Available online at www.sciencedirect.com

SCIENCE DIRECT®

Performance Evaluation 58 (2004) 43–80

PERFORMANCE EVALUATION An International Journal

www.elsevier.com/locate/peva

Configuration of distributed message converter systems

Thomas Risse[a,*], Karl Aberer[b], Andreas Wombacher[a], Mike Surridge[c], Stephen Taylor[c]

[a] Fraunhofer IPSI, Integrated Publication and Information Systems Institute, Dolivostrasse 15, 64293 Darmstadt, Germany
[b] School of Computer and Communication Sciences, EPFL, CH-1015 Lausanne, Switzerland
[c] IT Innovation Centre, 2 Venture Road, Chilworth Science Park, Southampton SO16 7NP, UK

Received 1 July 2002; received in revised form 28 April 2004; accepted 3 May 2004

Available online 28 July 2004

Abstract

Finding a configuration of a distributed system satisfying performance goals is a complex search problem that involves many design parameters, like hardware selection, job distribution and process configuration. Performance models are a powerful tool to analyze potential system configurations, however, their evaluation is expensive, such that only a limited number of possible configurations can be evaluated. In this paper we present a systematic method to find a satisfactory configuration with feasible effort, based on a two-step approach. First, performing a queuing network analysis a hardware configuration is determined and then a software configuration is incrementally optimized by simulating Layered Queuing Network models. We applied this method to the design of performant EDI converter systems in the financial domain, where increasing message volumes need to be handled due to the growing importance of B2B interaction.
© 2004 Elsevier B.V. All rights reserved.

Keywords: System configuration; Performance modeling; Queuing networks

1. Introduction

Electronic data interchange is an important aspect in the implementation of business processes. The exchange of data between heterogeneous systems requires support for different data formats (EDIFACT, XML, etc.). Enterprises use different proprietary in-house formats. So the incoming and outgoing messages must be converted from the inbound format to the in-house format, as well as from the in-house format to the outbound format. The volume of data each enterprise delivers and receives will grow rapidly in the next years. This leads to growing demands on the performance of EDI converter systems.

This was the motivation to investigate within the POEM (Parallel Processing of Voluminous EDIFACT Documents) project the question of how performant parallel converter systems can be built, based

* Corresponding author. Tel.: +49-6151-869906.
E-mail addresses: risse@ipsi.fhg.de (T. Risse), karl.aberer@epfl.ch (K. Aberer), wombach@ipsi.fhg.de (A. Wombacher), ms@it-innovation.soton.ac.uk (M. Surridge), sjt@it-innovation.soton.ac.uk (S. Taylor).

on the typical infrastructures currently available in large enterprises, such as shared memory processing architectures (SMP). Also distributed architectures involving different machine types have been considered.

A critical step in the implementation of a distributed system, such as a performant parallel message converter system, is to identify a hardware and software configuration for it, given performance requirements and system constraints derived from the business requirements. For this problem we could not identify any existing systematic approach.

Though at a first glance it appears that this problem can be tackled using standard search or constraint problem solving techniques once the problem has been modeled, we have to take account of the fact that there exists a combinatorial search space of possible configurations and obtaining information on the performance of specific configurations for bounding the search space is extremely expensive. Evaluating the performance of a system configuration during search can in principle be performed in two ways:

(1) Testing the real system: this is (economically) expensive and can be done only in very limited ways. Selected measurements of single system components in order to obtain basic information for building realistic models of composite systems are feasible.
(2) Evaluating a system model: Based on information obtained from system measurements of components, performance models for composite system configurations can be built. The analytical solution of such models is usually only possible for relatively simple and homogeneous models. In general, we have to rely on the simulation of performance models as soon as we model real systems at some level of detail. Simulations of performance models on the other hand are computationally intensive. This is not problematic in the case of modeling and analyzing a single existing system, which is the standard application of performance models. However, when exploring a space of possible configurations the use of simulations needs to be strictly limited.

Thus, given the limited possibilities of testing system configurations the straightforward application of standard search or constraint problem solving techniques would be prohibitively expensive. Also the behavior of system configurations is in many cases highly non-linear, i.e. small changes in the configurations can imply large changes in performance. Information obtained from testing specific configurations does not in all cases easily extrapolate to similar configurations.

For predicting the system performance we use a performance model based on *Layered Queuing Network* (*LQN*) *simulation*. Such a model allows predicting the performance of one specific system configuration. Simulating all possible configurations would be prohibitively expensive, considering their large number and the high cost of a single LQN simulation. For example, the configuration of a system such as described in Section 5.3 would require about 107 simulations to test all possible configurations. Also applying branch-and-bound techniques would still require approximately 100 simulations.

We focus on exploiting existing knowledge on the problem structure in order to reduce the search space. The remaining search problems can then be solved applying straightforward optimization methods and reduce the effort for performance evaluation. For example, in case of the system from Section 5.3 we would require only 5–10 simulations.

In this paper we give a complete overview of the solution we developed for configuration of a distributed system for parallel processing of different job classes on a heterogeneous, distributed architecture with given throughput and response time goals. The solution matches the requirements set out before. It consists of

(1) A method to search suitable configurations that minimizes the use of expensive methods for performance evaluation to the largest degree. The critical design parameters are the selection of hardware, the distribution of the different jobs to different hosts, and the configuration of processes on the hosts. This method had been first presented in [1] and is refined in this paper.
(2) Software tools that have been developed in order to automate the application of our method to a given configuration problem.
(3) An evaluation of the method based on a real-world application and system, namely of EDI converters. Since our method comprises a strong heuristic component we consider this evaluation as particularly important in order to reassure that the method we propose produces useful and relevant results in practice.

With respect to the search of a configuration (1) we pursue the following strategy:

(1) *Hardware configuration*: We approximate single host performance by a coarse model that requires few key parameters which are relatively inexpensive to obtain from single host testing. This approximate model can be solved analytically. Based on it we perform hardware selection and determine the workload distribution for the selected host configuration, such that the required performance can be achieved according to the model.
(2) *Software configuration*: Based on the workload distribution and hardware configuration determined in hardware configuration an LQN model of the complete system is built. It is used to determine a software configuration that actually achieves the performance that has been predicted in the hardware configuration. Since simulations of the complete model are rather expensive, we use a greedy heuristic, which tries to minimize the number of simulations required for finding the optimal software configuration.

Our heuristic approach to system design does not necessarily result in “the” optimal configuration. However, since the business requirements (such as expected message load) are only approximate, we need to find a reasonable configuration that covers the processing requirements approximately at acceptable cost. The configuration method avoids obvious design flaws, like gross over- or undersizing of the system or bottlenecks. Business requirements, like available hardware or hardware cost, often influence the decisions on the hardware configuration substantially and can be taken into account.

A software tool to apply the proposed method has been implemented in Java. It integrates tools for symbolic computation (Mathematica [2]) and an LQN solver [3]. The implementation allows us to automatically determine a complete configuration based on the problem specification.

For evaluation we configured a real message converter system applying our method and conducted performance measurements of the real systems. These tests confirmed that our method finds a hardware configuration that matches business requirements and a software configuration that is optimal for the given hardware configuration. Thus we confirmed the utility of our approach. The system tests also provided interesting insights on the role of scheduling for the system performance. In particular, we could show that our modified bin-stretching approach [4], which uses processing time estimation and was used in the implementation, produced the best results. A sensitivity analysis was performed in order to determine the dependency of system performance on deviations from the expected workload. This analysis shows that variations in the message distribution often have only slight impact on the system performance, thus accurate workload prediction by users are not required to provide adequate performance under changing workload. In summary, these evaluations confirmed that our method provides a useful tool in order to efficiently and accurately determine system configurations matching business requirements.

1.1. Related work

A large body of related work exists in the area of performance prediction and capacity planning for parallel and distributed systems. The prediction methods can classified into stochastic models and deterministic models. Stochastic models take into account the variance of execution times. But as shown in [5] stochastic models have never been evaluated with respect to their accuracy when modeling real-world applications because they require complex solution techniques. In deterministic models the variance of the execution time is assumed to be negligible. In [5,6] the authors show the practical usability of deterministic models. Some prediction methods require a special architectural environment, e.g. predictions in the NOW computing model [7] are based on program execution graphs. Most of the methods are using queuing networks. In [5,8,9] queuing networks are used to model parallel or distributed systems and communication networks. Also database systems have been evaluated in [10] using queuing networks.

Fontenot describes the general problem of software congestion [11]. He points out that software bottlenecks could be avoided by using multiple parallel software instances. But no method has been given for determining the necessary number of software instances to achieve a specific performance goal.

Capacity planning is another area, where performance predictions of distributed systems are made. In [9] the authors developed a method for capacity planning of client–server systems by investigating several workload parameters and applying them to a queuing network model of the system. A similar approach is used in [12,13] to predict the performance of a large-scale data-intensive information system. If the configuration does not meet the performance goals the authors propose a manual refinement of the system.

In [14] the authors propose an LQNS-based automated method to optimize the response time of a pre-configured system. The optimization is performed by adapting the task priorities, the task allocations to processors and the splitting of task.

To summarize, most of the existing work is focusing on predicting or evaluating the performance of a given, specific system. Devising an automatic method using performance models in order to search for a hardware and software configuration of a system within given constraints such that performance goals regarding throughput and response time are achieved is to our knowledge new.

1.2. Overview of the paper

In Section 2 we provide an overview of our approach including application background and system architecture. In Section 3 we give background on LQN models and describe the LQN model that has been built for our message converter application. Section 4 introduces our configuration method. In Section 5 we describe its application to our business case. In Section 6 we present the results from our real system measurements, which we performed to verify the validity of our method. Finally Section 7 gives a conclusion and an outlook on possible continuations of our work.

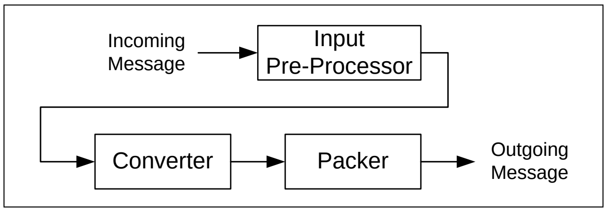

Fig. 1. Sequential processing of messages.

2. Overview of the approach

2.1. Application background: message processing

In the banking sector large EDI messages containing transaction information need to be converted from an inbound format to formats of in-house systems while keeping strict deadlines. The processing of messages requires a sequence of different steps that are applied to each message, as shown in Fig. 1.

