



(b) 40% utilization



(c) 60% utilization



(d) 80% utilization

Figure 7.5.: Mean relative errors with the artificial workload.

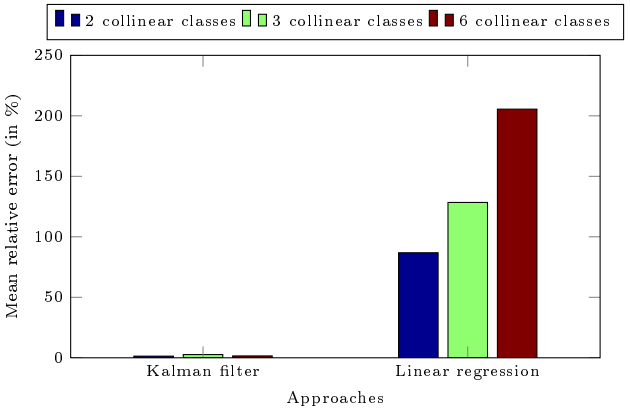


Figure 7.6.: Mean relative error in the presence of multicollinearities.

its measurement model (Equation 3.1.4 on page 17) it is able to compensate the loss of information due to collinear throughput measurements. We assume that the inclusion of response time measurements in the measurement model makes it robust to multicollinearities.

We conclude that only the linear regression approach shows a high sensibility to collinear throughput measurements. This result for the linear regression corresponds with earlier experiences described by Pacifici et al. [PSST08] and Casale et al. [CCT08]. The Kalman filter approach can cope well with the type of collinear workload used in this experiment.

7.1.3.2. Workload with Additional Wait Phases

In this experiment, we use the workload described in Section 5.4.1.1 where an additional pause of a certain duration is inserted in the middle of a task. This pause could stand for delays incurred by hardware or software contention, e.g., the waiting time for an I/O device. The workload consists of four classes and causes a CPU utilization of about 40% on the experiment machine. Only two of the workload classes have an additional wait included, one of these workload classes contains a wait T_{wait} and the other class $2 \cdot T_{wait}$. We chose this distribution of the waiting times so that the responses time of a workload class are not proportional to the resource demand. We did three experiment runs while varying T_{wait} between 20 ms, 50 ms and 100 ms.

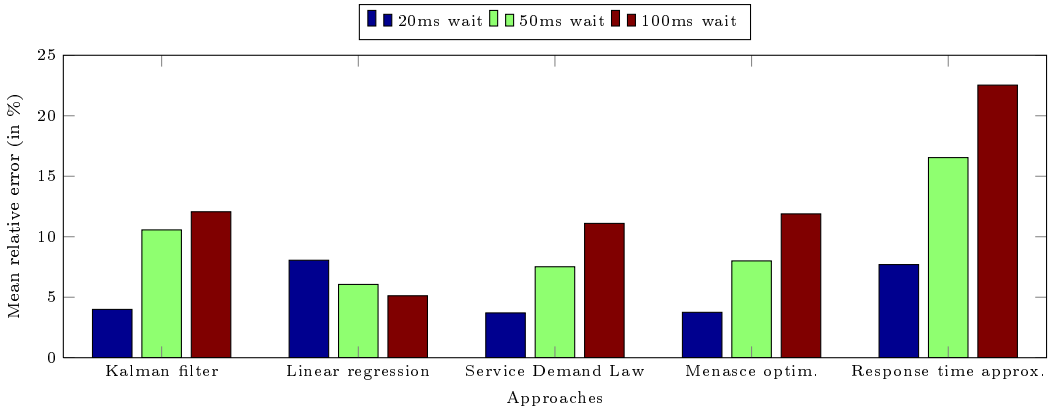


Figure 7.7.: Mean relative error in the presence of additional wait phases.

Figure 7.7 shows the results of this experiment. It depicts the mean relative error of each estimation approach and each experiment run. The additional wait influences primarily

the measured response time. Therefore, the estimation accuracy of the linear regression approach, which is based on the Utilization Law, is not impaired. The other estimation approaches all reveal a sensitivity to the duration of the wait because they depend on response time measurements. The response time approximation approach is influenced most by the additional wait depending solely on response time measurements. The other three approaches, namely Kalman Filter, Service Demand Law and Menascé optimization, show similar behavior. The relative mean error of the Service Demand Law approach rises with higher waiting times because we use the apportioning scheme described by Brosig et al. [BKK09]. This apportioning scheme is based on the assumption that the response times are proportional to the resource demands. This is not the case in this experiment. Other apportioning methods might be more appropriate here.

This experiment shows that delays due to software or hardware contention can have a significant influence on the estimation accuracy. All estimation approaches relying on response time measurements are affected negatively by additional delays. The estimation accuracy is inversely proportionally to the length of the delay. Delays during the processing of requests due to software or hardware contention need to be identified and considered specifically during resource demand estimation.

7.1.3.3. Workload with Background Jobs

In this experiment, we used the workload described in Section 5.4.1.1 where an additional background job is included, which causes a certain load on the monitored system. The four workload classes cause a utilization of 40%. The background job is not included in any of the four workload classes. It represents any kind of unforeseen work that the performance engineer did not include in his model. We did four experiment runs varying the utilization due to the background job between 5%, 10%, 20% and 40%.

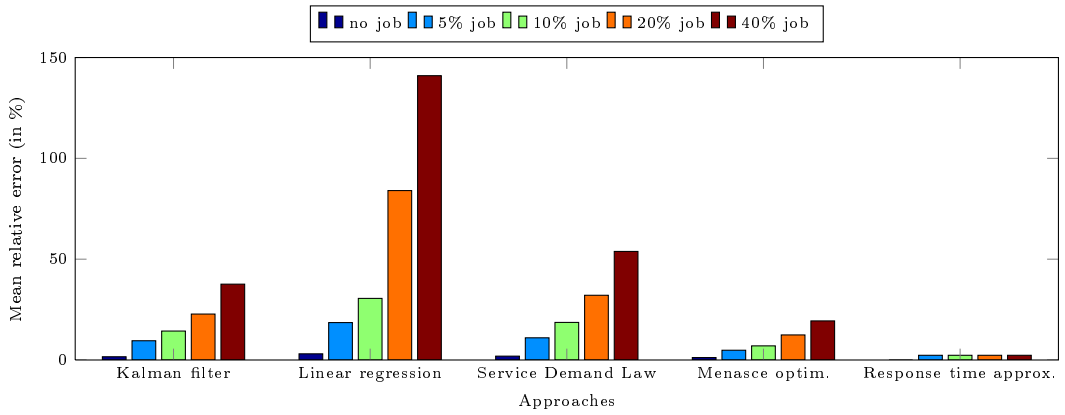


Figure 7.8.: Mean relative error in the presence of background jobs.

Figure 7.8 shows the results of this experiment. In general, all estimation approaches except for the response time approximation are negatively influenced by background jobs. A background job primarily affects the utilization measurements. If the background job causes a substantial utilization, the response times are also influenced because requests have to wait longer for resources. Therefore, estimation approaches solely based on response time and throughput measurements, e.g., the Menascé optimization approach, are less influenced by background jobs than approaches based on utilization measurements.

