Interval Parameters for Capturing Uncertainties in an EJB Performance Model

JOHANNES LÜTHI

CATALINA M. LLADÓ

Bericht Nr. 2000-08

Dezember 2000

Universität der Bundeswehr München

Fakultät für

INFORMATIK

Werner-Heisenberg-Weg 39 • 85577 Neubiberg • Germany

Interval Parameters for Capturing Uncertainties in an EJB Performance Model

Johannes Lüthi[1] and Catalina M. Lladó[2]

Technical Report No. 2000-08

[1]Institut für Technische Informatik
Universität der Bundeswehr München
Werner-Heisenberg-Weg 39, 85577 Neubiberg, Germany
email: `luethi@informatik.unibw-muenchen.de`
URL: `www.informatik.unibw-muenchen.de/inst4/luethi`

[2]Department of Computing
Imperial College of Science, Technology and Medicine
180 Queens Gate, London SW7 2BZ, United Kingdom
email: `cl16@doc.ic.ac.uk`

Abstract

Exact as well as approximate analytical solutions for quantitative performance models of computer systems are usually obtained by performing a series of arithmetical operations on the input parameters of the model. However, especially during early phases of system design and implementation, not all the parameter values are usually known exactly. In related research contributions, intervals have been proposed as a means to capture parameter uncertainties. Furthermore, methods to adapt existing solution algorithms to parameter intervals have been discussed. In this paper we present the adaptation of an existing performance model to parameter intervals. The approximate solution of a queueing network modelling an Enterprise JavaBeans server implementation is adapted to interval arithmetic in order to represent the uncertainty in some of the parameters of the model. A new interval splitting method is applied to obtain reasonable tight performance measure intervals. Monotonicity properties of intermediate computation results are exploited to achieve a more efficient interval solution. In addition, parts of the original solution algorithm are modified to increase the efficiency of the corresponding interval arithmetical solution.

Keywords: distributed systems, performance modelling, queueing, enterprise JavaBeans, parameter uncertainties, interval parameters

Contents

1 Introduction

Building a performance model typically involves two different types of abstraction: firstly, the structural properties of a real system (existing or planned) are modelled. The result of this structural abstraction may for example be a queueing network model or a Petri net. Secondly, quantitative behaviour, such as e.g. the arrival process of customers or routing behaviour, of components of the real system has to be characterised. The result of this abstraction step is usually a set of model parameters. However, often not every aspect of the real system is known exactly when the model is developed. Especially in early phases of design and implementation, uncertainties may exist. This is true for both, structural as well as parametrical model aspects. This work deals with uncertainties associated with model parameters.

The use of intervals to characterise parameter uncertainties in performance models has first been proposed by Majumdar [Maju 91]. There are many situations where parameter intervals occur naturally: although an exact value for a parameter may not be known, the designer may provide a reasonable range of values for that parameter. If parameters are obtained via measurement, confidence intervals are an important tool to increase the reliability of the results. Parameter intervals may also occur in a situation where bounding analysis is used at one level of a hierarchical model producing input parameter intervals on another level. Parameter intervals are also suitable for worst-case analysis as well as sensitivity studies. Furthermore, the mathematical treatment of other approaches to model parameter uncertainties such as e.g. parameter histograms is based on intervals [Luth 98a].

When parameters of an analytical model are characterised by intervals, performance measure intervals can be obtained by adapting existing solution algorithms and formulae for the corresponding model characterised by single value (SV) parameters. This is done by replacing conventional arithmetic by so-called interval arithmetic. I.e., basic operations and elementary functions for real numbers are replaced by corresponding arithmetic defined for intervals. However, the so-called *dependency problem* may cause extremely wide intervals for the computed performance measures [Neum 90]. Interval splitting as an approach to overcome this problem is proposed by Majumdar and Ramadoss [Maju 95]. Improved interval splitting methods applied in this work are proposed in [Luth 00]. In [Luth 98b], Lüthi and Haring use monotonicity properties to obtain an efficient interval solution for the well-known mean value analysis (MVA) algorithm for closed single class queueing network models. There are two major advantages of using interval arithmetic as opposed to traditional techniques for uncertainty analysis like Monte-Carlo [Rubi 81] and Quasi-Monte-Carlo [Nied 78] methods or sensitivity analysis (see for example [Have 95] for a comparison of these two approaches in the context of Markov reward models): (a) results produced by interval analysis are safe performance bounds, i.e., it is guaranteed that the possible range of performance measures is always enclosed by the obtained interval results; (b) if interval splitting is applied, the accuracy of the obtained interval results is automatically known to the analyst.

So far, research on using intervals as parameters for performance models does not include application of proposed methods to *real* models of *real* systems. In this paper we report experiences made by adapting an existing analytical performance model to interval parameters. In a software performance model of an *Enterprise JavaBeans* (EJB) server implementation presented by Lladó and Harrison [Llad 00], the timing parameters are not known exactly. Due to restricted access to the real system, accurate measurements to obtain the service rate parameters for various components of the system cannot be performed. In order to capture this type of parameter uncertainty the timing parameters of the model are replaced by intervals laid around parameter estimates obtained via expert guess. An approximate mathematical solution of the model is adapted to handle these interval parameters. For that purpose, the solution algorithm is transformed into interval arithmetic. Moreover, interval splitting is used to obtain sufficiently tight performance measure intervals. This way, the uncertainty in model parameters is also reflected in the performance results. Additionally, sensitivity analysis of the system under study is supported by the interval version of the model. This type of analysis is of special importance in the presence of uncertain parameters. Results of these studies will be reported in future work. For the adaptation to interval parameters, some of the original expressions are rewritten such that the effect of the dependency problem is reduced. Furthermore, monotonicity of intermediate results is exploited. With these optimisations, the efficiency of the interval splitting algorithm can be significantly improved.

The rest of the paper is organised as follows: in the next section, some mathematical background about interval parameters is discussed in more detail. Section 3 presents the software performance model to be adapted to interval parameters. Techniques to obtain a more efficient interval solution by rewriting the original expressions and by exploiting monotonicity of intermediate results are considered in Section 4. Section 5 demonstrates the effect of these optimisations along the lines of some experiments. In Section 6, the results are summarised and possibilities for future work are discussed.

2 Interval Parameters

2.1 Definitions and Introduction

A real interval is a set of the form

$$X = [\underline{x}, \overline{x}] = \{x \in I\!R \mid \underline{x} \leq x \leq \overline{x}\},$$

where $\underline{x}, \overline{x} \in I\!R$ and $\underline{x} \leq \overline{x}$. $\underline{x}$ and $\overline{x}$ are called *endpoints* of the interval. In particular, $\underline{x}$ is called *lower bound*, and $\overline{x}$ is called *upper bound* of the interval $X = [\underline{x}, \overline{x}]$. If S is a nonempty bounded subset of $I\!R$, we denote the *hull* of S by $\Box S = [\inf(S), \sup(S)]$. The hull is the tightest interval enclosing S. An interval vector $X = (X_1, \ldots, X_n)$ is also referred to as a *box*. For a real function f, continuous on every closed box on which it is defined, the *range* of a box X is defined as:

$$f^*(X) = \Box\{f(x) \mid x \in X\} = \{f(x) \mid x \in X\}.$$

Given a performance model with interval parameters we are usually interested to find the range of associated performance measures. Because of the continuity of f, the range is itself an interval: $f^*(X) = [\underline{f}, \overline{f}]$. In general, the computation of the range is a global optimisation problem with box constraints. I.e., the global minimum $\underline{f}(x) = \min_{x \in X} f(x)$ and the global maximum $\overline{f}(x) = \max_{x \in X} f(x)$, subject to $x \in X$ have to be found.