The 'Input Pre-Processor' regularly checks for the arrival of new messages. Afterwards it analyzes the messages regarding the syntax format (e.g. EDIFACT [15]), message structure and size. The information about message structure and size is used for the distribution of messages among the available hosts performing the conversion. The 'Converter' task transforms the message to an intermediate format, which is finally converted to the target format by the 'Packer' task. From requirements analysis we obtain information on the expected message volumes, the message size distribution and the requirements on response time. Qualitative requirements are scalability, availability, and constraints on the type of available hardware.

2.2. System architecture

To satisfy requirements on availability, reliability, scalability and high throughput, a parallel architecture is used. The system can be built from different host types, allowing the use of existing hardware and the incremental extension of the system with new hardware. The generic architecture of the system is shown in Fig. 2. The global scheduler distributes incoming messages to the individual hosts. The distribution strategy it uses will be based on the configuration determined by our system design method. The local scheduler controls the execution of tasks and the distribution of the tasks to the processors on the different hosts. The local scheduler is tightly coupled with the operating system. A more detailed description of the architecture and the processing steps can be found in [16].

2.3. System configuration approach

In Fig. 3 we provide an overview of the system configuration process. In a first step the available hardware is determined from the business constraints. For the available hardware, hardware measurements are performed for obtaining basic performance data that can be fed into the subsequent configuration process. Furthermore software constraints, e.g. the maximum number of process instances for a processing step, are determined. These constraints are used in the software configuration step to avoid the selection of unrealizable software configurations. Once the hardware and software properties are determined the

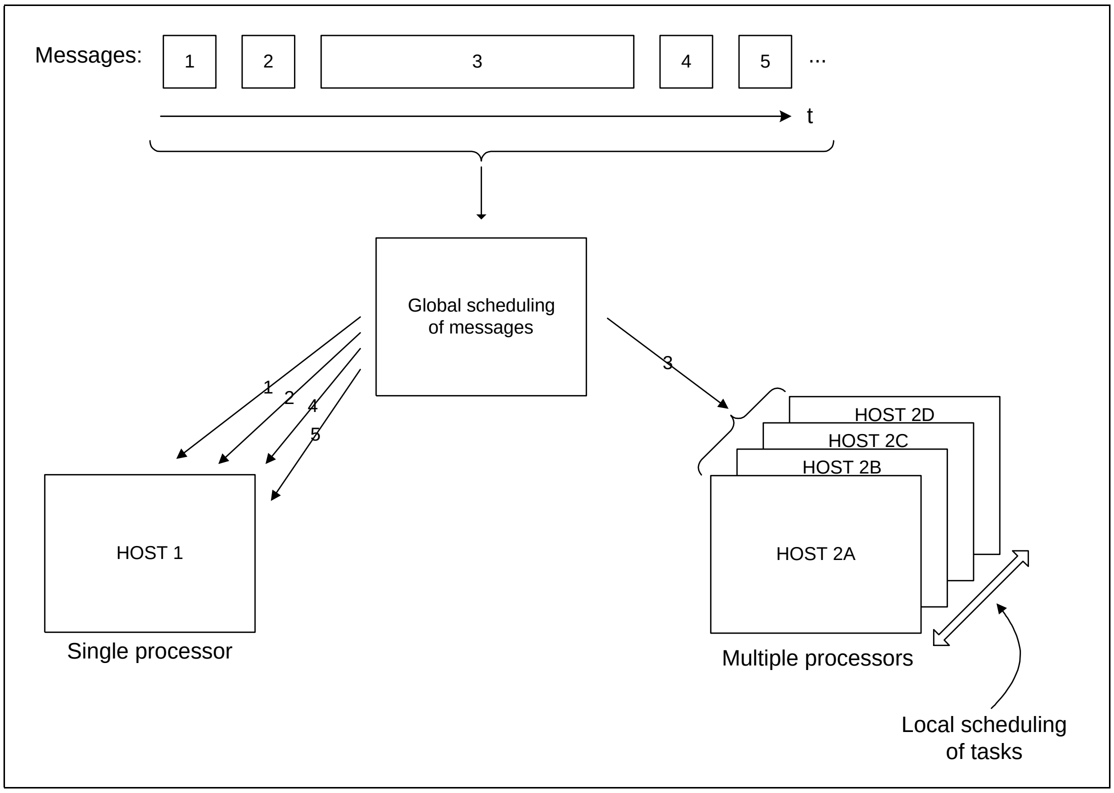

Fig. 2. System architecture.

processing goals in terms of throughput and response time for the different kind of tasks are specified. The first main step in our method is then hardware configuration. A system configuration selecting from the available hardware is determined, which has the capacity to achieve the processing goals. This step is based on approximate, analytical performance models that abstract from the properties of specific software configurations. They can be evaluated without using simulations.

Based on the selected hardware configuration a performance model based on Layered Queueing Networks (LQN) is constructed, that considers also the software's process structure. Using simulations of the performance model and modifying the model by a greedy search method an optimized process configuration is identified, satisfying the processing goals achievable by the selected hardware configuration. A final simulation of the complete system is then performed in order to verify that the processing goals have been reached. This verification may fail when no software configuration can be found that satisfies the performance predicted in the hardware configuration step.

In such a case a new iteration of the hardware configuration step is performed. Other hardware configurations may be identified since hardware configuration is performed in a non-deterministic manner taking into account business requirements, such as available hardware and hardware cost. If no suitable hardware configuration can be found it is possible to adapt the processing goals. If this also does not result in a suitable configuration, as a last resort, alternative hardware needs to be considered. In general adaptations of processing goals and available hardware can be expected to occur rather rarely since on

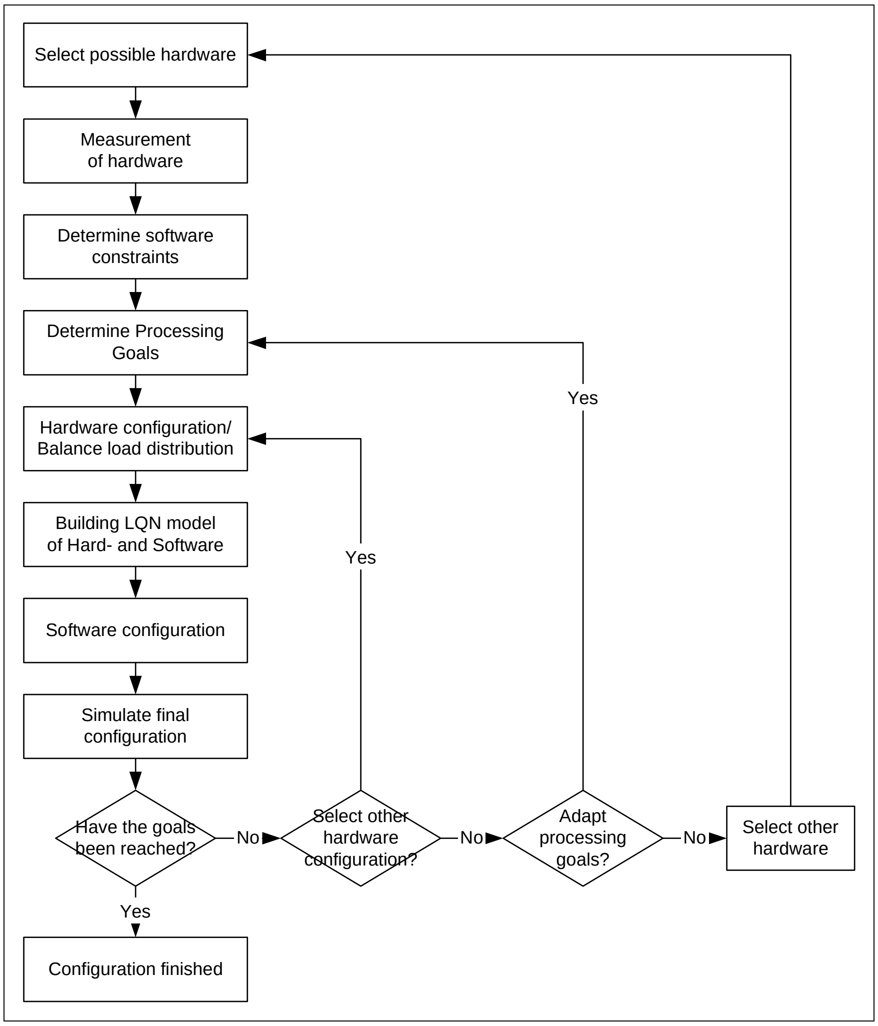
Select possible hardware
Measurement of hardware
Determine software constraints
Determine Processing Goals
Hardware configuration/ Balance load distribution
Building LQN model of Hard- and Software
Software configuration
Simulate final configuration
Have the goals been reached?
No
Select other hardware configuration?
No
Adapt processing goals?
No
Select other hardware
Yes
Yes
Yes
Configuration finished

Fig. 3. Configuration process graph.

the one hand the processing goals and hardware selection are mostly business-driven decisions, on the other hand the system designers have at least a very rough understanding of the necessary hardware performance required to satisfy the processing goals. Finding within those constraints a solution making optimal use of resources is where we provide support through our configuration method.

2.4. *Using exhaustive search*

An alternative to our system configuration method would be to use exhaustive search. This requires to model each possible hardware and software configuration, determine the throughput and response time values for each model by simulation and select finally the best configuration. For a system as described in Section 5.3 with two hosts, two different host types, three processing steps and up to four instances per processing step, the total number of possible models would be 12 288. Their simulation requires more than 30 h using the existing simulation tools.

However, in addition determining each of the models is non-trivial in itself. For the system model it is necessary to know the average message distribution among the hosts, which depends in turn on the configuration of software and hardware. The message distribution among the hosts can only be determined by an additional simulation of the scheduling or load balancing mechanism, which considers the software configuration. A naive distribution scheme, e.g. proportional to the host processing capacities, will lead to inaccurate results.

In our method the result of the hardware configuration step provides on optimized distribution of the messages among the hosts. Hence an additional scheduling simulation is not necessary as long as the scheduler used in the system guarantees the determined distribution.

3. Performance modeling

Response time and throughput are key factors for defining the quality of a system [17]. Hence to achieve the performance requirements it is necessary to understand the effect of various configuration decisions at an early stage. As the systems are normally not available during the design phase, a model is required to analyze the system behavior. Such a model must be able to represent different types of tasks and resources with synchronous and asynchronous execution of tasks. Also the simulation of the distribution of tasks among several processor and hosts must be possible.

A common approach to performance modeling is based on Queuing Networks (QN) [18–20]. The standard model of queuing networks is restricted to the modeling of hardware servers.

The queueing network model was extended by Woodside et al. [21,22] with Stochastic Rendezvous Networks (SRVN) to model both hardware and software servers. SRVNs differ from the classical QNs in two ways. First, each node in the model can act both as client and as server. Second, servers in SRVNs can have two (or more) phases of execution. The first phase models the blocking of a client by a server during a request (rendezvous). Subsequent execution phases of the server occur after the server replies to the request; the client and server can then execute in parallel.