Figure 7.9 shows the development of the total mean relative error of all experiment runs over time. The Kalman filter approach continuously adapts its resource demand estimates during phases of high load due to a running background job. It distributes the resource consumption between the workload classes resulting in higher resource demand estimates

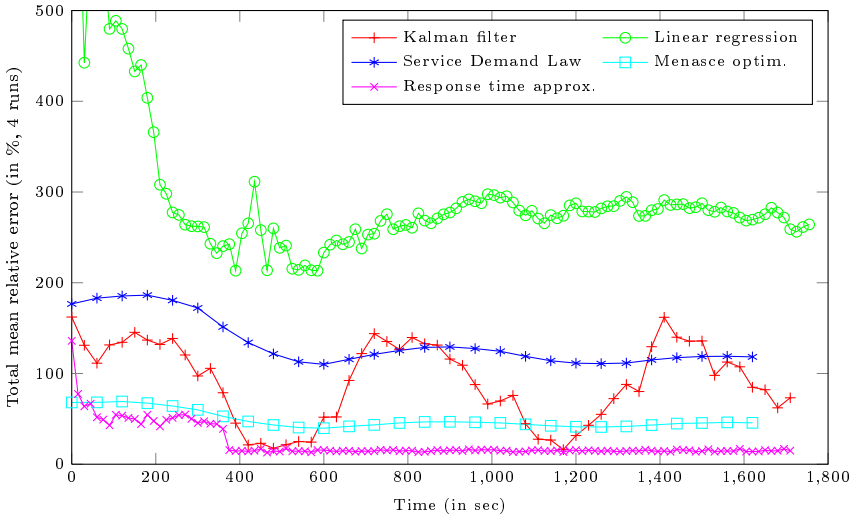


Figure 7.9.: Temporal development of the total mean relative error with background jobs.

when a background job is running. When no background job utilizes the CPU, it quickly reduces its estimates again. This illustrates the ability of the Kalman filter to track time-varying resource demands. However, it might not be our intention that the Kalman filter changes its estimates in presence of background jobs. Other estimation approaches do not show this behavior because they take all measurements equally into account.

We conclude that unforeseen background jobs can significantly influence the results of all estimation approaches. Estimation approaches relying on utilization measurements are affected more by background jobs than estimation approaches solely based on response time measurements. At any rate, a performance engineer must consider background jobs that are possibly running on a system when estimating resource demands.

7.1.4. Discussion

We showed with a number of experiments, that the accuracy of an estimation approach can be significantly influenced by the workload characteristics. None of the considered approaches to resource demand estimation yield accurate estimates in all situations. In the base experiment, the mean relative error is mostly below 10%. When varying the workload characteristics, we observe mean relative errors of almost 200% in the worst case.

Shortly summarized, we could show that the following workload characteristics have a significant influence on the estimation accuracy:

- If the utilization of the monitored system is around 80% or higher, the Kalman filter design of Kumar et al. [KTZ09] has convergence issues in some cases.
- Multicollinearities in the throughput measurements only influence the estimation accuracy of the linear regression approach negatively.
- Additional delays due to software or hardware contention reduce the accuracy of estimation approaches relying on response time measurements.
- Background jobs that are not included in the workload model can cause problems with estimation approaches based on utilization measurements.

A performance engineer needs to consider these workload characteristics when choosing an approach to resource demand estimation.

7.2. Scenario B: TPC-W Benchmark

In the following scenario, we used the TPC-W application benchmark in order to analyze the behavior of the approaches to resource demand estimation with realistic workloads. We want to compare the predictive estimation accuracy of different approaches to resource demand estimation regarding utilization and response time predictions. Furthermore, this scenario extends the evaluation to environments with resources.

The TPC-W benchmark is designed as a three-tier internet application. We deployed the TPC-W application benchmark on two servers. The front-end server hosted the presentation and logic tier, the back-end server contained a database. See Section 5.4.2.3 for details about the experiment setup. We estimated the resource demands for the CPUs of the front-end and back-end server. We identified these two resources as potential bottlenecks under heavy workload. Other resources are not considered in the performance model. See Section 5.4.2.4 for an in-depth description of the performance model.

We started with one experiment run in which we applied a workload whose arrival pattern follows that of the FIFA 98 world cup workload described in Section 5.4.2.2. During the experiment run we monitored the utilization of each system as well as the throughput and response times of each workload class over a period of two hours. The measurements were obtained with a sampling interval of one second. Based on these measurements, we obtained resource demand estimates for each resource and each workload class. As we showed in Section 7.1.1.3, the estimation accuracy depends on the length of the monitoring window. Therefore, we used different monitoring window sizes for each estimation approach. We directly used the measurements with a sampling rate of one second for linear regression . We averaged the measurements over 30 seconds for the Kalman filter and over 60 seconds for the other estimation approaches. Furthermore, we removed the first five minutes from the measurement traces to ensure that the system is in a steady state.

The estimated resource demands are used to parameterize the QPN model in Figure 5.7 on page 49. After model parameterization we have a separate model for each estimation approach. We simulated all models with the SimQPN 2.0 tool, which offers an efficient and well-proven simulator for QPN models [KB06, KSM10]. We used the batch means method for steady state analysis. The simulation stopped if the relative precision of the response time is below 2% at a confidence level of 98%. We repeated the simulation with 400, 600, 800 and 1000 clients in the system. Afterwards, we carried out a number of experiment runs with the standard workload of the TPC-W benchmark with these numbers of EBs and measured the average utilization and response time at the system.

First experiments runs revealed high errors in the predicted response times. We identified the network as a source of significant delays. Each request at the web-server causes several database queries requiring a round-trip between front-end and back-end server. A static analysis of the TPC-W source code showed that one HTTP request for a TPC-W transaction results in approximately six database queries on average. Under load, the measured round-trip time between front-end and back-end server is approximately 0.5 milliseconds on average. Assuming that all database queries are executed sequentially, we estimate the total network delay to approximately three milliseconds. The estimated resource demands at the front-end and back-end servers are all between one and two milliseconds and the network delays cannot be neglected. Therefore, we subtract a constant delay of three milliseconds from all measured response times before using them for resource demand estimation. Thus, we exclude that this delay is erroneously attributed to the resource demands of the CPUs of the servers by estimation approaches relying on response time measurements. We added an additional place with IS scheduling to the performance model with a constant delay of three milliseconds.

(a) 400 EBs.

	U_{db}	U_{web}	$R_{browse}[ms]$	$R_{order}[ms]$
Measured	0.28	0.28	6.22	8.29
Kalman filter	0.26 (8.84%)	0.29 (3.86%)	5.42 (12.81%)	7.25 (12.61%)
Service Demand Law	0.25 (12.46%)	0.29 (1.19%)	5.60 (9.94%)	6.81 (17.82%)
Linear regression	0.25 (12.42%)	0.29 (2.11%)	6.46 (3.84%)	5.93 (28.48%)
Menasce optim.	0.33 (18.15%)	0.33 (16.10%)	6.54 (5.10%)	8.14 (1.83%)

(b) 600 EBs.