In the special case of so-called N-monotonic functions, the range can be computed using only single value evaluations of f with appropriate combinations of parameter interval endpoints as input parameters. To be more specific, let $f(x_1, \ldots, x_n)$ be monotonically increasing w.r.t. all parameters x_i, $i \in I$ and monotonically decreasing w.r.t. all parameters x_i, $i \in D$, where $I \cup D = \{1, \ldots, n\}$. Then the range of f with interval parameters $X_1 = [\underline{x}_1, \overline{x}_1], \ldots, X_n = [\underline{x}_n, \overline{x}_n]$ can be computed as follows:

$$f^*(X_1, \ldots, X_n) \quad = \quad [f(y_1, \ldots, y_n),\, f(z_1, \ldots, z_n)]\,,$$

where $y_i = \underline{x}_i$, $z_i = \overline{x}_i$ if $i \in I$, and $y_i = \overline{x}_i$, $z_i = \underline{x}_i$ if $i \in D$. In [Luth 98b], such a situation is discussed in detail for the example of the *Mean Value Analysis* (MVA) algorithm for closed single class queueing networks.

2.2 Interval Arithmetic

For many performance measures, monotonicity properties do not hold and general optimisation methods are often difficult to apply and of high computational complexity. However, an enclosure $F(X) \supseteq \{f(x)|x \in X\}$ for the range can be obtained using so-called *interval arithmetic* (for a detailed introduction see for example the book [Neum 90]): On the set of intervals, the *elementary operations* $\circ \in \{+, -, \cdot, /, \hat{}\} =: \Omega$ are defined by setting:

$$X \circ Y = \square\{x \circ y \,|\, x \in X, y \in Y\} = \{x \circ y \,|\, x \in X, y \in Y\}, \quad \forall \circ \in \Omega.$$

Furthermore, the elements φ of a predefined set Φ of elementary continuous real functions are extended to interval arguments by defining:

$$\varphi(X) = \square\{\varphi(x) \,|\, x \in X\} = \{\varphi(x) \,|\, x \in X\},$$

for all intervals X such that $\varphi(x)$ is defined for all $x \in X$.

From monotonicity properties it follows that the elementary operations $\circ \in \{+, -, \cdot, /\}$ can be computed in terms of the endpoints of the intervals $X = [\underline{x}, \overline{x}], Y = [\underline{y}, \overline{y}]$: $X + Y = [\underline{x} + \underline{y}, \overline{x} + \overline{y}]$, $X - Y = [\underline{x} - \overline{y}, \overline{x} - \underline{y}]$, $X \cdot Y = [\min(\underline{x}\underline{y}, \underline{x}\overline{y}, \overline{x}\underline{y}, \overline{x}\overline{y}), \max(\underline{x}\underline{y}, \underline{x}\overline{y}, \overline{x}\underline{y}, \overline{x}\overline{y})]$, and $X/Y = X \cdot [1/\overline{y}, 1/\underline{y}]$, if $0 \notin Y$. Analogously, (piecewise) monotonicity of the elementary functions can be exploited to define their evaluations along the lines of computations with the interval endpoints of the argument. E.g., because of the monotonicity of the exponentiation function we know that for any interval $X = [\underline{x}, \overline{x}]$, $\exp(X) = [\exp(\underline{x}), \exp(\overline{x})]$. Using the interval extensions of elementary operations and functions, an arithmetic expression can be evaluated with intervals by substituting the variables by the corresponding intervals and step by step application of interval arithmetic.

Interval arithmetic can serve as a tool to obtain interval extensions of real functions. However, due to an effect known as *dependency problem*, in general, interval arithmetic does not provide the exact range of a function. This effect is also known as *overestimation*. The root of the dependency problem is the memoryless nature of interval arithmetic if a parameter occurs multiple times in an arithmetic expression since each occurrence of an interval variable in an expression is treated independently [Neum 90]. For example, the expression $X - X$ is evaluated to $\{x_1 - x_2 \,|\, x_1, x_2 \in X\} = [\underline{x} - \overline{x}, \overline{x} - \underline{x}]$, instead of $\{x - x \,|\, x \in X\} = [0, 0]$. Sometimes an expression can be reformulated to avoid multiple occurrence of parameters or at least to reduce the number of occurrences of an interval parameter. The application of this technique is demonstrated in Section 4. However, in general multiple occurrence of interval parameters cannot always be avoided. Therefore the dependency problem often causes crucial overestimation of the actual range of an evaluated function.

2.3 Interval Splitting

A way to overcome overestimation due to the dependency problem is to split the original input parameter intervals into subintervals and evaluate the arithmetic expression using these subintervals as input parameters. The principal idea for interval splitting is to subdivide the input parameter intervals into a number of subintervals, compute interval evaluations of the arithmetic expression with the subintervals as input parameters, and find the overall result by computing the minimum of all lower bounds and the maximum of all upper bounds of the intermediate results. Analogously, an interval parameter vector (box) is split into subboxes. In [Skel 74] it is shown that the results obtained from interval splitting converge to the actual range if the width of the subintervals approaches zero. This means that it is guaranteed that interval splitting is indeed a technique to obtain sufficiently tight interval results.

In the *brute force splitting* (BFS) algorithm, in every iteration the input parameter intervals are split into two subintervals of equal length. The parameter (sub)intervals considered in iteration s (i.e. splitting degree s) are collected in P^s, the set of potential input parameter intervals. In every iteration, the splitting degree s is incremented and a new set P^s of input parameter intervals to be considered is initialised. Subsequently, P^s is filled with subintervals of all intervals $X \in P^{s-1}$. Finally, the minimum of all lower bounds and the maximum of all upper bounds of evaluations of these subintervals is computed. These steps are iterated until the difference between successive iterations becomes smaller than a predefined stopping criterion ϵ.

Note that the number of subintervals at splitting degree s is 2^s. More general, if n parameters are characterised by intervals (i.e. we have an n-dimensional input parameter box), it holds that $|P^s| = 2^{sn}$. The application of the BFS algorithm for the solution of interval-based computer performance models is presented in [Maju 95].

The high complexity of BFS can be significantly reduced if not every subinterval is considered for further splitting. Given a subinterval X it may eventually be concluded from the obtained interval results that neither the lower nor the upper bound of the range is produced by that subinterval. In that case, X need not be considered any further in the search for the lower and upper bound of the range; i.e., X need not be split into additional subintervals. This idea of selective interval splitting was introduced by Skelboe in the context of general purpose optimisation of rational interval functions [Skel 74]. For the experiments presented in Section 5 we use a modified selective splitting algorithm described in [Luth 00]. This algorithm combines interval and conventional evaluations to reduce the number of subintervals that have to be considered.

3 Model of an EJB Server Implementation

The model to be adapted to interval parameters represents an EJB (*Enterprise JavaBeans*) server implementation, as a central scheduler of a distributed, three-tier, client-server architecture. This type of architecture is typical for large, Java-supported, Internet applications. The Kensington Enterprise Data Mining system [Ltd], [Chat 99] is the real application under study, whose application server (or scheduler) implements the EJB-1.1 specification [Micr 99].