Several approaches exist for solving the model. The exact method by translating the model via Petri Nets into Markov Chains can be used for small models. For large models the state explosion makes this solution impractical for most real systems [7,23]. Other approaches approximate the solution by adapting the *Mean Value Analysis* (MVA) of Queuing Networks to SRVNs [23]. The *Method of Layers* introduced by Riola [24] divides the complete model into several submodels (layers), which are solved by the MVA. The model in the method of layers differs from the SRVN in that it distinguishes between software and hardware servers. Woodside et al. combined the SRVN and the method of layers to *Layered Queuing Networks* [17]. Further enhancements and generalizations can be found in [8].

3.1. Layered queuing networks

The LQN model consists of several components. The core of LQN models consists of directed acyclic graphs whose nodes are tasks with service entries. A task processes calls to its entries in FIFO order using a waiting queue. It consists of one or more processors. This technique is also known as single respectively multiple servers. Hence the number of entries executed in parallel depends on the number of assigned

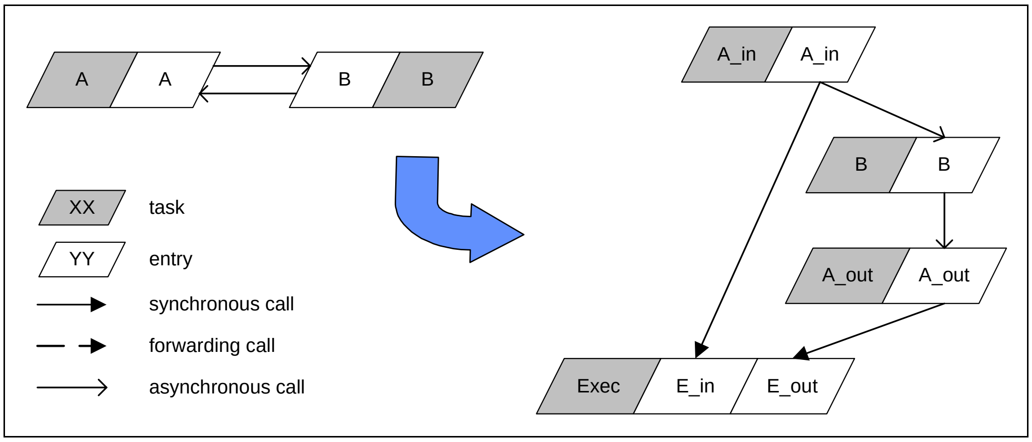

Fig. 4. Transformation of cyclic graphs.

processors. We distinguish three types requests, respectively arcs, in the model:

- *synchronous requests* with a reply.
- *asynchronous requests* without a reply.
- *forwarded requests*: The reply to a synchronous request is directed to another server, not back to the client (with some probability *P*). This can nest to any level.

The execution of entries is divided into two *phases*. The first phase is the service phase. Within this phase the calling client is blocked till the first phase of the server is terminated. The second phase is executed in parallel to the client. Several calling conventions, like RPC or ADA rendezvous, can be modeled with the concept of execution phases.

3.2. Transformation rules

3.2.1. Transformation of cycles

Cycles occur in software systems frequently, e.g. a controlling process calls asynchronously a subprocess. This subprocess notifies the control process by calling it again. Hence this forms a simple cycle. Cycles cannot be directly modeled within an LQN. A cyclic relation in a system can be transformed to an acyclic model by the use of semaphores. In [25] the following transformation rule has been derived to transform a system model with cycles and open arrivals into a valid LQN model.

A task *A* participating in a cyclic relation is split into three tasks *A_in*, *A_out* and *Exec*. Hence the entry of task *A* is distributed among tasks *A_in* resp. *A_out* with zero service time for their entries. The service time of the entry of *A* is moved to task *Exec*. Task *A_in* entry *A_in* calls task *B*. Task *B* calls task *A_out*. The entries *A_in* and *A_out* have to make requests to task *Exec* in order to execute on the CPU. This is depicted in Fig. 4.

3.2.2. Transformation of open arrivals

Two types of Queueing Networks are distinguished: closed networks and open networks. Closed networks are used for determining an equilibrium state of the network model. Closed networks do not have

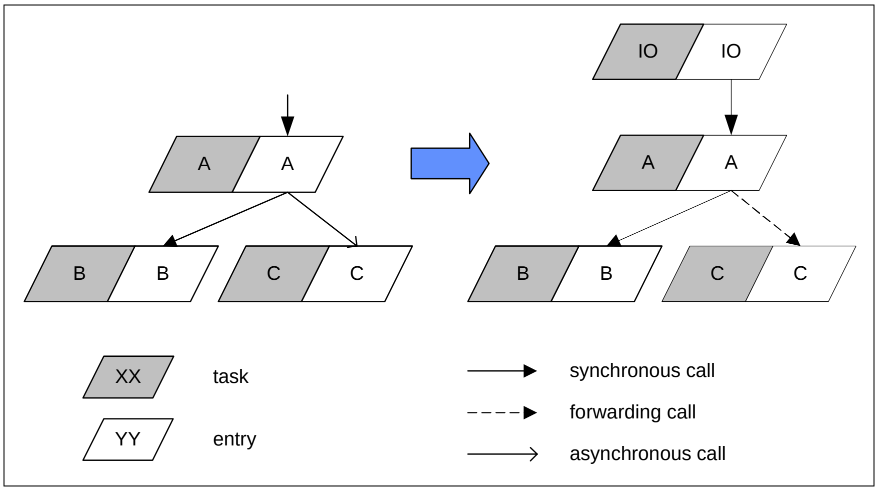

Fig. 5. Transformation of open arrivals.

a specific arrival rate at the starting point of the network, which can be determined from outside, but consider the highest possible arrival rate of the network. Open networks process an arrival rate parameter, for investigating the behavior of the model while the system load is increased. Because we are interested in determining maximum throughput, we only need closed network models. Hence in [25] the following rule is given to transform an open network to a closed network.

In networks with open arrivals a node receives messages from outside at a rate λ. Any LQN model with open arrivals can be turned into a closed model by replacing the open arrival with a "fixed" number of pure clients with low service times. This transformation ensures a saturation of the system without overflowing the task queues [25]. In addition, asynchronous calls have to be substituted by forwarding calls. The forwarding call assures that a new job is accepted only when the system has free capacities. The synchronous and forwarding calls are not affected by this transformation. This is depicted in Fig. 5.

3.3. *Modeling of a converter system*

Within the LQN model of the converter system a process instance is modeled as a task. Multiple instances of a process are modeled by using multiple processors for a task. The processors of a task within the LQN model are not used to model the system CPU or other resources because a task uses several resources and has different execution demands on each resource. Hence resources like CPU, file I/O and DB are also modeled as tasks.

The construction of the LQN model is based on the data flow within the system. Fig. 6 shows the data flow within the converter system. The system has an open arrival I/O element; therefore the model is an open network model. The model also contains cyclic asynchronous connections, e.g. between the local scheduler and the unpacker. Thus, the model has to be transformed by the rules described in Section 3.2. The resulting model for the reference host with one message type is depicted in Fig. 7. To build the complete model for two message types with different processing characteristics all processing steps have to be duplicated. The model does not include cycles and has been transformed into a closed network model.

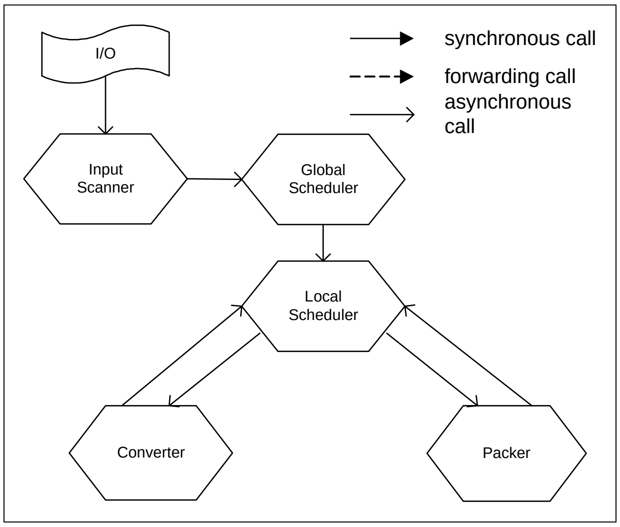

Fig. 6. Simple model of the system.

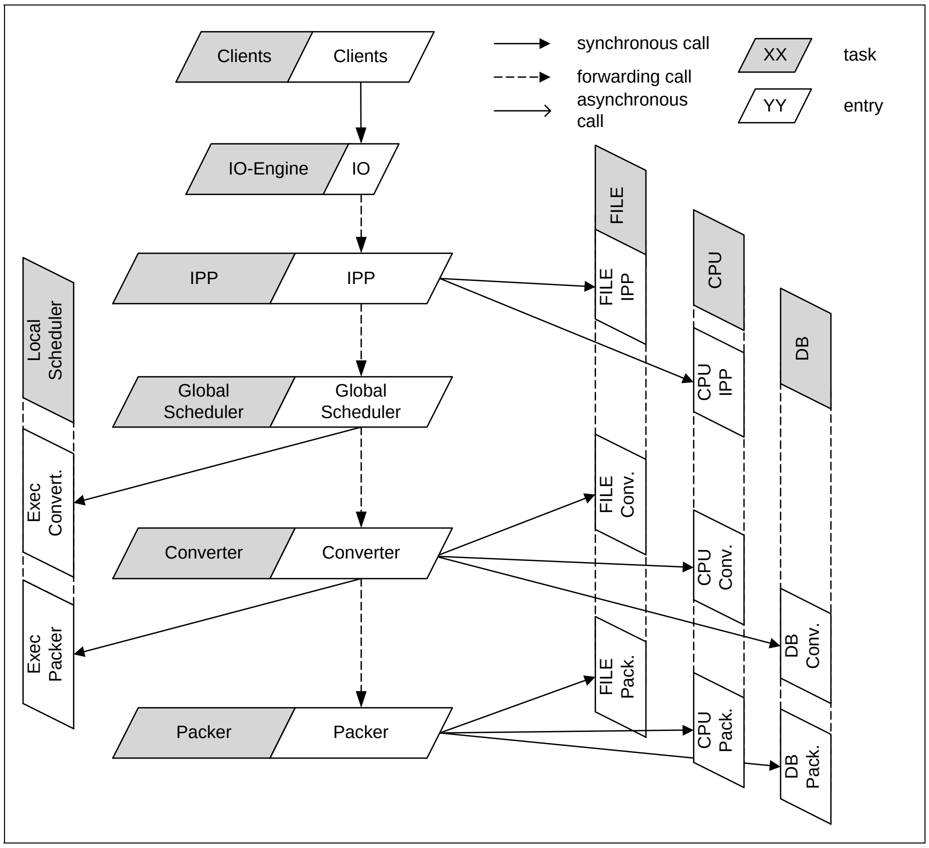

Fig. 7. LQN model of one host and one message type.