	U_{db}	U_{web}	$R_{browse}[ms]$	$R_{order}[ms]$
Measured	0.37	0.42	7.64	9.67
Kalman filter	0.39 (3.53%)	0.44 (5.55%)	5.99 (21.63%)	8.27 (14.45%)
Service Demand Law	0.37 (0.42%)	0.43 (3.84%)	6.21 (18.70%)	7.69 (20.42%)
Linear regression	0.37 (0.88%)	0.42 (1.74%)	7.19 (5.91%)	6.59 (31.90%)
Menasce optim.	0.49 (32.85%)	0.49 (18.07%)	7.63 (0.14%)	9.71 (0.46%)

(c) 800 EBs.

	U_{db}	U_{web}	$R_{browse}[ms]$	$R_{order}[ms]$
Measured	0.53	0.57	10.43	13.50
Kalman filter	0.51 (3.65%)	0.59 (3.93%)	6.93 (33.56%)	9.97 (26.11%)
Service Demand Law	0.50 (7.19%)	0.58 (2.00%)	7.18 (31.18%)	9.05 (32.92%)
Linear regression	0.49 (7.99%)	0.57 (0.45%)	8.35 (19.97%)	7.55 (44.05%)
Menasce optim.	0.66 (23.07%)	0.66 (16.56%)	9.87 (5.37%)	13.05 (3.30%)

(d) 1000 EBs.

	U_{db}	U_{web}	$R_{browse}[ms]$	$R_{order}[ms]$
Measured	0.60	0.71	15.62	19.14
Kalman filter	0.64 (6.83%)	0.73 (3.80%)	8.70 (44.33%)	13.21 (30.96%)
Service Demand Law	0.62 (2.48%)	0.71 (0.93%)	8.77 (43.85%)	11.44 (40.25%)
Linear regression	0.62 (2.10%)	0.71 (0.54%)	10.65 (31.79%)	9.62 (49.72%)
Menasce optim.	0.82 (35.90%)	0.82 (15.86%)	15.74 (0.79%)	21.63 (13.02%)

Table 7.1.: Prediction results for the TPC-W order mix in a native environment.

7.2.1. Results

Table 7.1 compares the simulation results with corresponding measurements at the real systems. We consider the CPU utilization of the front-end server U_{web}, the CPU utilization of the back-end server U_{db}, the average response time R_{browse} of the browse workload class and the average response time R_{order} of the order workload class. The table contains the measured values as well as the simulation results for each estimation approach. For each simulation result the table shows the absolute values as well as the relative deviation from the measured values in parentheses. The relative deviation is defined as

$$\Delta_X = \frac{|X_{simulated} - X_{measured}|}{X_{measured}} \tag{7.2}$$

where X is one of the previously listed performance measures. The response time approximation approach is not considered in this scenario because our implementation only supports a single resource.

The workload used to obtain measurements for the resource demand estimation caused a utilization of approximately 30-40%. Therefore, all estimation approaches also provided the best results with 400 and 600 EBs. The Service Demand Law approach and the Kalman filter approach provide the best overall results at these load levels with utilization errors of 12% and below and response time errors of mostly below 20%. In general, the accuracy of the predictions of these two approaches are very similar. Both yield resource demand estimates that predict the utilization of the system best, but fail to adequately explain the response times at higher load levels with 800 EBs and more.

The linear regression approach provide resource demand estimates that can predict the utilization of both systems well with relative errors of below 10% for almost all load

levels. However, it seems as if the linear regression struggles to attribute the observed load to the two workload classes correctly. The deviation of R_{browse} from the measured response times is significantly lower than the deviation of R_{order}. We can explain this with multicollinearities in the throughput measurements. The correlation coefficient between the throughput of order requests and the throughput of browse requests is always close to one. This is because we only varied the total arrival rate according to the arrival patterns of the FIFA 98 World Cup workload. We did not change the mix between workload classes over time. According to Stewart et al. [SKZ07], linear regression works better if the mix of the workload classes changes over time.

The Menascé optimization approach yields resource demand estimates that lead to the best response time predictions. In most cases the response time error is 5% or less. Only in the case with 1000 EBs the response time error is above 10% for one workload class. However, it significantly overestimates the utilization of both systems in all cases. We assume that the measured response times include some additional, unknown delays that cannot be attributed to the CPU of the system. Nevertheless, the Menascé optimization approach includes these delays in the resource demand estimates of the CPU, as we have already shown in Section 7.1.3.2. These overestimated resource demands lead to significant errors when predicting the utilization of a system. Furthermore, we observed that the resource demand estimates are essentially the same for all resources. There are only differences between resource demand estimates of different workload classes. If no predetermined resource demands are provided, the Menascé optimization approach gets only measurements that describe the performance behavior of both resources, such as average response time and throughput. It does not have any information specific to one resource. Therefore, it does not have any clue how the work is distributed between the two resources and assumes that the work is equally distributed between the two resources. For each resource, it requires at least one predetermined resource demand of any workload classes. Otherwise, the utilization predictions are meaningless. Other estimation approaches based on optimization that also include utilization measurements in their model [LXMZ03, LWXZ06] might be more appropriate for cases with several resources.

7.2.2. Discussion

We conclude that the predictive accuracy can significantly vary between the considered estimation approaches with a realistic workload. Especially, if the performance model does not represent each source of delays individually, which is often not feasible due to the complexity of modern computer systems, we get different resource demand estimates. Generally speaking, approaches based on utilization measurements, such as Service Demand Law and linear regression, provide estimates which yield more accurate predictions of the utilization of a system. In contrast, approaches based on response time measurements, such as Menascé optimization, are better suited when predicting the response time of a system. Although, the Kalman filter includes both types of measurements in its model, its predictive accuracy in this scenario is comparable to that of approaches solely based on utilization measurements.

7.3. Scenario C: Multi-core Processor

In Section 7.1, we ran the experiments on a single-core processor machine. However, multi-core processors are the standard in most computer systems today. In this scenario, we repeated the base experiment, which is described in Section 7.1.2, with a multi-core processor. First, we used the same tasks as in Section 7.1 where all work is done sequentially. Then we changed the workload so that the tasks split their work between several threads. Section 5.4.3 contains a description of this parallel workload. We monitored the

global utilization over all enabled cores, i.e., 100% utilization means that all cores of the processor are fully utilized. In the following, we describe the results of these experiments.

7.3.1. Sequential Tasks

We carried out several experiment runs with four workload classes and a utilization level of 40%. We varied the number of enabled cores of the processor between 1 and 4.