EJB is a new component architecture, created by Sun, for the development and deployment of object-oriented, distributed, enterprise-level applications. A bean instance (or component) is created and managed at runtime by a container, so that a task can only access a bean instance through its container. The number of active (bean) instances for each container is limited due to memory and performance constraints.

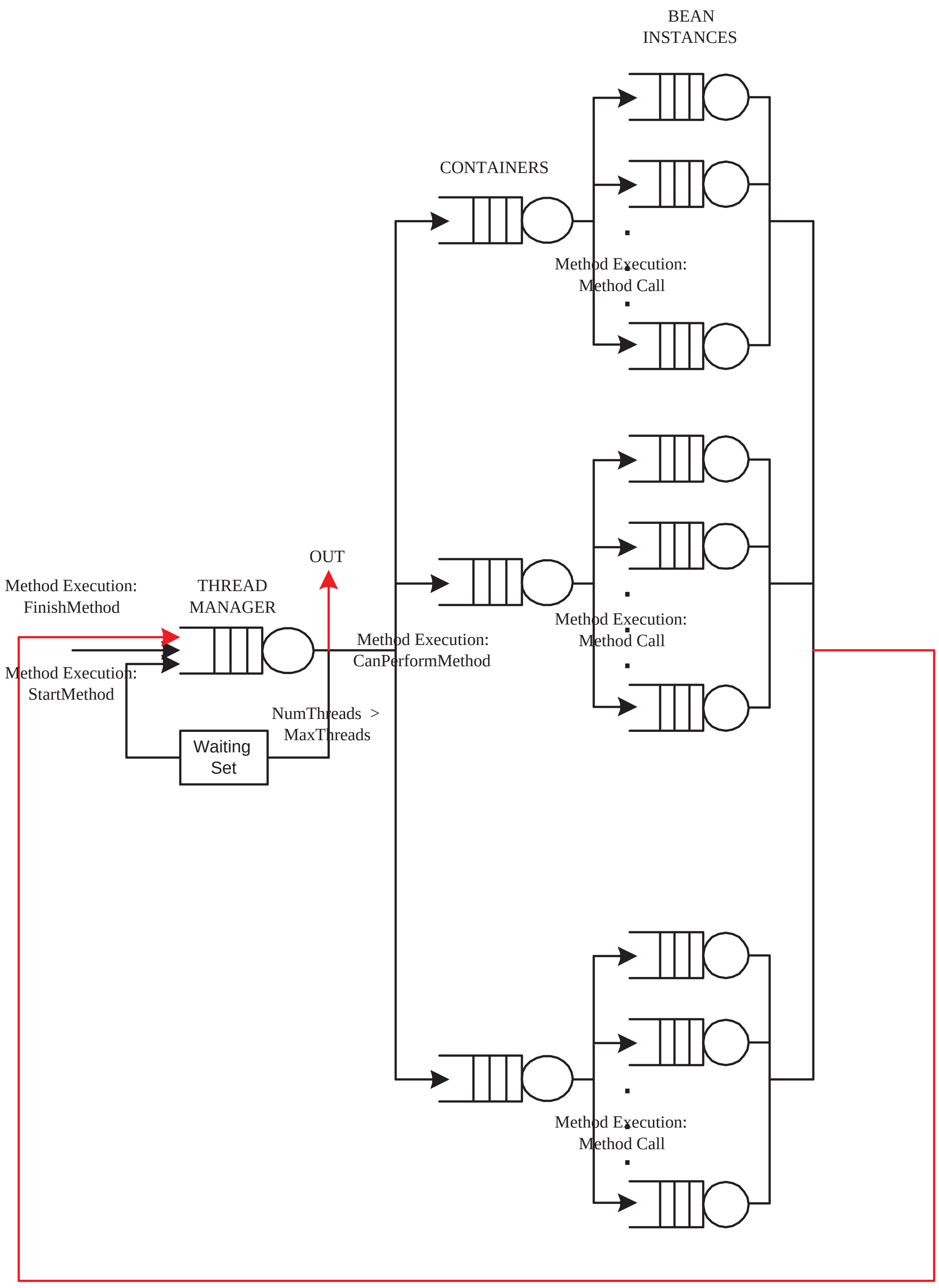

BEAN INSTANCES
CONTAINERS
Method Execution: Method Call
Method Execution: Method Call
Method Execution: Method Call
OUT
Method Execution: FinishMethod
THREAD MANAGER
Method Execution: CanPerformMethod
Method Execution: StartMethod
NumThreads > MaxThreads
Waiting Set

Figure 1: Global queueing network for a method execution

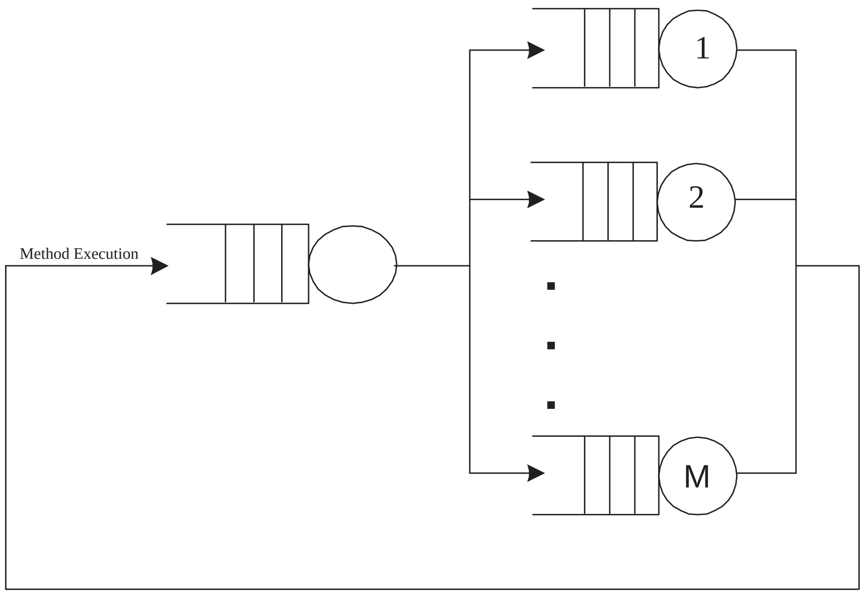


Figure 2: Short-circuited FES sub-network

3.1 Queueing Model

Since a method execution is the most common operation over a bean instance, its specific behaviour is modelled. This consists of access to the container server followed by access to the instance server required. Blocking can arise since the execution of a method in a bean instance requires this instance to be *active*. A more detailed description of this execution can be found in [Llad 00].

The queueing network model shown in Fig. 1 summarises the method execution behaviour. The queueing network consists of $1 + C + C * M$ stations, where 1 corresponds to the thread manager station, C is the number of containers in the system (i.e. the number of different bean classes) and M is the maximum number of (different) bean instances for a bean class that can be active at the same time.

Every node in the model has first-come-first-served (FCFS) queueing discipline. For mathematical tractability and the desire for an efficient (approximate) solution, all service times are assumed to be exponential random variables and routing probabilities are assumed to be constant and equal across each of the C bean containers. The M bean instances attached to each bean container are also equally utilised overall, but the specific routing probabilities in each network-state depend on the blocking properties, which are described below.