3.4. Simulation tools

As already mentioned several approaches exist for solving an LQN model.

LQN's can be solved using analytic methods or simulation. The analytic method is impractical for most real systems due to the state explosion after translating them into Petri Nets or Markov Chains [7,23]. Simulation of LQN models gave accurate results (order of 10–30% compared to live system), but are nevertheless computationally expensive.

The 'TimeBench' tool [3] helps to construct and simulate LQN models and to generate code for different target environments. The LQNS tool (Layered Queuing Network Solver) [3] is used for solving LQN models but it includes a 'Multi Point Solver' (MultiSRVN) for examining the effect of parameter variations. Hence it can be used to analyze different system configurations or for sensitivity analysis. All solvers are able to use different methods for solving LQN models, e.g. 'Mean Value Analysis' [23] or 'Method of Layers' [24]. The LQNS tool is used for our simulations in Section 5.

Even if the solvers compute approximate solutions, the cost of simulations can vary substantially. The simulation of a single host can be done within a few minutes. The simulation of a complete model with e.g. only three hosts as presented in [10] required already approximately 40 h. So it is necessary to minimize the number of simulation runs. For that reason we have developed a new system design method.

3.5. Model parameters

The model parameters we rely on are the service times of each entry within a task. In the model described in Section 3.3 the service time represents the total time a processing step makes use of some resource during its execution. In order to determine the parameters we were using timestamps that were integrated into the program code of the real system. These measurements have been performed for each message and host type.

3.6. Evaluation of the LQN model

We have compared the performance predicted by the LQN model with the behavior of the real system. That is discussed in more detail in Section 5. We conducted the analysis by comparing response times at different arrival rates and determining system saturation, i.e. the highest arrival rate at which the response time does not increase. It turned out that the performance model is sufficiently precise and the deviations are within the range of measurement precision.

4. System design method

The goal of our system design method is to configure the software and hardware of a distributed system, such as a message converter system, in a systematic and efficient way. The problem is a complex optimization problem with many design dimensions. Expensive system simulations have to reduced to a minimum. A direct and naive application of standard search methods such as branch-and-bound [26], simulated annealing [27] or genetic algorithms [28] forbids itself. For example the direct application of a branch-and-bound search for a very simple case, such as the one we will study in Section 5.2, would require approximately 100 simulations, whereas we can reduce these with our method to about

5–10. In addition, the simulation times are not uniformly distributed. With our strategy we specifically avoid very expensive simulation runs. This reduces the simulation time needed by orders of magnitudes, e.g. from days to minutes. We accomplish this by exploiting knowledge on the problem structure and using heuristics to reduce the size of the search space. In particular we partition the search problem into several subproblems which can be solved separately, some of them completely analytically. For each of the subproblems different search methods can be applied. Our main focus is on the problem decomposition, whereas the application of refined search methods leaves room for further development.

4.1. Goal parameters

The goal parameters are derived from the business requirements. They need to be satisfied by the selected system configuration. Specifically they are:

Expected message distribution: The messages are clustered into classes with similar processing characteristics. In the following we assume that the processing of messages is only sensitive to the message size. Thus the message distribution can be given as a set MT such that for $m \in MT$ we have (s_m, f_m), where s_m is the average size of the message in bytes and f_m is the frequency of messages of type m measured, e.g., in terms of messages/hour. Hence the required throughput, measured in messages/hour, is $T^m_{\text{req}} = f_m, m \in MT$.

Hardware constraints: The hardware configuration is a multi-set H with elements from the set of possible host types HT. Certain machine configurations can be excluded in response to business requirements. For example, we will use a minimal $\min h$ and maximal $\max h$ number of hosts that are allowed ($\min h \leq |H| \leq \max h$). Or, certain hosts have to occur in the configuration (e.g. existing hardware).

Expected response time R^m_{req}: The response time is the sum of waiting time and processing time. The expected maximal response time is specified for each message type $m \in MT$.

4.2. Overall approach

The configuration method is based on the LQN model that has been introduced in Section 3, and which is based on model parameters that are derived from the real system. In order to limit the number of simulations of the complete converter system during the system configuration process, we proceed as follows.

From the simulations of the LQN models we created for each combination of host type $h \in HT$ and message type $m \in MT$, we determine approximate values for the minimal response time and the maximal throughput. We determine minimal response time by running a simulation with an extremely low utilization and maximal response time by a simulation with extremely high utilization. These values can be obtained efficiently as only single hosts are simulated. We use these two values to compute two approximate response-time-throughput models for the behavior of each host for each message type. These models provide lower and upper bounds which we then interpolate. Based on this interpolated response-time-throughput model we select the hardware configuration and obtain a distribution of the workload on the different hosts. Using this hardware configuration we then iteratively modify and simulate the software configuration on the hosts until we achieve the response-time-throughput behavior that has been predicted by using the response-time-throughput model.

4.3. Response-time-throughput model

The purpose of the response-time-throughput model is to estimate the system response time for a single message type without complex simulations. As this cannot be done exactly, a lower and upper bound of the response time is established in a first step. These bounds are based on measurements at low and high system utilizations. Afterwards an approximation function of the response time will be derived from the lower and upper bounds. We will use this function as our response-time-throughput model.

Our system model consists of single server service centers as well as multiple server service centers. A multiple server service center consists of a single message queue and several processors, which can process as many messages in parallel as processors are available. Furthermore we assume a random arrival of messages with an exponentially distributed density, which is a Poisson distribution. The correctness of this assumption has been verified by analyzing the arrivals at a real bank. We also assume an exponential distribution of the response time at each node, with an increasing rate of processing requests. For the model description we use the following notations from [19,20]:

C	Set of all tasks
S	Set of all single server tasks with $S \subseteq C$
M	Set of all multiple server tasks with $M \subseteq C$
$R_k(t)$	Residence time of task $k \in C$ with throughput t
$U_k(t)$	Utilization of task $k \in C$ with throughput t
D_k	Service time of task $k \in C$
$D_{\max}$	Service time of the slowest task with $D_k \leq D_{\max}$ for $k \in C$
T_k	Throughput of task $k \in C$
$A_k(t)$	Average number of messages in queue of task $k \in M$

Furthermore we will make use of the following fundamental law regarding the utilization of a service center with $s \geq 1$ processors:

$$U = \frac{D \cdot T}{s}. \tag{1}$$

The throughput of a service center with $s \geq 1$ processors satisfies $T \leq s/D$. In the saturation state of a service center the throughput $T_{\max}$ is (see [20])

$$T_{\max} = \frac{s}{D}. \tag{2}$$

Complex systems consist of several service centers with different properties. They cannot be modeled by one node as otherwise the estimations differ a lot from the real system. From [20] it is known that the system response time $R(t)$ of a complex system is calculated as the sum of the residence times at all service centers:

$$R(t) = \sum_{k \in C} R_k(t). \tag{3}$$

The residence time $R_k(t)$ is the total time spent by a request at service center k on average. For a single service center the residence time is given in [19,20] as

$$R_k(t) = \frac{D_k}{1 - U_k(t)} = \frac{D_k}{1 - D_k \cdot t}. \tag{4}$$

The calculation of the residence time for a multiple server service center has to distinguish between the queueing and non-queueing state. Hence if more processors are available than jobs ($A_k = 0$), then the residence time is equal to its service time. Otherwise ($A_k > 0$) the jobs have to be queued. The residence time for a multiple server service center has been approximated in [29] as

$$R_k(t) = \frac{D_k \cdot (1 + A_k(t))}{\min(1 + A_k(t) \cdot s)}. \tag{5}$$

The average number of messages in a queue is equal to the time averaged queue length [20]:

$$A_k(t) = t \cdot R_k(t). \tag{6}$$

So (5) can be transformed to

$$R_k(t) = \begin{cases} D_k, & 1 + t \cdot R_k(t) < s, \\ \dfrac{D_k}{s - t \cdot D_k}, & 1 + t \cdot R_k(t) \geq s. \end{cases} \tag{7}$$

In the first case the number of processors is larger than the number of messages. Hence no queueing occurs. In the second case the residence time increases because arriving messages are queued.

4.3.1. Derivation of an upper bound for response time

For deriving the upper bound we use the response time of the system at low utilization $R_0 \approx R(0)$ and the maximal throughput $T_{\max}$. R_0 we obtain by simulating the system with throughput $T \approx 0$. $T_{\max}$ we obtain by simulating the a saturated system and measuring the throughput. At high utilization we know that there exists a service center with $s_{\max}$ processors and $D_{\max}$ service time such that

$$T_{\max} = \frac{s_{\max}}{D_{\max}} \tag{8}$$

and for all other service centers k we have

$$\frac{D_k}{s_k} \leq \frac{D_{\max}}{s_{\max}}. \tag{9}$$

Distinguishing the different types of service centers we obtain from (3) and (7) for the response time

$$R(t) = \sum_{k \in S} \frac{D_k}{1 - t \cdot D_k} + \sum_{k \in M \wedge 1 + t \cdot D_k \geq s_k} \frac{D_k}{s_k - t \cdot D_k} + \sum_{k \in M \wedge 1 + t \cdot D_k < s_k} D_k. \tag{10}$$

Using (8) and (9) we obtain

$$\begin{aligned} R(t) &\leq \frac{1}{1 - t \cdot D_{\max}/s_{\max}} \sum_{k \in S} D_k + \frac{1}{s_k} \frac{1}{1 - t \cdot D_{\max}/s_{\max}} \sum_{k \in M \wedge 1 + t \cdot D_k \geq s_k} D_k \\ &= \frac{1}{1 - t/T_{\max}} \sum_{k \in S} D_k + \frac{1}{s_k} \frac{1}{1 - t/T_{\max}} \sum_{k \in M \wedge 1 + t \cdot D_k \geq s_k} D_k + \sum_{k \in M \wedge 1 + t \cdot D_k < s_k} D_k \\ &\leq \frac{1}{1 - t/T_{\max}} \sum_{k \in C} D_k \leq \frac{1}{1 - t/T_{\max}} R_0 = R_{\text{up}}(t). \end{aligned}$$