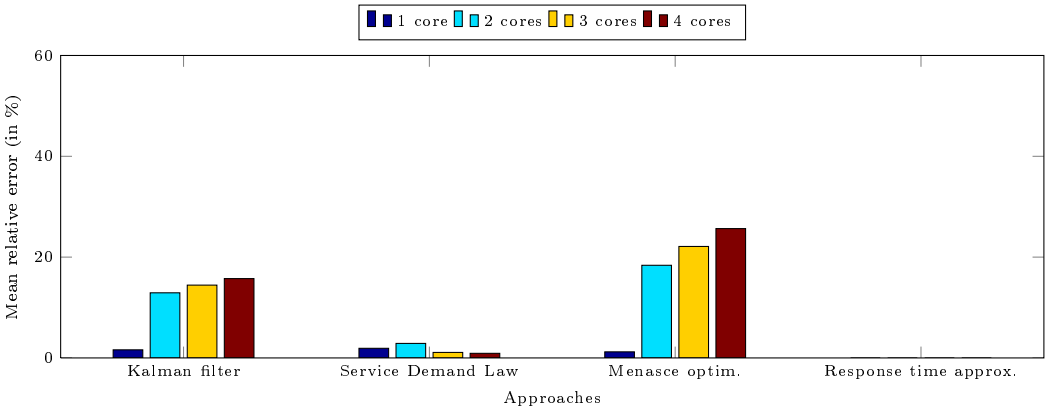


Figure 7.10.: Mean relative error with a varying number of processor cores.

Figure 7.10 shows the mean relative error when varying the number of enabled cores. Response time approximation is not influenced by the number of enabled processor cores. Since the tasks in this experiment are all sequential, the response times of the individual tasks are not reduced by enabling more processor cores. More enabled processor cores only improve the maximum possible throughput.

The Service Demand Law for resources with multiple servers is defined as

$$D_{i,c} = \frac{P_i \cdot U_{i,c}}{X_{0,c}} \tag{7.3}$$

where P_i is the number of servers at resource i [MDA04]. We extended the Service Demand Law approach accordingly. We set P_i to the number of enabled processor cores. The resulting resource demand estimates are very accurate with estimation errors below 5%. This relationship appropriately describes the behavior of a multi-core processor in this experiment. We did not consider the linear regression approach here. It is based on the Utilization Law, which in turn can be directly derived from the Service Demand Law. Therefore, we do no expect any influence of multi-core processors on the estimation accuracy of linear regression.

Kumar et al. already consider systems with multiple processors in their measurement model for the Kalman filter [KTZ09]. Essentially, they define the response time as

$$R_{i,c} = \frac{D_{i,c}}{1 - \frac{1}{P_i} \sum_{d=1}^{C} \lambda_d \cdot D_{i,d}} \tag{7.4}$$

where P_i is the number of servers of resource i. However, the experiment results show that this relationship does not correctly explain the observed response times. As a result, the estimation accuracy of the Kalman filter declines substantially with several processor cores resulting in estimation errors between 12% and 15%.

We used the response time equation in Equation 7.4 for the Menascé optimization approach as well. The relative error of the Menascé optimization approach is even worse than that of the Kalman filter. It is between 18% and 25%. We assume that the Kalman filter can compensate a part of the error caused by the insufficient model with the additional information from the utilization measurements.

7.3.2. Parallel Tasks

In this experiment we enabled all four cores of the processor and varied the workload type instead. We used parallel tasks that distribute the calculations over several threads. A task can utilize several cores of the processor simultaneously. Therefore, the response time can be substantially reduced on multi-core processors. The workload consisted of four workload classes that caused a total utilization of approximately 40% on average. We carried out several experiment runs while varying the number of threads between 1 and 8.

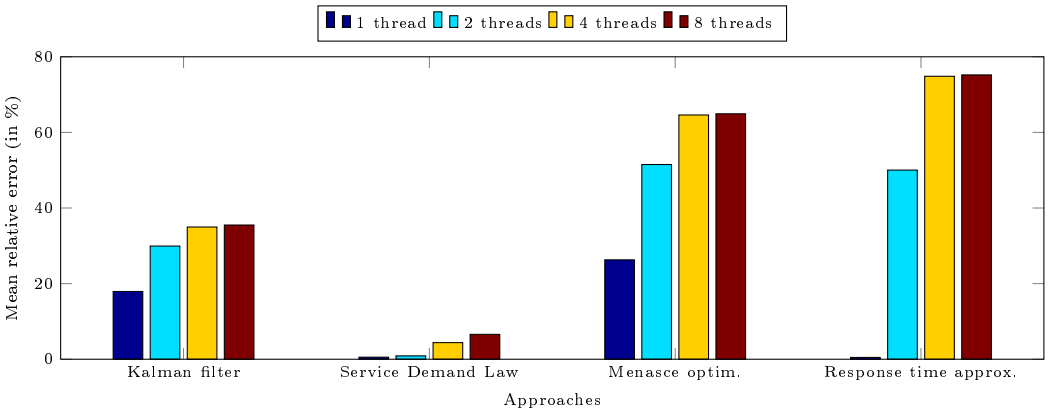


Figure 7.11.: Mean relative error with a varying number of threads.

Figure 7.11 depicts the mean relative error of each estimation approach with varying number of threads. All estimation approaches that rely on response time measurements, namely Kalman filter, Menascé optimization and response time approximation, are negatively affected by parallel workloads. If the parallel tasks use two or more threads, the mean relative error of these approaches lies between 30% and 80%. We need more sophisticated models if requests are processed by several threads simultaneously.

The mean relative error of the Service Demand Law approach is relatively low, below 10% in all experiment runs. We can observe only a slight increase in the mean relative error with more threads. We assume that this is due to a higher scheduling overhead at the operating system level.

7.3.3. Discussion

We conclude that approaches to resource demand estimation that primarily rely on utilization measurements, such as linear regression and Service Demand Law, work better with multi-core processors and parallel processing than estimation approaches using response time measurements, such as the Kalman filter and Menascé optimization. As we showed in this scenario, Equation 7.4 insufficiently describe systems with multi-core processors and parallel processing. Future research is necessary to extend estimation approaches based on response times to support parallel systems.

7.4. Scenario D: Virtualization

In this scenario, we analyze the estimation accuracy of different approaches to resource demand estimation in virtualized environments. The general experiment procedure is the same as that in Section 7.2. We also used the TPC-W application benchmark, but in this scenario we deployed it on virtualized servers instead. The front-end and back-end servers both ran in a separate VM on the same physical computer. See Section 5.3.2 for details about the setup of the virtualized servers. The workload used here is the same as in Section 7.2.

(a) 400 EBs.

	U_{db}	U_{web}	$R_{browse}[ms]$	$R_{order}[ms]$
Measured	0.21	0.25	10.72	11.73
Kalman filter	0.22 (3.75%)	0.24 (6.83%)	5.25 (51.02%)	5.88 (49.87%)
Service Demand Law	0.24 (15.27%)	0.30 (17.17%)	5.95 (44.45%)	6.53 (44.31%)
Linear regression	0.25 (17.16%)	0.30 (19.38%)	7.88 (26.49%)	4.73 (59.68%)
Menasce optim.	0.52 (147.81%)	0.51 (101.37%)	11.67 (8.93%)	12.90 (9.93%)

(b) 600 EBs.