To simplify this system, the Flow Equivalent Server method (FES) is applied (see [Harr 93], for example). The FES method reduces the number of nodes by aggregating sub-networks into single, more complex (i.e. queue length dependent) nodes. Applying this method to our system, each FES sub-network consists of $M + 1$ stations where 1 corresponds to the container for a bean class and M is as above. After short-circuiting, this sub-network results

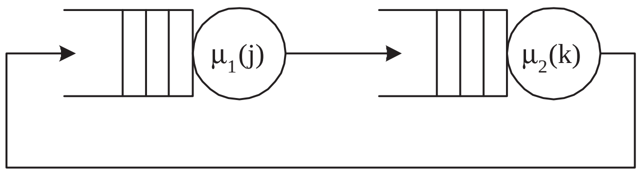


Figure 3: Container (sub)Model

in the closed one shown in Fig. 2, which is analysed to obtain its throughput for different populations N. The throughput function will determine the service rate function for an FES node in the overall network.

Blocking is the critical non-standard characteristic in this network; a client that has completed service in the container station is blocked if the required bean instance is not active and there is no free instance to passivate, i.e., no idle server in the model. Blocking time is the time required for the first of the M parallel servers to clear its queue in a blocking-after-service discipline.

3.2 The Aggregated Server and Container (sub)Model

The (sub)model of a subsystem comprising a bean container and its M instance servers (i.e. bean method execution servers) at constant population N is shown in Fig. 3, where the second server corresponds to the aggregated node for the M parallel servers.

When there are j clients at the outer (container) server and $k = N - j$ at the M parallel servers, the service rate functions $\mu_1(j)$ (with blocking) and $\mu_2(k)$, are estimated as follows:

$$\mu_1(j) = \begin{cases} 1/(m_1 + \beta_{N-j} b(N-j)), & \text{if } (N-j) \geq M, \\ 1/m_1, & \text{otherwise,} \end{cases} \tag{1}$$

where m_1 is the mean service time for server 1 (the outer server) when there is no blocking and β_{N-j} is the dynamic blocking probability, which is derived in the next subsection. $b(N-j)$ is the mean blocking time when there are j customers at the outer server ($N-j$ customers at the parallel servers) and it is estimated by $b(k) = k/(M^2\mu)$ (see [Harr 00]), where μ is the service rate of each of the parallel servers.

$$\mu_2(k) = \sum_{n=1}^{M} \xi_{kn} n \mu, \tag{2}$$

where the parameter ξ_{kn} is the probability that n out of the M servers are busy, given that there are k customers at the parallel servers altogether; it is also derived in the next subsection.

Clearly the visitation rate is the same for both servers. The steady state queue length probability distribution for this network – $p(j)$ for the state with j tasks at server 1 and

$N - j$ at server 2 – is then calculated as a product form in standard fashion in the following way:

$$p(j) = \frac{1}{G} \frac{1}{\prod_{k=1}^{j} \mu_1(k) \prod_{k=1}^{N-j} \mu_2(k)} , \tag{3}$$

where G is calculated as

$$G = \sum_{j=1}^{N} \frac{1}{\prod_{k=1}^{j} \mu_1(k) \prod_{k=1}^{N-j} \mu_2(k)} . \tag{4}$$

Throughput $T(N)$ at population N is then given by:

$$T(N) = \sum_{j=1}^{N} p(j)\mu_1(j) . \tag{5}$$

3.3 Instance-Active and Client-Blocking Probabilities

Let z_k denote the equilibrium probability that, at an instant when a task completes service at the container server (with rate $1/m_1$) with k tasks at the M instance servers, at least one of the instance servers is idle. Let α denote the probability that the instance required by a task completing service at the container server is active, i.e. that the task can immediately join that instance's queue (whether or not empty) and so is not blocked. Then the dynamic blocking probability for a task completing service at the outer (container) server when there are k tasks at the parallel servers is:

$$\beta_k = \frac{(1 - z_k)(1 - \alpha)\frac{m1}{m1+\beta_k b(k)}}{z_k + (1 - z_k)\frac{m1}{m1+\beta_k b(k)}}. \tag{6}$$

This quantity is the ratio of the equilibrium probability flux from unblocked states with k tasks at the parallel servers to a blocked state, divided by the total flux from unblocked states (see [Kell 79], for example). α is approximated by M/I, assuming the next arrival to the parallel servers requires each of the I instances (I is the total number of instances available to each client) with equal probability. From Eq. (6), β_k can be expressed as follows:

$$\beta_k = \frac{\sqrt{m_1^2 + 4b(k)z_k(1 - z_k)(1 - \alpha)m_1} - m_1}{2b(k)z_k}. \tag{7}$$

The parameter z_k is estimated by considering an M-state Markov chain Ξ_t^k for each population size $k \geq M$ at the parallel servers, where each state corresponds to the number of busy queues in the system.

For population size $k \geq M$ at the parallel servers, let the equilibrium probability that $\Xi \equiv \Xi_\infty^k = l$ $(l = 1, \ldots, M)$ be denoted by $\pi_k(l)$. The following recursive function is derived

$\boldsymbol{C}$	Containers or bean classes
$\boldsymbol{M}$	Parallel servers
$\boldsymbol{I}$	Different instances
$\boldsymbol{N}$	Population
$\boldsymbol{m_1}$	Mean service time for outer server
$\boldsymbol{\mu}$	Service rate for each of the parallel servers
$\boldsymbol{j}$	Clients at the outer server
$\boldsymbol{\mu_1(j)}$	Service rate for outer server with queue length j
$\boldsymbol{\mu_2(k)}$	Service rate for aggregated server with queue length k
$\boldsymbol{b(k)}$	Mean blocking time when there are k clients at the parallel servers
$\boldsymbol{\beta(k)}$	Blocking probability when there are k clients at the parallel servers
$\boldsymbol{\xi_{kn}}$	Probability that n out of M parallel servers are busy, when there are k clients at the parallel servers
$\boldsymbol{\alpha}$	Probability that the instance required by task completing service at the outer server is active
$\boldsymbol{z_k}$	Probability that when a task completes service at the outer server with k tasks at the parallel servers, at least one of these servers is idle
$\boldsymbol{p(j)}$	Steady state probability distribution of queue length at outer server

Table 1: Notation for the model parameters

from the balance equations determined by the M-state Markov chain.

$$\pi_k(n) = \begin{cases} 1, & \text{if } n = 1, \\ \frac{(I-n+1)(k-1)}{n(n-1)m_1\mu I}\pi_k(n-1), & \text{if } 2 \leq n < M, \\ \frac{(I-n+1)(k-1)(m_1 M^2\mu+(1-\alpha)k)}{M^2 m_1^2\mu^2 In(n-1)}\pi_k(n-1), & \text{if } n = M. \end{cases} \tag{8}$$

Normalising the $\pi_k(n)$ to give the probabilities

$$\xi_{kn} = \frac{\pi_k(n)}{\sum_{l=1}^{M}\pi_k(l)}\ , \tag{9}$$

we now estimate z_k by $1 - \xi_{kM}$ and β_k follows for $M \leq k < N$. The same applies to $\mu_2(k)$, but when there is no blocking, i.e. when $k \leq M$ or $k \leq I$, $\mu_2(k) = \frac{kI\mu}{k+I-1}$.

Table 1 summarises the notation used for the different parameters of the model.