This bound has the property that for

$$t \to 0 : R_{\mathrm{up}}(t) \to R(t). \tag{11}$$

4.3.2. Derivation of a lower bound for response time

For deriving the lower bound we base our estimation on the measurement of $T_{\max}$ by simulating the system at utilization $U \approx 1$ and assume again there exists a service center with maximal throughput using $s_{\max}$ processors and having service time $D_{\max}$ as determined by Eq. (8). For the case this service center is a single processor center we have

$$R(t) \geq \frac{D_{\max}}{1 - t \cdot D_{\max}} = \frac{1}{T_{\max}} \cdot \frac{1}{1 - t/T_{\max}}. \tag{12}$$

For the case this is a multiple processor center with t such that $1 + t \cdot D_{\max} \geq s_{\max}$ we have

$$R(t) \geq \frac{D_{\max}}{s_{\max} - t \cdot D_{\max}} = \frac{1}{T_{\max}} \cdot \frac{1}{1 - t/T_{\max}}. \tag{13}$$

For the case this is a multiple processor center with t such that $1 + t \cdot D_{\max} < s_{\max}$ we have

$$R(t) \geq D_{\max} = \frac{s_{\max}}{T_{\max}}. \tag{14}$$

The condition $1 + t \cdot D_{\max} < s_{\max}$ can be transformed to

$$t < \frac{s_{\max} - 1}{s_{\max}} T_{\max}. \tag{15}$$

And thus we get (for an unknown value $s_{\max}$)

$$R_{\mathrm{low}}(t) = \begin{cases} \dfrac{s_{\max}}{T_{\max}} & \text{if } t < \dfrac{s_{\max} - 1}{s_{\max}} T_{\max}, \\ \dfrac{1}{T_{\max}} \cdot \dfrac{1}{1 - t/T_{\max}} = \tilde{R}_{\mathrm{low}}(t) & \text{otherwise.} \end{cases} \tag{16}$$

This bound has the property that for

$$t \to T_{\max} : R_{\mathrm{low}}(t) \to \infty. \tag{17}$$

4.3.3. Derivation of an approximation function

It can be seen from (11) that $R_{\mathrm{up}}(t)$ converges to the measured value R_0 for low throughput values resp. utilization. Vice versa, from (17) we see that $R_{\mathrm{low}}(t)$ converges to ∞ as $t \to T_{\max}$ just as $R(t)$ does. Thus the actual response time is a function that starts at low utilization with values close to $R_{\mathrm{up}}(t)$ and arrives for high utilization at values close to $R_{\mathrm{low}}(t)$. Since $R_{\mathrm{up}}(t)$ and $R_{\mathrm{low}}(t)$ differ considerably, we propose to approximate $R(t)$ by linearly interpolating the two bounds by a function $R_{\mathrm{approx}}(t)$ that matches $R_{\mathrm{up}}(t)$ at low utilization and $R_{\mathrm{low}}(t)$ at high utilization:

$$R_{\mathrm{approx}}(t) \approx \frac{t}{T_{\max}} \cdot \tilde{R}_{\mathrm{low}}(t) + \left(1 - \frac{t}{T_{\max}}\right) \cdot R_{\mathrm{up}}(t) = R_0 + \frac{t}{T_{\max}^2} \cdot \frac{1}{1 - t/T_{\max}}. \tag{18}$$

By this construction $R_{\mathrm{approx}}(t)$ has the property that $R_{\mathrm{approx}}(t) \to R(t)$, both for $t \to 0$ and $t \to T_{\max}$, and $\tilde{R}_{\mathrm{low}}(t) \leq R_{\mathrm{approx}}(t) \leq R_{\mathrm{up}}(t)$. Using $R_{\mathrm{low}}(t)$ instead of $R_{\mathrm{low}}(t)$ can be justified as for small values of t,

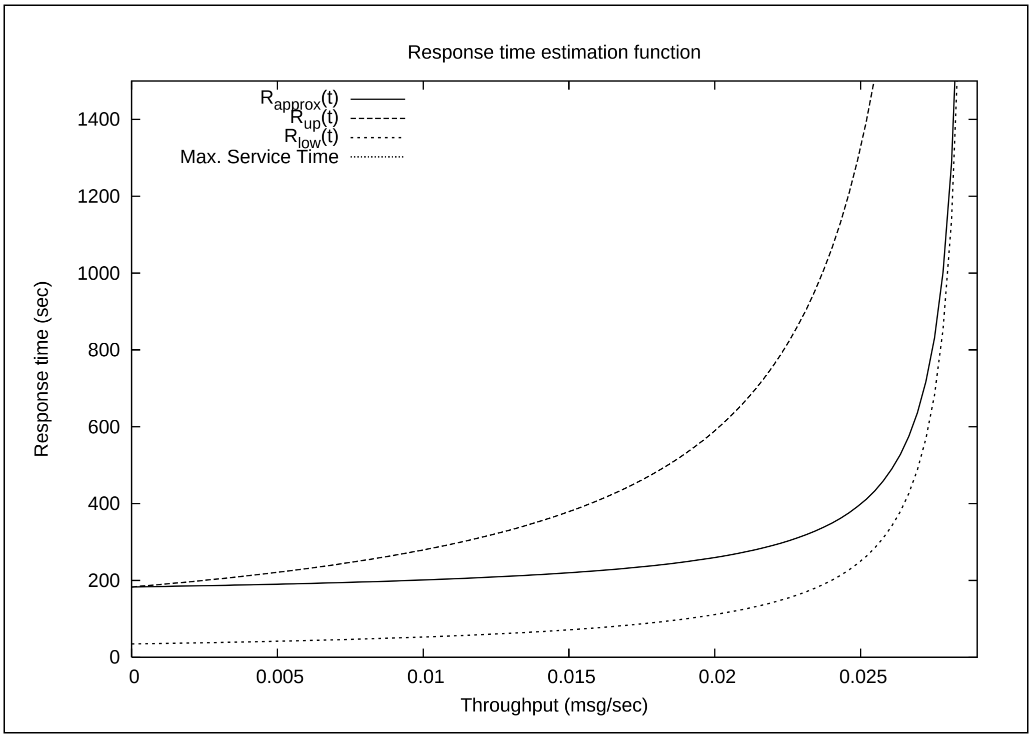

Fig. 8. Response time estimation functions.

$R_{\mathrm{up}}(t)$ is dominating in $R_{\mathrm{approx}}(t)$. As a result we can eliminate the unknown quantity $s_{\max}$. However, we have to check that the property $R_{\mathrm{low}}(t) \leq R_{\mathrm{approx}}(t)$ still holds (which would have been trivial if we used $R_{\mathrm{low}}(t)$ in the definition of $R_{\mathrm{approx}}(t)$). This can be easily verified:

$$R_0 + \frac{t}{T_{\max}^2} \cdot \frac{1}{1 - t/T_{\max}} \geq \frac{s_{\max}}{T_{\max}} = D_{\max}, \tag{19}$$

since $R_0 \geq D_{\max}$.

Fig. 8 shows the functions $R_{\mathrm{up}}(t)$, $\tilde{R}_{\mathrm{low}}(t)$ and $R_{\mathrm{approx}}(t)$. It can be seen that the approximation function $R_{\mathrm{approx}}(t)$ starts at $R_{\mathrm{up}}(t)$ but converges with higher throughput values to $\tilde{R}_{\mathrm{low}}(t)$. Furthermore the maximum service time as defined in (2), which is always lower than $R_{\mathrm{approx}}(t)$ is shown. This is obvious as the response time always includes additionally the queuing time of the service center.

4.4. *Configuration algorithm*

4.4.1. *Configuration of a single host*

For each host of type $h \in H$ and for each message type $m \in MT$ that is processed on this host we have to determine the desired throughput and response time values. First we measure for each message type the values of R_0 and $T_{\max}$ separately. Thus we obtain the approximation function R_{approx} for the response-time function. Then the throughput $T^{h,m}$ corresponding to the required response time $R_{\mathrm{req}}^{\mathrm{m}}$ is determined as

$$R_{\text{approx}}(T^{h,m}) = R^{m}_{\text{req}}. \tag{20}$$

Usually a host h will not devote its whole capacity to a single message type m, but only a fraction $p^{h,m}$. The sum of these values cannot exceed the total processing capacity of the host, thus for a host h

$$\sum_{m\in MT} p^{h,m} \leq 1. \tag{21}$$

4.4.2. Configuration of a distributed system

Now we give an algorithm that determines a host configuration H_s that satisfies the response time and throughput goals for all message types. The response time goals are already taken into account by the choice of the values $T^{h,m}$. Thus in the configuration algorithm we have to select the hosts, such that they provide for each message type m the necessary throughput.

The proposed algorithm is related to the well known bin-packing problem [30,31]. In our case we have to distribute items (=message classes) among a minimum number of bins (=hosts). Both, the capacity of hosts (=size of bins) and the number of hosts are variable. The variable bin packing algorithm as proposed in [32–34] is handling different bin sizes, but the size is fixed after the selection of the bin. The extensible bin-packing algorithm, which allows a stretching of the bin [35,36] is not applicable as the maximum number of bins is fixed. Another difference to standard bin-packing approaches is the necessity to preempt message classes as the capacity of a single host is often too small. Preemption is often used for scheduling algorithms [37,38], but the goal of these algorithms is to distribute items among a fixed set of resources. Our algorithm is based on a *First Fit bin packing* approach with regular recomputation of the host capacities and preemption of the items. The capacity for processing message type m is determined as

$$\textit{capacity}\,(m) = \sum_{h\in H} T^{h,m} \cdot p^{h,m} \geq T^{m}_{\text{req}}. \tag{22}$$

The algorithm is given as follows:

```
$H_s=\emptyset$
For all $m \in MT$
    Distribute the workload for $m$ on hosts with free capacity such that
    $\sum_{m\in MT} p^{h,m} < 1$ and recompute $\textit{capacity}\,(m)$
    While $\textit{capacity}\,(m) < T^{m}_{\text{req}}$
        Add a host $h_a$ to $H_s$ that can satisfy the response time requirement
        Set $p^{h,m} = \min(1, (T^{m}_{\text{req}} - \textit{capacity}\,(m))/T^{h_a,m})$
        Recompute $\textit{capacity}\,(m)$
```

H_s holds the host configuration and is returned as a result. $p^{h,m}$ provides a message distribution that guarantees the throughput and response time goals according to the approximate model of the system behavior. The resulting distribution can be unbalanced since the host selected in the last step can have a very low load.