	U_{db}	U_{web}	$R_{browse}[ms]$	$R_{order}[ms]$
Measured	0.39	0.47	30.85	36.63
Kalman filter	0.33 (15.38%)	0.35 (24.32%)	5.64 (81.72%)	6.38 (82.58%)
Service Demand Law	0.36 (6.39%)	0.45 (4.65%)	6.59 (78.63%)	7.32 (80.03%)
Linear regression	0.37 (5.65%)	0.45 (3.83%)	8.87 (71.26%)	5.20 (85.79%)
Menasce optim.	0.78 (101.84%)	0.76 (63.17%)	21.27 (31.04%)	24.08 (34.25%)

(c) 800 EBs.

	U_{db}	U_{web}	$R_{browse}[ms]$	$R_{order}[ms]$
Measured	0.58	0.86	222.88	303.75
Kalman filter	0.43 (24.91%)	0.47 (45.47%)	6.16 (97.24%)	7.10 (97.66%)
Service Demand Law	0.48 (16.80%)	0.59 (31.72%)	7.73 (96.53%)	8.63 (97.16%)
Linear regression	0.49 (15.62%)	0.60 (31.05%)	10.53 (95.28%)	5.99 (98.03%)
Menasce optim.	0.98 (70.69%)	0.96 (11.15%)	114.58 (48.59%)	131.21 (56.80%)

Table 7.2.: Prediction results for the TPC-W order mix in a virtualized environment.

7.4.1. Results

Table 7.2 compares the simulation results with corresponding measurements at the virtualized systems. The table does not include the results with 1000 EBs because the system is overloaded with that number of simultaneous clients. We used the same constant delay as in Section 7.2 that captures the influence of the network. Round-trip measurements at the virtualized systems yielded approximately the same delay under load.

The load curve of the virtualized systems is different than that of their native counterparts. With 400 EBs the utilization of both systems is slightly lower in the virtualized environment. In contrast, the utilization is significantly higher in the virtualized systems with 600 and 800 EBs. The CPU utilization of the virtualized system shows a non-linear behavior. Therefore, all estimation approaches fail to provide resource demand estimates that can be used to accurately predict the CPU utilization at different load levels. The estimates from the Kalman filter predicts the CPU utilization best in the case with 400 EBs. Linear regression and the Service Demand Law provide the best estimates with 600 EBs.

We measured only the CPU utilization in the VMs and did not consider the CPU utilization of the Xen Dom0 in this scenario. However, the TPC-W benchmark causes a considerable amount of network traffic. Furthermore, databases require a lot of hard disk accesses. These I/O activities result in a certain CPU utilization of the Xen Dom0. For instance, a typical webserver can cause 20%-45% CPU overhead in the Xen Dom0 [CG05]. A performance engineer must consider this CPU overhead when predicting the performance of a virtualized system. Especially, we have to attribute the CPU overhead in the Xen Dom0 due to I/O activities to each VM in order to be able to predict the utilization of the Xen Dom0 correctly [LZJ+11].

The measured average response time in Table 7.2 cannot be explained with any of the estimated resource demands. The Menascé optimization approach can provide estimates that work well with 400 EBs. However, the measured average response times with 600 and 800 EBs are 30%-50% higher than those predicted with the resource demand estimates of the Menascé optimization approach. Furthermore, the resource demand estimates of

the Menascé optimization approach clearly fail to explain the utilization of the VMs. The virtualization platform seems to infer additional delays. These delays have a great influence on the observed response times.

7.4.2. Discussion

We showed in this scenario that virtualized environments are a challenge for the estimation of resource demands. In particular, we identified the following issues when applying the considered estimation approaches in a virtualized environment:

- The utilization in the VMs has a non-linear relationship to the system load. We might require estimation approaches for load-dependent resource demands.
- The virtualization overhead needs to be attributed to the VM that caused it. We need to use methods to apportion the virtualization overhead between the VMs, such as described in [CG05].
- The virtualization platform causes additional delays. The considered approaches to resource demand estimation cannot be used to estimate these delays.

7.5. Concluding Remarks

In this chapter, we presented the evaluation results of different approaches to resource demand estimation. We showed that there is no optimal approach to resource demand estimation for all situations regarding the estimation accuracy. External conditions, such as multi-collinearities in the workload, background jobs or delays due to hardware and software contention, have a significant influence on the accuracy of the estimation approaches. A performance engineer needs to choose an estimation approach depending on the characteristics of a given application scenario.

We also showed that multi-core processors can negatively influence the accuracy of estimation approaches, particularly with highly parallel workloads. Especially, this is an issue with estimation approaches that rely on response time measurements. Existing estimation approaches need to be adapted to reflect the performance characteristics of multi-core processors and parallel workloads.

Virtualized environments pose a major challenge to resource demand estimation. The accuracy of the estimation approaches significantly decreases with virtualized systems as shown with experiments in a realistic environment running the TPC-W benchmark. Future research needs to analyze the influence of the virtualization platform on the estimation approaches and it might be necessary to develop estimation approaches that specifically consider the characteristics of the virtualization platform.

8. Summary and Outlook

8.1. Summary

In this thesis, we classified existing approaches to resource demand estimation and evaluated them in a realistic experiment environment. The classification consists of an overview of the state-of-the-art in resource demand estimation, a classification scheme for approaches to resource demand estimation and a categorization of the existing approaches according to the classification scheme. As part of the evaluation, we implemented a subset of the estimation approaches and conducted a series of experiments comparing the accuracy of the estimation approaches we implemented.

We carried out a comprehensive literature research on the topic of resource demand estimation and identified a number of different estimation approaches. We focused our literature research on approaches that depend only on measurements that can be obtained from a running system without intrusive application instrumentation. The identified estimation approaches use different models to describe the relationship between the observable measurements and the "hidden" resource demands and rely on different mathematical methods to infer estimates for the resource demands. Mathematical methods commonly used for resource demand estimation are linear regression, Kalman filtering, mathematical optimization and ICA. The diverse models and mathematical methods imply different assumptions about the system structure and the workload. We gave an overview of the state-of-the-art in resource demand estimation and described each estimation approach including its intrinsic assumptions and limitations.

The findings of the literature research are incorporated in the classification scheme for approaches to resource demand estimation that we developed. The classification scheme consists of a set of dimensions. Each dimension describes a major aspect of an approach to resource demand estimation. We considered the following dimensions in our classification scheme: input parameters, output metrics, robustness, accuracy, supported types of resources, support for virtualization, applicability and existing evaluation. The dimensions capture points of variation between the different approaches to resource demand estimation. Thus, the classification scheme enables the systematic comparison of estimation approaches and facilitates the selection of an estimation approach that fits best in a given application scenario. Furthermore, it points out future research directions, such as the high-level estimation of resource demands of I/O devices or the support for new features of modern CPUs.