4 Interval Adaptation of the Model

As discussed in Section 2, the general strategy to adapt an existing analytical model to interval parameters is to substitute single value parameters by intervals and perform step by step interval arithmetical evaluations of all intermediate expressions. In general, the dependency problem arising with multiple occurrence of input parameters in the involved calculations can be overcome by interval splitting techniques. However, the efficiency — and thus the practical applicability — of splitting techniques can be significantly improved if the interval adaptation is optimised in two respects: (a) wherever possible, monotonic behaviour of intermediate expressions should be exploited to avoid overestimation of intermediate intervals, and (b) wherever possible, expressions should be rewritten such that the number of occurrences of interval input parameters is reduced. In the following subsections we show the application of these techniques to the various computational steps of the model solution described in the previous section. Computations that are not subject to the dependency problem are not considered.

4.1 Computation of the Probabilities ξ_{kn}

For the computation of the blocking probabilities β_k and the service rates $\mu_2(k)$ we need the probabilities ξ_{kn} that n out of M parallel servers are busy, when there are k clients at the parallel servers ($M \le k \le N$ and $1 \le n \le M$). Furthermore, we also need the probability $z_k = 1 - \xi_{kM}$ that at least one of the parallel servers is idle when a task completes service at the outer server while there are k tasks at the parallel servers.

Due to the recursion in Eq. (8), m_1 and μ occur multiple times in each of the expressions $\pi_k(n)$. Since with the exception of the case $n = M$, m_1 as well as μ appear only in the denominator, this does not cause overestimation of intervals for $\pi_k(n)$. However, in the normalisation step, the dependency problem is in effect, because by having $\pi_k(n)$ in the numerator and the sum $\sum_{l=1}^{M} \pi_k(l)$ in the denominator, m_1 and μ have both increasing as well as decreasing influence on ξ_{kn}.

In the following we rewrite the expressions for $\pi_k(n)$ in a way that allows to cancel as many occurrences of m_1 and μ as possible in the normalisation. In a first step, we extract the interval parameters m_1 and μ from the recursion of Eq. (8). This can be done by defining the following recursive expression $\tau_k(n)$ that does not depend on m_1 and μ:

$$\tau_k(n) = \begin{cases} 1, & \text{if } n = 1, \\ \frac{(I-n+1)(k-1)}{n(n-1)I}\tau_k(n-1), & \text{if } 2 \le n \le M. \end{cases} \tag{10}$$

Using these $\tau_k(n)$, the $\pi_k(n)$ defined in Eq. (8) can be rewritten as:

$$\pi_k(n) = \begin{cases} \frac{\tau_k(n)}{(m_1\mu)^{n-1}}, & \text{if } 1 \le n < M, \\ \frac{\tau_k(M)(m_1 M^2\mu + (1-\alpha)k)}{M^2(m_1\mu)^M}, & \text{if } n = M. \end{cases} \tag{11}$$

To rewrite the normalisation step, we need the ratios $\pi_k(l)/\pi_k(n)$, $1 \leq l \leq M$. Because $l = M$ is a special case in the definition of $\pi_k(n)$, $1 \leq l < M$ and $l = M$ are treated separately:

$$\frac{\pi_k(l)}{\pi_k(n)} = \frac{\tau_k(l)}{\tau_k(n)}(m_1\mu)^{n-l}, \quad \text{if } l < M$$

and

$$\frac{\pi_k(M)}{\pi_k(n)} = \frac{\tau_k(M)(m_1 M^2 \mu + (1-\alpha)k)}{\tau_k(n)(m_1\mu)^{M-n} m_1 M^2 \mu} = \frac{\tau_k(M)}{\tau_k(n)}\left[\frac{1}{(m_1\mu)^{M-n}} + \frac{(1-\alpha)k}{M^2(m_1\mu)^{M-n+1}}\right].$$

Now the reciprocals of the probabilities ξ_{kn}, $1 \leq n < M$ can be rewritten as follows:

$$\begin{aligned}\xi_{kn}^{-1} &= \sum_{l=1}^{M}\frac{\pi_k(l)}{\pi_k(n)} = 1 + \sum_{l=1}^{n-1}\frac{\tau_k(l)(m_1\mu)^{n-l}}{\tau_k(n)} + \sum_{l=n+1}^{M-1}\frac{\tau_k(l)}{\tau_k(n)(m_1\mu)^{l-n}} \\ &\quad + \frac{\tau_k(M)}{\tau_k(n)}\left[\frac{1}{(m_1\mu)^{M-n}} + \frac{(1-\alpha)k}{M^2(m_1\mu)^{M-n+1}}\right] \\ &= 1 + \frac{1}{\tau_k(n)}\left[\sum_{l=1}^{n-1}\tau_k(l)(m_1\mu)^{n-l} + \sum_{l=n+1}^{M}\frac{\tau_k(l)}{(m_1\mu)^{l-n}} + \frac{\tau_k(M)(1-\alpha)k}{M^2(m_1\mu)^{M-n+1}}\right].\end{aligned} \tag{12}$$

Note that in this expression for ξ_{kn}^{-1}, $1 \leq n < M$, the computation is separated in a part where m_1 and μ contribute with an increasing effect and a part where m_1 and μ contribute with a decreasing effect, respectively. Within these parts, the parameters m_1 and μ are cancelled as often as possible. This significantly reduces the effect of the dependency problem as compared to the original expressions in Eqs. (8) and (9).

Again, the case $n = M$ is treated separately:

$$\begin{aligned}\xi_{kM}^{-1} &= \sum_{l=1}^{M}\frac{\pi_k(l)}{\pi_k(M)} = 1 + \frac{1}{\tau_k(M)}\sum_{l=1}^{M-1}\frac{\tau_k(l)(m_1\mu)^{M-l+1}}{M^2(m_1\mu) + (1-\alpha)k} \\ &= 1 + \frac{M^2}{\tau_k(M)\left[M^2 + (1-\alpha)k/(m_1\mu)\right]}\sum_{l=1}^{M-1}\tau_k(l)(m_1\mu)^{M-l}.\end{aligned} \tag{13}$$

Since in that expression it can be seen that m_1 and μ contribute with an increasing effect to ξ_{kM}^{-1}, the probability $z_k = 1 - \xi_{kM}$ is monotonically increasing w.r.t. the parameters m_1 and μ. Thus, an interval $Z_k = [\underline{z}_k, \overline{z}_k]$ can be obtained by single value evaluation of z_k using the endpoints of m_1 and μ's parameter intervals. I.e., $\underline{z}_k = z_k(\underline{m}_1, \underline{\mu})$ and $\overline{z}_k = z_k(\overline{m}_1, \overline{\mu})$.

4.2 Monotonicity Properties of μ_1 and μ_2

In the interval computation of $\mu_1(j)$ and $\mu_2(k)$, we do not take into account the fact that ξ_{kn} (and therefore also z_k) are not independent from m_1 and μ. Along the lines of step by step interval arithmetic, we treat $\mu_1(j)$ and $\mu_2(k)$ as if they were depending on the input parameters m_1, μ, and ξ_{kn}, with $M \leq k \leq N$ and $1 \leq n \leq M$. Thus, we are interested in monotonicity properties of $\mu_1(j)$ and $\mu_2(k)$ w.r.t. these input parameters.

From Eq. (2) one can see that $\mu_2(k)$ is monotonically increasing w.r.t. μ as well as w.r.t. ξ_{kn}. Thus, computation of an interval for $\mu_2(k)$ is straight forward and can be done along the lines of computations with the endpoints of intervals for μ and ξ_{kn}.