4.4.3. Balancing the load distribution

In a final step we adapt the distributions $p^{h,m}$ and assign spare capacities $p^{h,m}_{\text{free}}$ such that hosts of the same type are assigned the same capacities and spare capacities are evenly distributed among the different message types. Thus for all hosts h_1 and h_2 and messages $m \in MT$.

$$p^{h_1,m} = p^{h_2,m} \quad \text{and} \quad p^{h_2,m}_{\text{free}} \quad \text{if type}(h_1) = \text{type}(h_2). \tag{23}$$

We distribute the spare capacities such that the relative increase in throughput *increase*

$$\mathit{increase} = \sum_{h\in H} p^{h,m}_{\text{free}}, \quad \text{for all } m \in MT \tag{24}$$

is equal for all message types and maximized. This leads to a linear optimization problem with the additional constraints

$$\sum_{h\in H} p^{h,m} + p^{h,m}_{\text{free}} \le 1, \quad \text{for each } h \in H, \tag{25}$$

$$\mathit{capacity}(m) \ge T^{m}_{\text{req}}, \quad \text{for each } m \in MT \tag{26}$$

and optimization of the value of *increase*. An alternative approach would be to optimize the response time by decreasing the throughput requirements. However, this requires the simultaneous optimization of the distribution and throughput values and thus would lead to a non-linear optimization problem.

4.4.4. Handling of constraints on software configuration

In order to determine the response-time-throughput model it is necessary to determine the maximum throughput of the system at full utilization. In principle, maximum throughput could be obtained from real system measurements. However, in a real system there exist constraints on the software configuration. For example, only one converter instance can be executed on a host. The measurement of such a system will indicate saturation, without having the resources of the system, e.g. CPUs, fully utilized. Thus a software bottleneck exists [11,39]. For correct configuration it is necessary to obtain system throughput with saturated resources. Hence, the software constraints have to be eliminated, which is only possible by simulation of the LQN model.

4.5. Software configuration

In the software configuration phase the number of process instances that are executed at each host in parallel for each task is determined. This step pursues two goals.

The first goal is to avoid suboptimal configurations, where both the response time and the throughput do not achieve the values predicted by $R_{\text{approx}}(t)$. This can occur when a process instance is busy while hardware resources are still available. A *saturated* software instance is not necessarily executing on a processor. It may be waiting for the processor, other hardware devices or for the response of another software instance. Hence such a software instance can be the *bottleneck* of the system if it acts as a server for other software components. If the load is increased beyond the saturation point, no additional useful work is performed [11].

To avoid software bottlenecks the number of process instances executed at each host in parallel must be increased. This strategy increases the throughput but also the response time because more instances share the same resources. Thus, the second goal is to match for each host and message type the

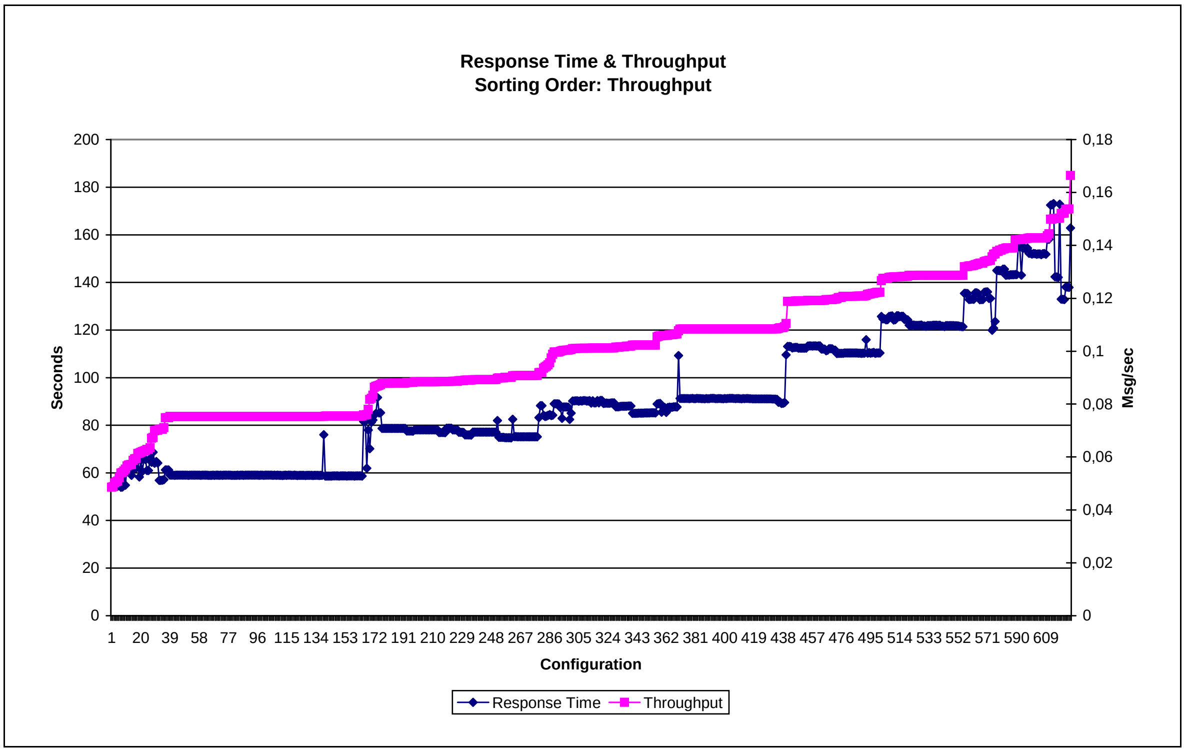

Fig. 9. Throughput and response time for all configurations.

response-time-throughput behavior that has been predicted during the hardware configuration as precisely as possible. For software configuration, the message load distribution on the hosts, that have been determined during hardware configuration, are incorporated into the LQN model. In this way a complete, distributed converter system is modeled.

A simple, but costly, approach to find a system configuration is exhaustive search by simulating all possible software configurations. Though this approach is not practical, for illustration purposes we conducted such an analysis once. The result is shown in Fig. 9. The resulting values for throughput and response time are ordered by throughput. A step corresponds to an increase of the number of software instances at a bottleneck. One can observe the tradeoff between the throughput increase and the corresponding response time increase. It is interesting to note that there exist, in the sense of Pareto optimality, suboptimal configurations.

4.6. *A software configuration algorithm*

A complete search through all possible configurations is not practical if many systems have to be configured since simulations are too time-consuming. Hence a more efficient approach is required, in order to reduce the number of simulations substantially. The proposed method is an improved version of the algorithm presented in [1].

We start with a minimal software configuration and increase the number of processing instances of saturated components until the performance goals are achieved. This still would require many simulations of the complete model. Since hosts of the same type have the same workload $p^{h,m}$ for each message m (see also constraint (23) in Section 4.4) hosts of the same type must have the same software configuration. Thus we configure each host type first individually, and then simulate the complete model to check whether the performance goals have been reached. The benefit of this approach, as compared to [1], is that predominantly single hosts are simulated and the number of simulations of the complete model is strongly reduced.

The throughput goals T^m_{req} and response time goals R^m_{req} are the same as for hardware configuration. The throughput goal for a single host $h \in H$ and a message $m \in MT$ is then $T^m_{\text{req}} \cdot p^{h,m}$. The number of instances of a processing step p on host $h \in H$ is denoted as $S^{h,p}$. The algorithm works as follows:

```
For all host types $h \in H$
  Set all $S^{h,p}$ to 1 and simulate the model
  While not ($capacity\,(m) \geq T^{h,m}_{\text{req}} \cdot p^{h,m}$ and $R^{k,m} < R^m_{\text{req}}$) and $S^{h,p} < S_{\max}$
    Increase each $S^{h,p}$ that has a higher utilization
    than a certain threshold by one.
    Update the model description according to $S^{h,p}$
    and simulate the model.
  Apply the configuration to all hosts of $H$ belonging
  to the same host type.
Simulate the complete model
```

The algorithm may not succeed. In such a case the different design parameters need to be reviewed, as discussed in Section 2.3.

4.7. Implementation

In our configuration method we employ different algorithms, including queuing network simulation and linear optimization. We implemented the hardware and software configuration algorithms in Java, integrating Mathematica [2] and the LQNS solver [3].

The software consists of two components for hardware and software configuration, which are controlled by the *Configuration Workflow Manager*. Thus software configuration can be done independently of the hardware configuration, e.g. in order to reconfigure an existing system if the message distribution is changing.

The architecture including the major data flows is shown in Fig. 10. The whole configuration process is controlled by the *Configuration Workflow Manager*. Its task is to start and monitor the different configuration steps and validate the input files. Monitoring is important since external applications are used that might terminate without notice or turn non-responsive. The process monitoring component is also regularly producing a status file which can be viewed using a WWW browser.

The configuration program uses three XML based parameter files: Hard- and software parameters with configurations goals, an LQN model template and the program configuration. The hard- and software parameters and configuration goals are those described in Section 4.1. The LQN model template file contains an LQN model of a single machine formulated in XML syntax. This template is used later

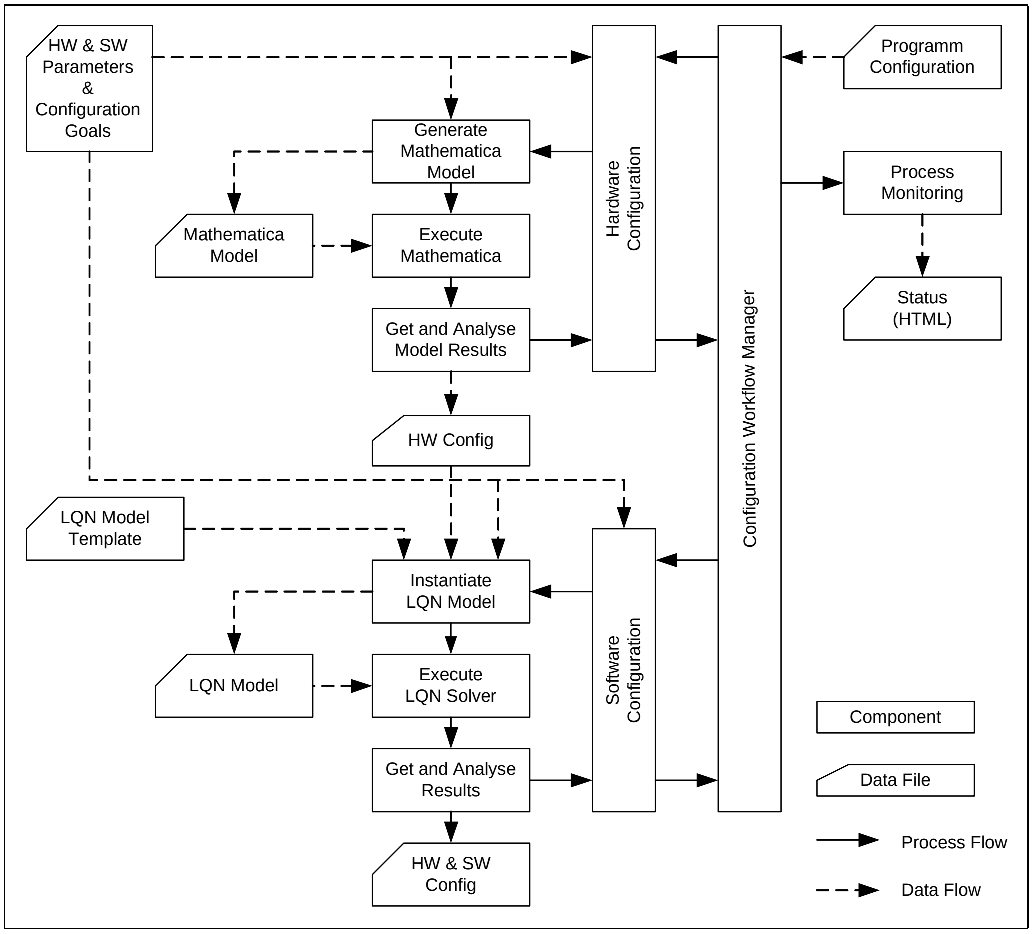

Fig. 10. Architecture.

to generate a complete model of the system. Finally the program configuration contains some general information for the program execution like the working directory, log file location or stylesheets.