The goal of the evaluation was the comparison of the estimation accuracy of different estimation approaches under varying conditions. We selected a subset of estimation approaches for this evaluation. The focus lay on estimation approaches that can be used for online estimation in productive environments. Furthermore, we ensured that at least one representative of each major model and mathematical method is included in the set of evaluated estimation approaches. The following estimation approaches were considered in the evaluation: response time approximation, Service Demand Law, linear regression, Kalman filter and Menascé optimization.

We designed and implemented a tool for the estimation of resource demands. The tool provides implementations of the estimation approaches we selected for evaluation. It relies on measurements from the Linux system monitoring tool `sar`, from the Ginpex framework [HKHR11] and from web server access logs. The tool can directly import measurement traces as produced by these programs. It contains a set of preprocessors to prepare the raw measurement data for resource demand estimation. Then it invokes a configurable number of estimation approaches and stores the results of each estimation approach for subsequent analyses.

A number of experiments where we used the Ginpex framework to generate controlled workloads showed that the estimation accuracy is significantly influenced by external conditions. Under optimal conditions the estimation error of all estimation approaches is mostly below 10%. Factors, such as multicollinearities in the workload, background jobs, which are not included in the performance model, or delays due to hardware or software contention, influence the results of the evaluated estimation approaches in different ways. With these disturbing factors, the estimation error of some estimation approaches increases up to a maximum of 200%. None of the evaluated estimation approaches works well in all situations. Therefore, we see the need for certain metrics that help to assess the reliability of approaches to resource demand estimation, e.g., robust confidence intervals for the estimated resource demands.

We applied the different approaches to resource demand estimation in a representative scenario using the TPC-W benchmark application and real workloads from the FIFA 98 world cup web site. The resource demand estimates produced by the different approaches were used to predict the utilization and the average response time of the web server and the database server at different load levels. With the Service Demand Law, linear regression and Kalman filter approaches, we obtained similar predictions for the utilization with a relative error of mostly below 10%. However, their response time predictions showed high relative errors. Especially at high load levels with a utilization between 50% and 70%, the relative errors of the response time predictions are above 30%. In contrast, the Menascé optimization approach provides resource demand estimates that accurately predict the response times at different load levels with a relative error of mostly below 10%. However, this comes at the cost of higher prediction errors for the utilization compared to the other estimation approaches. We conclude that depending on the estimation approach resource demands are optimized either for utilization or for response time predictions.

Our evaluation included scenarios with multi-core processors and virtualization. In the case of multi-core processors, we analyzed the accuracy of the estimation approaches with different number of cores and different levels of parallelism. The accuracy of estimation approaches relying on response time measurements significantly decreased with more processor cores and higher levels of parallelism.

In our virtualization scenario, we repeated the experiments with the TPC-W benchmark deployed on virtualized servers. This scenario showed that the evaluated estimation approaches fail to provide resource demands that can be used to predict the utilization and the average response time at different load levels.

8.2. Future Work

In this section, we point out future research directions. We first describe future work to enhance our classification scheme and our evaluation. Afterwards, we mention several topics for future research in the context of resource demand estimation, which we identified with the findings of our evaluation.

In the literature research, we did not specifically look at estimation approaches for hard disks or network delays. These were out of scope of this thesis. It might be worthwhile to include such estimation approaches in our classification scheme as the evaluation with the TPC-W benchmark showed the following:

- Database servers are often causing a significant amount of I/O activity due to frequent hard disk accesses.
- The network connection between the web server and the database server caused non-negligible delays. We estimated these delays with an ad-hoc method. More sophisticated estimation approaches are required here.

We see the following possibilities to extend the evaluation of the estimation approaches:

- In Section 7.2, we showed that the Menascé optimization approach fails in environments with multiple resources and multiple workload classes if no resource demands are known beforehand. The inclusion of utilization measurements might solve this issue. Therefore, it might be of interest to implement and evaluate an optimization-based estimation approach that relies on the average response time as well as utilization measurements of each resource. For instance, such an estimation approach is proposed by Liu et al. [LXMZ03, LWXZ06].
- All resource demands were stationary over time in our evaluation. Additional scenarios might assess the adaptation speed and estimation accuracy of different estimation approaches in case of time-varying resource demands.
- The use of a highly complex application, such as the SPECjEnterprise2010 application benchmark, might provide valuable experiences for resource demand estimation in productive systems.

The evaluation pointed out insufficiencies of existing estimation approaches in combination with current technologies. These are multi-core processors on the one hand and virtualization technologies on the other hand. Existing estimation approaches need to be adapted to appropriately reflect the performance characteristics of multi-core processors. Especially, this is the case with estimation approaches relying on response time measurements. The influence of virtualization platforms on resource demand estimation should be analyzed in detail and it might be necessary to develop estimation approaches that specifically take the characteristics of the virtualization platform into account. Future research needs to address these issues.

As shown in the evaluation, the accuracy of estimation approaches depends on the characteristics of the workload and the resources in a concrete application scenario. None of the existing estimation approaches provides accurate resource demand estimates in all situations. We believe that a systematic methodology to tailor approaches to resource demand estimation for specific environments would help performance engineers to build accurate and representative performance models. Future work can build upon our classification scheme and our comparative evaluation in order to develop such a methodology.

Bibliography

[Bau93] 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.

[Bay] "Bayes++ Bayesian Filtering." [Online]. Available: http://bayesclasses.sourceforge.net/Bayes++.html

[BDIM04] P. Barham, A. Donnelly, R. Isaacs, and R. Mortier, "Using magpie for request extraction and workload modelling," in *Proceedings of the 6th conference on Symposium on Opearting Systems Design & Implementation - Volume 6*. Berkeley, CA, USA: USENIX Association, 2004, pp. 259–272.

[BGN00] R. H. Byrd, J. C. Gilbert, and J. Nocedal, "A trust region method based on interior point techniques for nonlinear programming," *Mathematical Programming*, vol. 89, pp. 149–185, 2000.

[BKK09] F. Brosig, S. Kounev, and K. Krogmann, "Automated extraction of palladio component models from running enterprise Java applications," in *VALUETOOLS '09: Proceedings of the Fourth International ICST Conference on Performance Evaluation Methodologies and Tools*. ICST, Brussels, Belgium, Belgium: ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering), 2009, pp. 1–10.

[BS78] Y. Bard and M. Shatzoff, "Statistical Methods in Computer Performance Analysis," *Current Trends in Programming Methodology*, vol. III, 1978.

[CCT07] G. Casale, P. Cremonesi, and R. Turrin, "How to Select Significant Workloads in Performance Models," in *2007 Computer Measurement Group Conference Proceedings*, 2007.

[CCT08] G. Casale, P. Cremonesi, and R. Turrin, "Robust Workload Estimation in Queueing Network Performance Models," in *Parallel, Distributed and Network-Based Processing, 2008. PDP 2008. 16th Euromicro Conference on*, Feb. 2008, pp. 183–187.

[CDS10] P. Cremonesi, K. Dhyani, and A. Sansottera, "Service Time Estimation with a Refinement Enhanced Hybrid Clustering Algorithm," in *Analytical and Stochastic Modeling Techniques and Applications*, ser. Lecture Notes in Computer Science, K. Al-Begain, D. Fiems, and W. Knottenbelt, Eds. Springer Berlin / Heidelberg, 2010, vol. 6148, pp. 291–305.