Next we consider μ_1. To reduce the effect of the dependency problem we avoid computation of intermediate intervals for β_j and $b(j)$. Instead, we substitute β_j and $b(j)$ in Eq. (1). For a shorter notation we consider $\mu_1(N-j)^{-1} = \varphi(j)$ and show monotonicity properties of $\varphi(j)$ w.r.t. m_1, μ, and z_j:

$$\begin{aligned} \varphi(j) &= \mu_1(N-j)^{-1} = m_1 + \beta_j b(j) \\ &= m_1 + \frac{\sqrt{m_1^2 + \frac{4j}{M^2\mu} z_j(1-z_j)(1-\alpha)m_1} - m_1}{2z_j}. \end{aligned} \tag{14}$$

Using the notation $\delta = \frac{4j}{M^2\mu} z_j(1-z_j)(1-\alpha)$, for the derivative w.r.t. m_1 we get:

$$\frac{\partial\varphi(j)}{\partial m_1} = \frac{\partial}{\partial m_1}\left[m_1 + \frac{\sqrt{m_1^2+\delta m_1} - m_1}{2z_j}\right] = 1 + \frac{1}{2z_j}\left[\frac{2m_1+\delta}{2\sqrt{m_1^2+\delta m_1}} - 1\right]. \tag{15}$$

Since $(2m_1+\delta)^2 = 4m_1^2 + 4\delta m_1 + \delta^2$ is greater than $\left(2\sqrt{m_1^2+\delta m_1}\right)^2 = 4m_1^2 + 4\delta m_1$, it follows that $\frac{2m_1+\delta}{2\sqrt{m_1^2+\delta m_1}} > 1$. Hence,

$$\frac{\partial\varphi(j)}{\partial m_1} > 1. \tag{16}$$

Derivation w.r.t. μ yields:

$$\frac{\partial\varphi(j)}{\partial\mu} = \frac{-j(1-z_j)(1-\alpha)m_1}{M^2\mu^2\sqrt{m_1^2 + \frac{4j}{M^2\mu} z_j(1-z_j)(1-\alpha)m_1}} < 0. \tag{17}$$

Finally, using the notation $r = 4b(j)(1-\alpha)$, the derivative w.r.t. z_j is:

$$\begin{aligned} \frac{\partial\varphi(j)}{\partial z_j} &= \frac{\frac{rz_j(1-2z_j)m_1}{2\sqrt{m_1^2+rz_j(1-z_j)m_1}} - \sqrt{m_1^2+rz_j(1-z_j)m_1} + m_1}{2z_j^2} \\ &= \frac{m_1\left[-2m_1 - rz_j + 2\sqrt{m_1^2+rz_j(1-z_j)m_1}\,\right]}{4z_j^2\sqrt{m_1^2+rz_j(1-z_j)m_1}} < 0, \end{aligned} \tag{18}$$

because

$$\begin{aligned} & -2m_1 - rz_j + 2\sqrt{m_1^2+rz_j(1-z_j)m_1} < 0 \\ \Longleftrightarrow\ & 4m_1^2 + 4rz_jm_1 - 4rz_j^2m_1 < 4m_1^2 + 4m_1rz_j + r^2z_j^2 \\ \Longleftrightarrow\ & -4rz_j^2m_1 < r^2z_j^2, \end{aligned}$$

which holds since $r, z_j, m_1 > 0$. From Eqs. (16), (17), and (18) it follows that $\mu_1(j)$ is monotonically decreasing w.r.t. m_1, and monotonically increasing w.r.t. μ and z_{N-j}. Thus, an interval for $\mu_1(j)$ can be obtained by evaluating the single value expressions from Eqs. (1) and (7) using the endpoint combinations: $(m_1, \mu, z_{N-j}) = (\overline{m}_1, \underline{\mu}, \underline{z}_{N-j})$ and $(m_1, \mu, z_{N-j}) = (\underline{m}_1, \overline{\mu}, \overline{z}_{N-j})$, respectively.

4.3 Computation of the Probabilities $p(j)$

To compute the throughput of the submodel (see Eq. (5)), the queue length probabilities $p(j)$ are required. If these probabilities are computed as defined in Eqs. (3) and (4), a similar problem as in the computation of the ξ_{kn} arises due to the normalisation step. Thus, we rewrite the computation of $p(j)$ such that as many factors as possible can be cancelled during the course of the computation. Using the notations: $t_1(k) = \prod_{l=1}^{k} \mu_1(l)$ and $t_2(k) = \prod_{l=1}^{N-k} \mu_2(l)$, $p(j)$ and G can be rewritten as follows:

$$p(j) \quad = \quad \frac{1}{G \cdot t_1(j) t_2(j)} \tag{19}$$

and

$$G \quad = \quad \sum_{l=1}^{N} \frac{1}{t_1(l) t_2(l)} . \tag{20}$$

Using the relations

$$\frac{t_1(j)}{t_1(k)} = \begin{cases} \prod_{l=k+1}^{j} \mu_1(l), & \text{if } k < j, \\ 1, & \text{if } k = j, \\ \prod_{l=j+1}^{k} \frac{1}{\mu_1(l)}, & \text{if } k > j, \end{cases} \quad \text{and} \quad \frac{t_2(j)}{t_2(k)} = \begin{cases} \prod_{l=N-j+1}^{N-k} \frac{1}{\mu_2(l)}, & \text{if } k < j, \\ 1, & \text{if } k = j, \\ \prod_{l=N-k+1}^{N-j} \mu_2(l), & \text{if } k > j, \end{cases}$$

we can rewrite the reciprocals of the probabilities $p(j)$ as follows:

$$p(j)^{-1} \;=\; \sum_{k=0}^{N} \frac{t_1(j) t_2(j)}{t_1(k) t_2(k)} \;=\; \sum_{k=0}^{j-1} \frac{\prod_{l=k+1}^{j} \mu_1(l)}{\prod_{l=N-j+1}^{N-k} \mu_2(l)} \;+\; 1 \;+\; \sum_{k=j+1}^{N} \frac{\prod_{l=N-k+1}^{N-j} \mu_2(l)}{\prod_{l=j+1}^{k} \mu_1(l)} . \tag{21}$$

5 Experimental Results

As already stated, rewriting expressions to reduce the effect of the dependency problem and using monotonicity properties of intermediate results can significantly reduce the amount of overestimation for interval results. In this section, we illustrate the effect of the reformulations described in Section 4 with the example of the rewritten expressions for the probabilities ξ_{kn} (see Eqs. (8) and (9) for the original expressions and Eqs. (10), (12), and (13) for the rewritten expressions).