An example of part of a parameter configuration file is given in Fig. 11. In this example a message of type ‘small’ is specified by providing the throughput and response time goals (elements `<throughput_goal>` and `<response_time_goal>`). The hardware parameters are given in the `<hardware_parameters>` elements and specify for each host type and message type the maximum throughput and minimum response time. The `<host>` element is used to provide descriptive names for the hosts.

The first step of the configuration workflow is the *Hardware Configuration*. When executing this step a linear optimization problem has to be solved (Section 4.4). We use Mathematica [2] as our solver and generate Mathematica code automatically. The result of this step is an XML file providing the hardware configuration. This file is one of three input files to the software configuration step. The others contain the optimization goals and model parameters and the LQN template of a single host. In the LQN model instantiation step this information is used to automatically instantiate an LQN model of the whole system.

```xml
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE configuration SYSTEM "hwconfig.dtd">
<configuration>
  <message m_id="m_small">
    <description>Small Message</description>
    <throughput_goal>0.72</throughput_goal>
    <response_time_goal>100</response_time_goal>
  </message>
...
  <host h_id="host1">
    <description>Bluenun</description>
  </host>
...
  <hardware_parameters>
    <msg_parameter m_ref="m_small" h_ref="host1">
      <max_throughput>4.465</max_throughput>
      <min_response_time>1.35</min_response_time>
    </msg_parameter>
    <msg_parameter m_ref="m_small" h_ref="host2">
      <max_throughput>1.157</max_throughput>
      <min_response_time>1.87</min_response_time>
    </msg_parameter>
...
  </hardware_parameters>
</configuration>
```

Fig. 11. XML hardware configuration file.

For solving the LQN model the LQN simulator described in [3] is used. If after this step the goals are not reached a new iteration with an updated LQN model will be started (s.a. Section 4.5) till a complete hardware and software configuration is determined.

All intermediary and final results are written to XML files, which can be transformed by using XSLT to any other format. So it is possible to use the results in other programs, e.g. Microsoft Excel, for analysis or for system internal configuration files. A transformation of the result files to HTML for configuration status tracking is also part of the implementation.

5. Application of the configuration method

In this section we study in detail the design of two sample message converter systems as described in Section 2. The first example illustrates our configuration method for a complex system setting. The second example will be used for comparison with a less complex real system that was available for the measurements and will be presented in Section 5.3.

Both systems are based on the same basic hardware components for which system performance measurements have been performed. We consider a simplified message model consisting of two types of

Table 1
LQN model calibration test results

Task	Resource	Processing times (s) (large case)		Processing times (s) (small case)	
		Bluenun	Concorde	Bluenun	Concorde
IPP	CPU	2.69	2.67	1.85	1.84
	File	0.10	0.08	0.11	0.09
	Sum	2.79	2.75	1.69	1.92
Converter	CPU	272.82	270.36	1.40	1.39
	DB	85.82	104.79	0.46	0.56
	File	14.56	11.61	0.29	0.23
	Sum	373.19	386.76	2.15	2.18
Packer	CPU	27.81	27.56	0.23	0.22
	DB	154.81	189.03	0.82	1.00
	File	14.99	11.95	0.09	0.07
	Sum	197.60	228.55	1.14	1.30
Total		573.59	618.05	5.25	5.40

messages: very large and small messages. This is a scenario that is, though simple, fairly realistic for banking applications, where either messages for low-volume transactions, e.g. daily money transfers of a company, or messages for aggregate transactions, e.g. salary transfers for all employees of a company, are occurring. For setting the performance goals we assume that the percentage of very large messages is small, namely 5% of the total number of messages, and the percentage of small messages is large, namely 95% of the total number of messages. A small message consists of 50 financial transactions, while a large message consists of 10 000 financial transactions. In addition it is assumed, that the customer requires a minimum of two machines for ensuring availability. Two different machine types are available for setting up the configuration.

5.1. System model

In order to construct the LQN model we need to determine the performance parameters for all system components. The hardware consisted of two types of machines, two non-equivalent IBM F50 workstations, one with four processors (Type 1, called Bluenun) and one with two processors (Type 2, called Concorde). The relevant hardware components are the CPU, the file system and the database. The relevant software components are the Input Pre-Processor (IPP), the Converter and the Packer. The parameters are obtained by measuring the real system. Each combination of host and message type is measured separately. The results of the measurements are shown in Table 1.

5.2. Example 1

This example is the complex system setting for which we demonstrate the results of the configuration method without comparison to a real-world implementation. In this example the customer expects a throughput of 1 million transactions per hour, independent of message type. The response time should be on average 720 s for a large message and 8 s for a small message.

Table 2
Hardware parameters

Host type	Response time at low utilization (s)		Maximum throughput (messages/s)	
	Small message	Large message	Small message	Large message
Type 1	5.2508	573.6731	1.1721	0.0085
Type 2	5.4018	618.1078	0.5900	0.0067

Table 3
Performance requirements

	Small message	Large message
Throughput	0.4819 (messages/s) = 1735 (messages/h)	0.0254 (messages/s) = 91 (messages/h)
Response time (s)	8	720

5.2.1. Hardware configuration

As a first step, as described in Section 4.2 the response time for each message type at low system utilization and the maximum system throughput are obtained for both host types. The results are in Table 2.

The throughput goal of 1 million transactions per hour, with a distribution of 95% small and 5% large messages, is translated into message-based throughput goals as shown in Table 3. Based on these parameters the first phase of the hardware configuration algorithm leads to an unbalanced distribution of the workloads. Table 4 shows the required hosts, their type, and the workload generated by each message type on each host.

We see that 7 out of 8 hosts are fully utilized. These hosts are only processing large messages. Only host 8 is processing small messages and has spare capacity. In the next step the workloads are balanced. The result of this step is shown in Table 5. The balancing step assigns to all slow hosts (Type 1) some spare capacity of about 15%. Also the message distribution on hosts of the same type is now uniform.

5.2.2. Software configuration

For software configuration we have to build an LQN model for each host type, as well as for the complete system based on the hardware configuration from the previous step. For that purpose we have to specify the calling probabilities for each task (pre-processor, converter, packer) for each message type. The calling probabilities are derived from the workload distribution on the hosts determined in the

Table 4
Unbalanced workload results

	Number of hosts	Host type	Workload per message type	
			Small (%)	Large (%)
Hosts 1–3	3	Type 1	0	100
Hosts 4–7	4	Type 2	0	100
Host 8	1	Type 1	53.88	9.46

Table 5
Balanced workload results

	Number of hosts	Host type	Workload per message type		Throughput (messages/s)	
			Small (%)	Large (%)	Small	Large
Hosts 1–4	4	Type 1	13.47	85.69	0.1347	0.00402
Hosts 5–8	4	Type 2	0	85.59	0	0.00232
Sum					0.5388	0.02536

hardware configuration. The resulting calling probabilities, which correspond to the message distribution among the hosts, are shown in Fig. 12. The graph corresponds to the processing model from Fig. 6. The ‘Input Scanner’ has to process 5% large and 95% small messages. The results are collected by the global scheduler, which distributes them to specific hosts according to the throughput values determined in the hardware configuration. For example, host 1 has to process 24.54% of all messages. This number is determined as the quotient of the total throughput of host 1 and the total system throughput. Furthermore 3.23% of the assigned messages are large, which is the quotient of the throughput of large messages on host 1 and the total throughput of host 1. Hence 96.77% of the assigned messages are small. The calling probabilities for the other host are calculated accordingly.

There exist two software constraints, which influence the configuration result and the performance. The first and more important constraint relates to the fact that the packer process, which collects the results from the converter (see Section 2.2), can only run once per host, since it collects the converter results in a single file. The second constraint relates to the fact that the developers prefer to have a single Input Pre-Processor (IPP) process in the system in order to avoid synchronization problems. The second constraint will be only considered for the complete system configuration in the final step of the software configuration process. When addressing the configuration of individual hosts, it is assumed that each host has its own IPP process.

From the first configuration phase we obtain a single host configuration with two converter processes and a single packer process on each host. Applying this configuration to the complete system, and additionally taking into account the two software constraints previously mentioned, the simulation of the complete system reveals that the single IPP process is the bottleneck. Therefore we simulated a system, where one IPP process runs on each host. Unfortunately the simulation tool could not find a solution for that

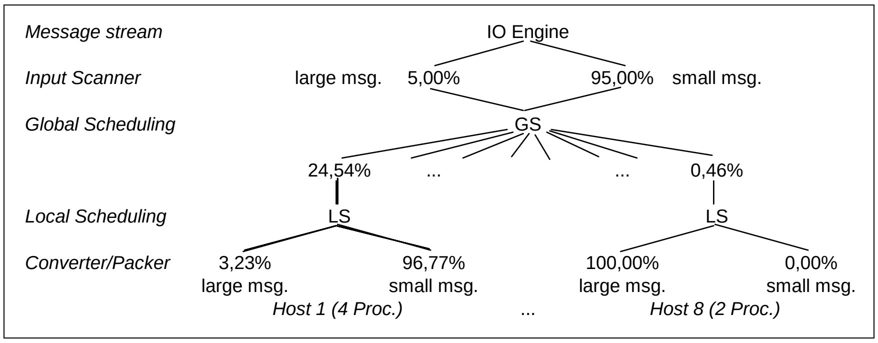

Fig. 12. Message distribution for example 1.