[CG05] L. Cherkasova and R. Gardner, "Measuring CPU overhead for I/O processing in the Xen virtual machine monitor," in *Proceedings of the annual conference on USENIX Annual Technical Conference*, ser. ATEC '05. Berkeley, CA, USA: USENIX Association, 2005, pp. 24–24.

[CP95] S. Chatterjee and B. Price, *Praxis der Regressionsanalyse*. Oldenbourg, 1995.

[Dos09] Z. Dostl, *Optimal Quadratic Programming Algorithms: With Applications to Variational Inequalities.* Springer Publishing Company, Incorporated, 2009.

[Fas] "FastICA." [Online]. Available: http://research.ics.tkk.fi/ica/fastica/

[GBL+11] H. Ghanbari, C. Barna, M. Litoiu, M. Woodside, T. Zheng, J. Wong, and G. Iszlai, "Tracking adaptive performance models using dynamic clustering of user classes," in *Proceeding of the second joint WOSP/SIPEW international conference on Performance engineering*, ser. ICPE '11. New York, NY, USA: ACM, 2011, pp. 179–188.

[HKHR11] M. Hauck, M. Kuperberg, N. Huber, and R. Reussner, "Ginpex: Deriving Performance-relevant Infrastructure Properties Through Goal-oriented Experiments," in *7th ACM SIGSOFT International Conference on the Quality of Software Architectures (QoSA 2011)*, Boulder, Colorado, USA, Jun. 2011.

[HO00] A. Hyvärinen and E. Oja, "Independent component analysis: algorithms and applications," *Neural Networks*, vol. 13, no. 4-5, pp. 411–430, 2000.

[IDC10] IDC, "Virtualization Market Accelerates Out of the Recession as Users Adopt "Virtualize First" Mentality," According to IDC, Apr. 2010.

[Kap07] T. Kappler, "Code-Analysis Using Eclipse to Support Performance Prediction for Java Components," 2007.

[KB06] S. Kounev and A. Buchmann, "SimQPN: a tool and methodology for analyzing queueing Petri net models by means of simulation," *Perform. Eval.*, vol. 63, pp. 364–394, May 2006.

[KCH+90] K. Kang, S. Cohen, J. Hess, W. Novak, and A. Peterson, "Feature-oriented domain analysis (FODA) feasibility study," Carnegie Mellon University, Tech. Rep. CMU/SEI-90-TR-021, Nov. 1990.

[Kou05] S. Kounev, *Performance Engineering of Distributed Component-Based Systems - Benchmarking, Modeling and Performance Prediction.* Shaker Verlag, Ph.D. Thesis, Technische Universität Darmstadt, Germany, Dec. 2005, best Dissertation Award from the "Vereinigung von Freunden der Technischen Universität zu Darmstadt e.V.".

[KPSCD09] S. Kraft, S. Pacheco-Sanchez, G. Casale, and S. Dawson, "Estimating service resource consumption from response time measurements," in *VALUETOOLS '09: Proceedings of the Fourth International ICST Conference on Performance Evaluation Methodologies and Tools.* ICST, Brussels, Belgium, Belgium: ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering), 2009, pp. 1–10.

[KSM10] S. Kounev, S. Spinner, and P. Meier, "From active data management to event-based systems and more," K. Sachs, I. Petrov, and P. Guerrero, Eds. Berlin, Heidelberg: Springer-Verlag, 2010, ch. QPME 2.0: a tool for stochastic modeling and analysis using queueing Petri nets, pp. 293–311.

[KTZ09] D. Kumar, A. Tantawi, and L. Zhang, "Real-time performance modeling for adaptive software systems," in *VALUETOOLS '09: Proceedings of the Fourth International ICST Conference on Performance Evaluation Methodologies and Tools.* ICST, Brussels, Belgium, Belgium: ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering), 2009, pp. 1–10.

[KZ06] T. Kelly and A. Zhang, "Predicting performance in distributed enterprise applications," HP Labs Tech Report, Tech. Rep., 2006.

[KZT09] D. Kumar, L. Zhang, and A. Tantawi, "Enhanced inferencing: estimation of a workload dependent performance model," in *VALUETOOLS '09: Proceedings of the Fourth International ICST Conference on Performance Evaluation Methodologies and Tools*. ICST, Brussels, Belgium, Belgium: ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering), 2009, pp. 1–10.

[LWXZ06] Z. Liu, L. Wynter, C. H. Xia, and F. Zhang, "Parameter inference of queueing models for IT systems using end-to-end measurements," *Performance Evaluation*, vol. 63, no. 1, pp. 36–60, 2006.

[LXMZ03] Z. Liu, C. H. Xia, P. Momcilovic, and L. Zhang, "AMBIENCE: Automatic Model Building using IferENCE," IBM Research, Tech. Rep., 2003.

[LZGS84] 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.

[LZJ+11] L. Lu, H. Zhang, G. Jiang, H. Chen, K. Yoshihira, and E. Smirni, "Untangling mixed information to calibrate resource utilization in virtual machines," in *Proceedings of the 8th ACM international conference on Autonomic computing*, ser. ICAC '11. New York, NY, USA: ACM, 2011, pp. 151–160.

[MA98] D. Menascé and V. Almeida, *Capacity planning for Web performance: metrics, models, and methods*. Prentice Hall, 1998.

[MAD97] D. Menascé, V. Almeida, and L. Dowdy, *Capacity planning and performance modeling: from mainframes to client-server systems*. Prentice Hall, 1997.

[Mat] "Fmincon documentation," accessed on May 11, 2011. [Online]. Available: http://www.mathworks.com/help/toolbox/optim/ug/fmincon.html

[MDA04] 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.

[Men08] D. Menascé, "Computing missing service demand parameters for performance models," in *CMG 2008*, 2008, pp. 241–248.

[MG00] D. A. Menascé and H. Gomaa, "A Method for Design and Performance Modeling of Client/Server Systems," *IEEE Trans. Softw. Eng.*, vol. 26, no. 11, pp. 1066–1085, 2000.

[MJ99] A. Martin and T. Jin, "Workload Characterization of the 1998 World Cup Web Site," Hewlett-Packard Laboratories, Tech. Rep., Sep. 1999.

[Nem89] G. L. H. Nemhauser, Ed., *Optimization*, ser. Handbooks in operations research and management science ; 1. Amsterdam [u.a.]: North-Holland, 1989.

[NKJT09] R. Nou, S. Kounev, F. Julií, and J. Torres, "Autonomic QoS control in enterprise Grid environments using online simulation," *J. Syst. Softw.*, vol. 82, pp. 486–502, Mar. 2009.

[Nor09] T. Norton, "Modeling Virtualized Environments in Simalytic Models by Computing Missing Service Demand Parameters," in *2009 Computer Measurement Group Conference Proceedings*, 2009.