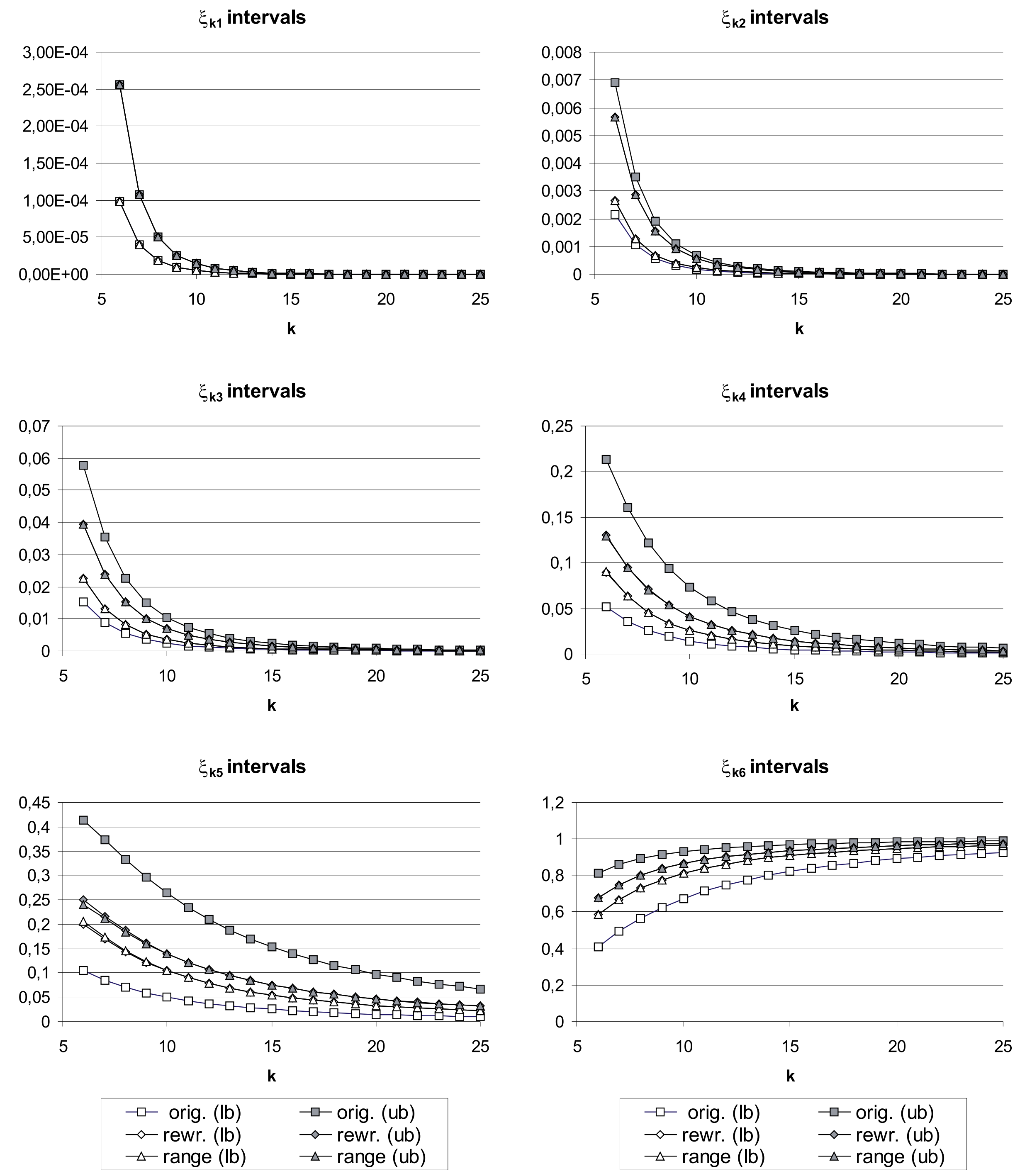


Figure 4: Interval results obtained for the probabilities ξ_{kn} comparing original and rewritten expressions. Each diagram shows results for a different value of n ($\xi_{k1}, \ldots \xi_{k6}$) for varying values of $k = 1, \ldots, 25$.

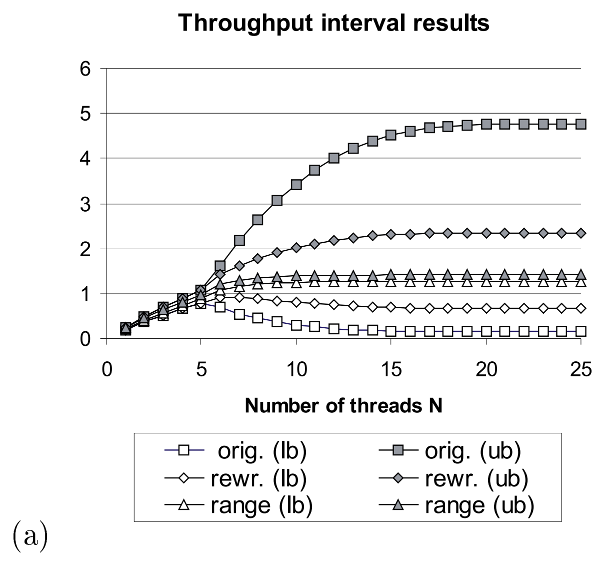
In the following experiments we use parameter values taken from evaluations described in [Llad 00]: $M = 6$ bean servers per container, $I = 20$ different bean instances, and a population of $N = 25$. The estimates for the service rate of the bean servers μ and the mean service time of the outer server m_1 are subject to uncertainty. Thus, these parameters are characterised by the intervals $\mu^{(iv)} = 1/4.1 \pm 5\%$ and $m_1^{(iv)} = 0.4 \pm 5\%$. Note that if one or more parameters are characterised by an interval, every (intermediate) result depending on these parameters is itself an interval.

Fig. 4 shows the intermediate interval results obtained for the probabilities ξ_{kn}, $k = 6, \ldots, 25$, $n = 1, \ldots, 6$. For each ξ_{kn}, three intervals are depicted: '*orig.*' denotes the interval results obtained using the original expressions without interval splitting, '*rewr.*' denotes intervals for ξ_{kn} obtained using the rewritten expressions without interval splitting, and '*range*' denotes the range for the ξ_{kn} intervals obtained with interval splitting to an accuracy of $\epsilon = 10^{-9}$ (for comparison purposes). Every interval is depicted by showing its lower (lb) and upper bounds (ub). It can be seen that if the original expressions are used, the dependency problem causes a significant overestimation of the ξ_{kn} intervals for $n > 1$. If the rewritten expressions are used for the interval calculation of the ξ_{kn} intervals, they almost precisely match the exact range of values obtained via interval splitting.

Fig. 5 shows the effect of using the original respectively rewritten expressions for the intermediate probability intervals ξ_{kn} when computing intervals for the submodel throughput $T(N)$ (see Eq. (5)). During the computation of the throughput, the other intermediate results ($\mu_1(j)$, $\mu_2(k)$, $p(j)$) are computed using the adaptations as described in Section 4. Fig. 5(a) depicts throughput intervals for populations $N = 1, \ldots, 25$. It can be seen in this figure that using the original expressions for ξ_{kn}, the throughput interval is much more overestimated than the throughput interval obtained using the rewritten expressions for ξ_{kn}.

Unfortunately, due to the dependency problem during the computation of $T(N)$, also the throughput intervals obtained by using the rewritten expressions are more than 10 times as wide as the actual range of the throughput (the innermost intervals in Fig. 5(a)). Thus, in both cases, interval splitting has to be applied to obtain reasonable tight enclosures of the throughput range. However, even though interval splitting may be necessary for both, original as well as optimised (w.r.t. interval computation) expressions, the computational effort is significantly reduced when using the rewritten formulae. Fig. 5(b) depicts the computational complexity required to obtain the range for the throughput with an accuracy of $\epsilon < 10^{-2}$. To obtain the range of the throughput, the *selective splitting with midpoint evaluation* (SSME) approach is used, which performs both, interval as well as conventional evaluations [Luth 00]. Thus, for each version (original and rewritten expressions), three graphs are shown: the number of interval evaluations (iv), the number of single value evaluations (sv), and the weighted sum $iv + sv/2$ (total) — the computational complexity for an interval evaluation is approximately twice as high as for a single value evaluation. Note that using the original expressions for the probabilities ξ_{kn} increases the number of evaluations during the interval splitting algorithm by a factor of more than 5. Similar observations are made if the original expressions of the other rewritten formulae are used.

This example shows that the adaptation of existing solution techniques to interval param-



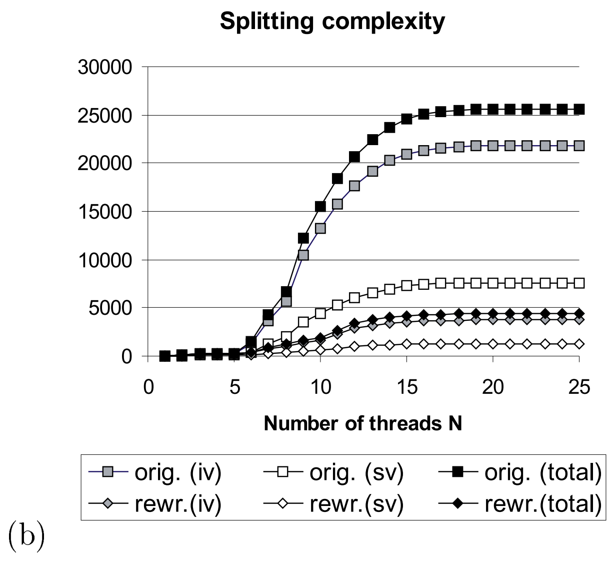


Figure 5: Comparison of original and rewritten expressions: (a) throughput interval results and (b) computational complexity for interval splitting algorithm.

eters has to be done with great care. For as many steps as possible, intermediate expressions have to be optimised for an efficient interval computation. I.e., wherever possible, monotonicity properties as well as possibilities to cancel occurrences of interval parameters have to be exploited.

6 Conclusions

Parameters of quantitative performance models of computer and telecommunication systems are not always known exactly. Parameter intervals are a convenient way to capture uncertainty in model parameters. Existing analytical solution algorithms can be adapted to intervals by replacing conventional arithmetical operations and elementary functions by corresponding interval operations. An unpleasant effect of interval arithmetic is the so-called dependency problem causing overly wide interval results. Interval splitting methods to overcome this problem have been proposed in the literature. In this paper we present the application of such methods to the approximate solution of a queueing network modelling an Enterprise JavaBeans server implementation. In this model, service rate parameters are characterised as intervals in order to capture associated uncertainty. The original performance measure formulae are optimised for an efficient interval arithmetical solution. A numerical example illustrates the effect of these modifications on the efficiency of the interval splitting approach that is used to obtain tight performance measure intervals.

The example resolution suggest that interval adaptation has to be done with great care. Although an interval-based solution algorithm can be obtained by simply substituting conventional by interval operations in the original solution algorithm, the analyst is advised to exploit (partial) monotonicity properties of the solution algorithm wherever possible. Furthermore, expressions should be rewritten to reduce the number of occurrences of interval parameters as much as possible. If such techniques are applied in the adaptation, interval results can be obtained with reasonable additional computational effort.

In future work the interval version of this model will also be used for sensitivity analysis. For example, it may be interesting to examine how the width of performance measure intervals depends on the width as well as on the (relative) position of the parameter intervals for the mean service rates of the container server and the bean servers.

References

[Chat 99] J. Chattratichat, J. Darlington, Y. Guo, S. Hedvall, M. Kohler, and J. Syed. "An Architecture for Distributed Enterprise Data Mining". In: *Proc. of the 7th International Conference on High Performance Computing and Networking Europe (HPCN Europe'99, April 12–14, Amsterdam, The Netherlands)*, p. , April 1999.

[Harr 00] P. G. Harrison and C. M. Lladó. "Performance Evaluation of a Distributed Enterprise Data Mining System". In: B. R. Haverkort, H. C. Bohnenkomp, and C. U. Smith, Eds., *Proceedings* 11th *Int. Conf. on Computer Performance Evaluation Modelling Techniques and Tools (TOOLS 2000, March 27–31, Schaumburg, IL, USA), LNCS 1786*, pp. 117–131, Springer Verlag, Berlin, March 2000.

[Harr 93] P. G. Harrison and N. M. Patel. *Performance Modelling of Communication Networks and Computer Architectures.* Addison-Wesley, 1993.

[Have 95] B. Haverkort and A. M. Meeuwissen. "Sensitivity & Uncertainty Analysis of Markov-Reward Models". *IEEE Transactions on Reliability*, Vol. 44, No. 1, pp. 147–154, March 1995.

[Kell 79] F. Kelly. *Reversibility and Stochastic Networks.* Wiley, 1979.

[Llad 00] C. M. Lladó and P. G. Harrison. "Performance Evaluation of an Enterprise JavaBean Server Implementation". In: *Proceedings* 2nd *International Workshop on Software and Performance (WOSP 2000, September 17–20, Ottawa, Canada)*, pp. 180–188, September 2000.

[Ltd] InforSense Ltd. "Kensington 2000". http://Kensington.doc.ic.ac.uk.

[Luth 00] J. Lüthi and C. M. Lladó. "Splitting Techniques for Interval Parameters in Performance Models". submitted for publication, preprint available as Technical Report No. 2000-07, Fakultät für Informatik, Universität der Bundeswehr München, D-85577 Neubiberg, Germany, 2000.

[Luth 98a] J. Lüthi. "Histogram-Based Characterization of Workload Parameters and its Consequences on Model Analysis". In: *Proceedings of the MASCOTS'98 Workshop on Workload Characterization in High-Performance Computing Environments, July 19–24, 1998, Montreal, Canada*, pp. 1/52 – 1/64, July 1998.

[Luth 98b] J. Lüthi and G. Haring. “Mean Value Analysis for Queueing Network Models with Intervals as Input Parameters”. *Performance Evaluation*, Vol. 32, No. 3, pp. 185–215, April 1998.

[Maju 91] S. Majumdar. “Interval Arithmetic for Performance Analysis of Distributed Computing Systems”. In: *Proceedings of the Canadian Conference on Electrical and Computer Engineering, Quebec, Canada, September 25–27*, September 1991.

[Maju 95] S. Majumdar and R. Ramadoss. “Interval-Based Performance Analysis of Computing Systems”. In: P. Dowd and E. Gelenbe, Eds., *Proceedings of the Third International Workshop on Modeling, Analysis and Simulation of Computer and Telecommunication Systems (Durham, NC, USA, Jan. 18-20, 1995)*, pp. 345–351, IEEE Computer Society Press, January 1995.

[Micr 99] Sun Microsystems. “Enterprise JavaBeans 1.1 Architecture”. http://java.sun.com/products/ejb, 1999.

[Neum 90] A. Neumaier. *Interval methods for systems of equations.* Vol. 37 of *Encyclopedia of Mathematics and its Applications*, Cambridge University Press, Cambridge, 1990.

[Nied 78] H. Niederreiter. “Quasi-Monte Carlo Methods and Pseudo-Random Numbers”. *Bulletin of the American Mathematical Society*, Vol. 84, No. 6, pp. 957–1041, November 1978.

[Rubi 81] R. Y. Rubinstein. *Simulation and the Monte Carlo Method. Wiley Series in Probability and Mathematical Statistics*, John Wiley & Sons, New York et al., 1981.

[Skel 74] S. Skelboe. “Computation of Rational Interval Functions”. *BIT*, Vol. 14, pp. 87–95, 1974.