[PSST08] G. Pacifici, W. Segmuller, M. Spreitzer, and A. Tantawi, "CPU demand for web serving: Measurement analysis and dynamic estimation," *Performance Evaluation*, vol. 65, no. 6-7, pp. 531–553, 2008, innovative Performance Evaluation Methodologies and Tools: Selected Papers from ValueTools 2006.

[RKCD10] J. Rolia, D. Krishnamurthy, G. Casale, and S. Dawson, "BAP: a benchmark-driven algebraic method for the performance engineering of customized services," in *Proceedings of the first joint WOSP/SIPEW international conference on Performance engineering*, ser. WOSP/SIPEW '10. New York, NY, USA: ACM, 2010, pp. 3–14.

[RKKD10] J. Rolia, A. Kalbasi, D. Krishnamurthy, and S. Dawson, "Resource demand modeling for multi-tier services," in *WOSP/SIPEW '10: Proceedings of the first joint WOSP/SIPEW international conference on Performance engineering.* New York, NY, USA: ACM, 2010, pp. 207–216.

[RV95] J. Rolia and V. Vetland, "Parameter estimation for performance models of distributed application systems," in *CASCON '95: Proceedings of the 1995 conference of the Centre for Advanced Studies on Collaborative research.* IBM Press, 1995, p. 54.

[RV98] 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*, ser. WOSP '98. New York, NY, USA: ACM, 1998, pp. 219–230.

[SBC+08] 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.

[Sch08] M. Schwende, "Modelling the Performance of Hard Disks and RAID Arrays for the Palladio Component Model," Master's thesis, KIT (Karlsruhe Institute of Technology), 2008.

[Sim06] D. Simon, *Optimal state estimation : Kalman, H. [infinity] and nonlinear approaches.* Hoboken, NJ: Wiley-Interscience, 2006.

[SKZ07] C. Stewart, T. Kelly, and A. Zhang, "Exploiting nonstationarity for performance prediction," *SIGOPS Oper. Syst. Rev.*, vol. 41, pp. 31–44, Mar. 2007.

[Smi90] C. U. Smith, *Performance Engineering of Software Systems*, 1st ed. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 1990.

[Smi02] C. U. Smith, *Encyclopedia of Software Engineering*, 2nd ed. Boston, MA, USA: John Wiley & Sons, Jan. 2002, ch. Software Performance Engineering, pp. 1545–1562.

[SW02] C. U. Smith and L. G. Williams, *Performance solutions : a practical guide to creating responsive, scalable software*, 1st ed., ser. The Addison-Wesley object technology series. Boston, Mass.: Addison-Wesley, 2002.

[TPC02] "TPC Benchmark W (Web Commerce)," Transaction Processing Performance Council (TPC), Standard Version 1.8, Feb. 2002.

[UPS+07] B. Urgaonkar, G. Pacifici, P. Shenoy, M. Spreitzer, and A. Tantawi, "Analytic modeling of multitier Internet applications," *ACM Trans. Web*, vol. 1, May 2007.

[WC08] J. Williams and L. Curtis, "Green: The New Computing Coat of Arms?" *IT Professional*, vol. 10, pp. 12–16, Jan. 2008.

[WXZ04] L. Wynter, C. H. Xia, and F. Zhang, "Parameter inference of queueing models for IT systems using end-to-end measurements," in *Proceedings of the joint international conference on Measurement and modeling of computer systems*, ser. SIGMETRICS '04/Performance '04. New York, NY, USA: ACM, 2004, pp. 408–409.

[WZL05] M. Woodside, T. Zheng, and M. Litoiu, "The Use of Optimal Filters to Track Parameters of Performance Models," in *Proceedings of the Second International Conference on the Quantitative Evaluation of Systems*. Washington, DC, USA: IEEE Computer Society, 2005, p. 74ff.

[ZCS07] Q. Zhang, L. Cherkasova, and E. Smirni, "A Regression-Based Analytic Model for Dynamic Resource Provisioning of Multi-Tier Applications," in *Proceedings of the Fourth International Conference on Autonomic Computing*. Washington, DC, USA: IEEE Computer Society, 2007, p. 27ff.

[ZWL08] T. Zheng, C. Woodside, and M. Litoiu, "Performance Model Estimation and Tracking Using Optimal Filters," *Software Engineering, IEEE Transactions on*, vol. 34, no. 3, pp. 391–406, May 2008.

[ZXSI02] L. Zhang, C. H. Xia, M. S. Squillante, and W. N. M. Iii, "Workload Service Requirements Analysis: A Queueing Network Optimization Approach," in *Proceedings of the 10th IEEE International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunications Systems*, ser. MASCOTS '02. Washington, DC, USA: IEEE Computer Society, 2002, p. 23ff.

[ZYW+05] 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: Proceedings of the 2005 conference of the Centre for Advanced Studies on Collaborative research*. IBM Press, 2005, pp. 334–345.

Acronyms

API Application Programming Interface.

BLUE Best Linear Unbiased Estimator.

DVFS Dynamic Voltage and Frequency Scaling.

EB Emulated Browser.

EKF Extended Kalman filter.

EMF Eclipse Modeling Framework.

FCFS First-Come-First-Serve.

ICA Independent Component Analysis.

IS Infinite-Server.

Java EE Java Enterprise Edition.

JDBC Java Database Connectivity.

JDK Java Development Kit.

JMX Java Management Extensions.

JNA Java Native Access.

JNI Java Native Interface.

LAD Least Absolute Differences.

LQN Layered Queueing Network.

LSQ Least Squares.

LTS Least Trimmed Squares.

MLE Maximum Likelihood Estimation.

MVA Mean Value Analysis.

NTP Network Time Protocol.

PS Processor-Sharing.

QN Queueing Network.

QP Quadratic Programming.

QPN Queueing Petri Network.

SNMP Simple Network Management Protocol.

SPE Software Performance Engineering.

SUT System under Test.

URL Uniform Resource Locator.

VM Virtual Machine.

VMM Virtual Machine Monitor.

WMI Windows Management Instrumentation.

Appendix

A. Conferences

Conference or Workshop
IEEE International Symposium on Workload Characterization
ACM SIGMETRICS
IFIP International Symposium on Computer Performance, Modeling, Measurements and Evaluation (PERFORMANCE)
ACM Workshop on Hot Topics in Measurement & Modeling of Computer Systems (HotMetrics)
ACM/IFIP/USENIX International Middleware Conference (Middleware)
WOSP/SIPEW International Conference on Performance Engineering (ICPE)
ACM International Workshop on Software and Performance (WOSP)
SPEC International Performance Evaluation Workshop (SIPEW)
IEEE/ACM International Symposium on Modelling, Analysis and Simulation of Computer and Telecommunication Systems (MASCOTS)
International ICST Conference on Performance Evaluation Methodologies and Tools (VALUETOOLS)
European Performance Engineering Workshop (EPEW)
International Conference on Quantitative Evaluation of SysTems (QEST)

Table A.1.: Conferences and workshops included in the literature research.