Table 6
Resulting system configuration

	Host type	IPP	Converter	Packer
Hosts 1 and 2	Type 1	1	2	1
Hosts 3 and 4	Type 1	0	2	1
Hosts 5–8	Type 2	0	2	1

Table 7
Performance after the software configuration

Host	Response time (s)		Maximum throughput (messages/s) per host	
	Small message	Large message	Small message	Large message
Hosts 1–4	10.33	578.67	0.10831	0.00362
Hosts 5–8	N/A	619.85	0	0.00209
Sum			0.43324	0.02284

case. The reason is that the algorithm of the LQNS solver does not converge in the rare case that two or more servers of the model saturate at approximately the same rate. As mentioned in [40] further research is necessary to identify the cause of this problem. As an alternative we simulated a system with two IPP processes on two hosts. The simulation results for this system show, that the additional IPP process increases the performance, but the new bottleneck is the packer process. Since having only one packer process on each host is a hard system constraint the performance cannot be increased further.

The resulting system configuration is given in Table 6. The number of converters and packers are the same on each host. Two hosts are configured to have an input scanner. Table 7 gives an overview of the system performance. Note that hosts 5–8 do not process small messages.

5.3. *Example 2*

In this example we describe the design of a real-world financial message converter system, as it was developed in the POEM project. The throughput goal is 250 000 transactions per hour, as compared to 1 million transactions in the previous example. For this system setting we also performed a sensitivity analysis, i.e. we determined to which degree the system tolerates deviations from the message distribution assumed in the configuration process. The results of the performance analysis will be compared to real system measurements in Section 6 using the two IBM AIX machines that were available for the real system tests. The test data was synthetically generated based on confidential statistical information provided by the banks.

5.3.1. *Hardware configuration*

The hardware parameters are the same as in Section 5.2.1, Table 2. The throughput goal of 250 000 transactions per hour with a distribution of 95% small and 5% large messages translates into the message-based throughput goals in Table 8. The first phase of the hardware configuration algorithm results in the unbalanced distribution of the workloads in Table 9.

After load balancing the workload is distributed as shown in Table 10.

Table 8
Performance requirements

	Small message	Large message
Throughput	0.12049 (messages/s) = 433 (messages/h)	0.00634 (messages/s) = 23 (messages/h)
Response time (s)	8	720

Table 9
Unbalanced workload results

	Host type	Workload per message type	
		Small (%)	Large (%)
Host 1	Type 2	33.74	59.97
Host 2	Type 1	0	100

Table 10
Balanced workload results

	Host type	Workload per message type		Throughput (messages/s)	
		Small (%)	Large (%)	Small	Large
Host 1	Type 2	0	100	0	0.00272
Host 2	Type 1	13.47	79.91	0.1205	0.00362
Sum				0.1205	0.00634

5.3.2. *Software configuration*

The calling probabilities required for the LQN model are given in Fig. 13.

The software constraints for the system are the same as in example 1. The software configuration gives the system configuration in Table 11 and the predicted performance is given in Table 12.

The results show that the packer process is, as in the previous example, the bottleneck of the system. As only a single packer process can run on a host, it is fully utilized within this configuration. Hence a further

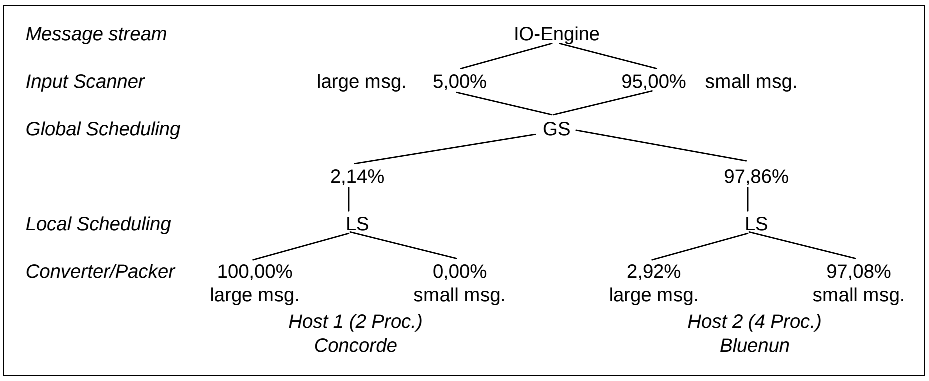

Fig. 13. Message distribution for example 2.

Table 11
Resulting system configuration

	IPP	Converter	Packer
Host 1	0	2	1
Host 2	1	3	1

Table 12
Performance after software configuration

Host	Response time (s)		Maximum throughput (messages/s)	
	Small message	Large message	Small message	Large message
Host 1	N/A	658.41	0	0.00269
Host 2	10.36	578.71	0.11891	0.00358
Sum			0.11891	0.00627

increase of the number of process instances of the other components would result in lower response times, but not in higher throughput values.

5.3.3. *Sensitivity analysis of the model*

An important factor for the system configuration is the assumed message distribution, i.e. 95% small and 5% large messages. This distribution is not static in a real-world application and can vary over time. Thus we analyzed the impact of variations in the message distribution on the configured system. The fraction of small messages was varied from 0 to 100% and, correspondingly, the fraction of large messages was decreased, in order to keep the total number of transactions constant. Note that increasing the fraction of transactions processed in small messages while keeping the total number of transactions constant, will increase the total workload. We were interested in analyzing of how this impacts the system performance.

In the ideal case of a static message distribution, i.e. 95% small and 5% large messages, the global scheduler could distribute the messages statically. For example, all small messages are then distributed to host 1 (‘Concorde’). In order to handle variations in the message distribution, the real system implementation uses an online scheduling algorithm for message distribution. The scheduler decides at message arrival on which host the message will be processed. We were using the modified bin-stretching scheduling algorithm introduced in [4,16]. This algorithm optimizes the distribution without requiring knowledge on future message arrivals. For conducting the sensitivity analysis using our LQN model we had thus to include into the simulation an additional step for modeling the effects of the scheduling algorithm. In this way it was possible to obtain realistic workloads for the hosts under changing message distributions.

The results of the simulation are summarized in Figs. 14 and 15. Fig. 14 shows the sustainable throughput for the individual hosts and the total throughput. The response times are shown in Fig. 15. The figure uses a logarithmic scale, because the response times for each message type are very different. From Fig. 14 we see that the throughput values change only slightly, except for the case with 100% small messages. In this case the number of individual messages that need to be processed increases substantially as compared to the number of transactions. Hence the processing overhead for each message increases on average, which

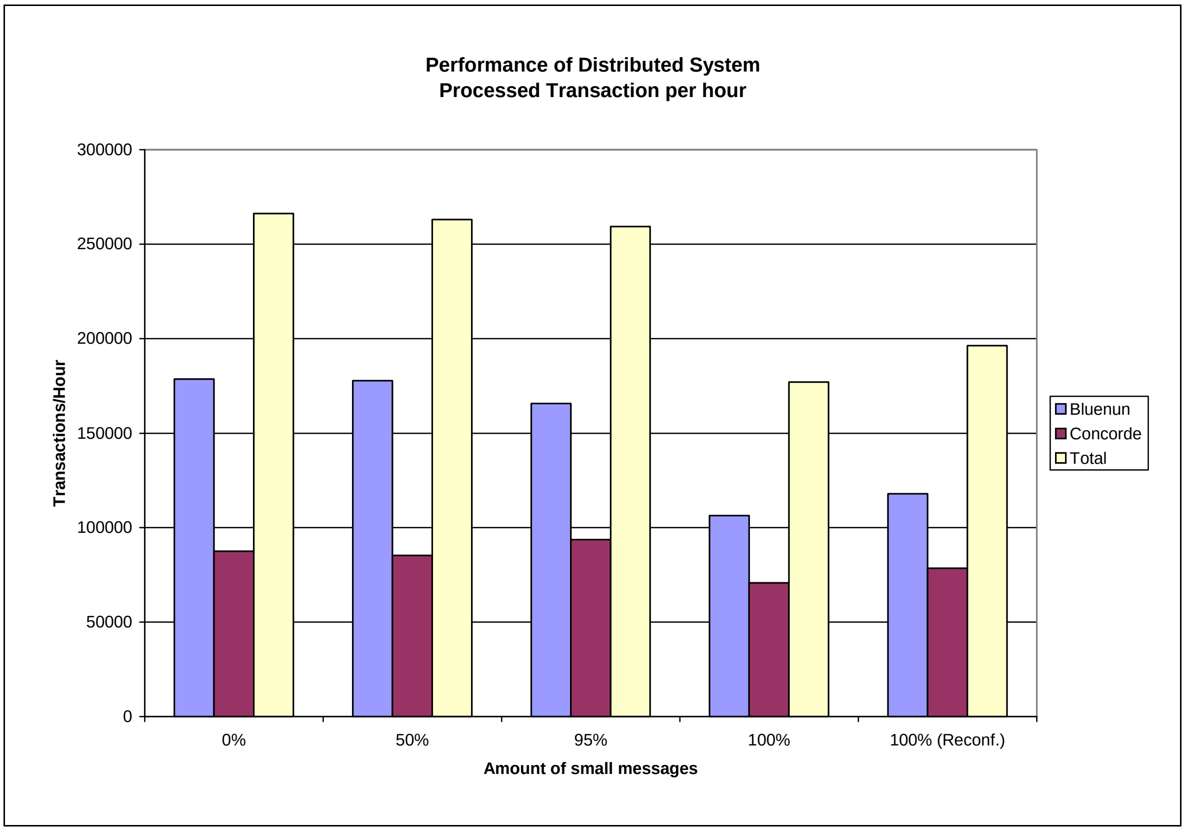

Fig. 14. Performance of the system in processed transactions per hour.

results in a lower total throughput. The corresponding phenomenon can be observed for the response times. The response times for the large messages stay at the same level for all cases on both hosts. The response times for the small messages decrease as the number of small messages increases, since the total load of the system decreases with the lower number of large messages. Hence the waiting times are reduced. But due to the high processing overhead for the small messages and the low number of converters and IPP processes the total throughput decreases for the case of 100% small messages. So if in a real system the message distribution would change to 100% small messages a reconfiguration of the system would be necessary. Such a reconfigured system would have two IPPs on each host. The simulation results are also shown in Figs. 14 and 15 with the label “100% (Reconf.)”. It can be seen that the performance increase is still rather low. The reason is the constraint of having only a single packer instance, which is the bottleneck of the system. For all other message distribution from 0 to 95% small messages only slightly differences exist. Therefore no reconfiguration is needed there.

5.3.4. *Discussion of results*

The hardware configuration algorithm has found for example 2 a configuration of one fast host and one slow host, that can satisfy the performance goals. The actual choice of hosts in the configuration algorithm is done non-deterministically. We assume that it is usually strongly driven by business constraints. Policies

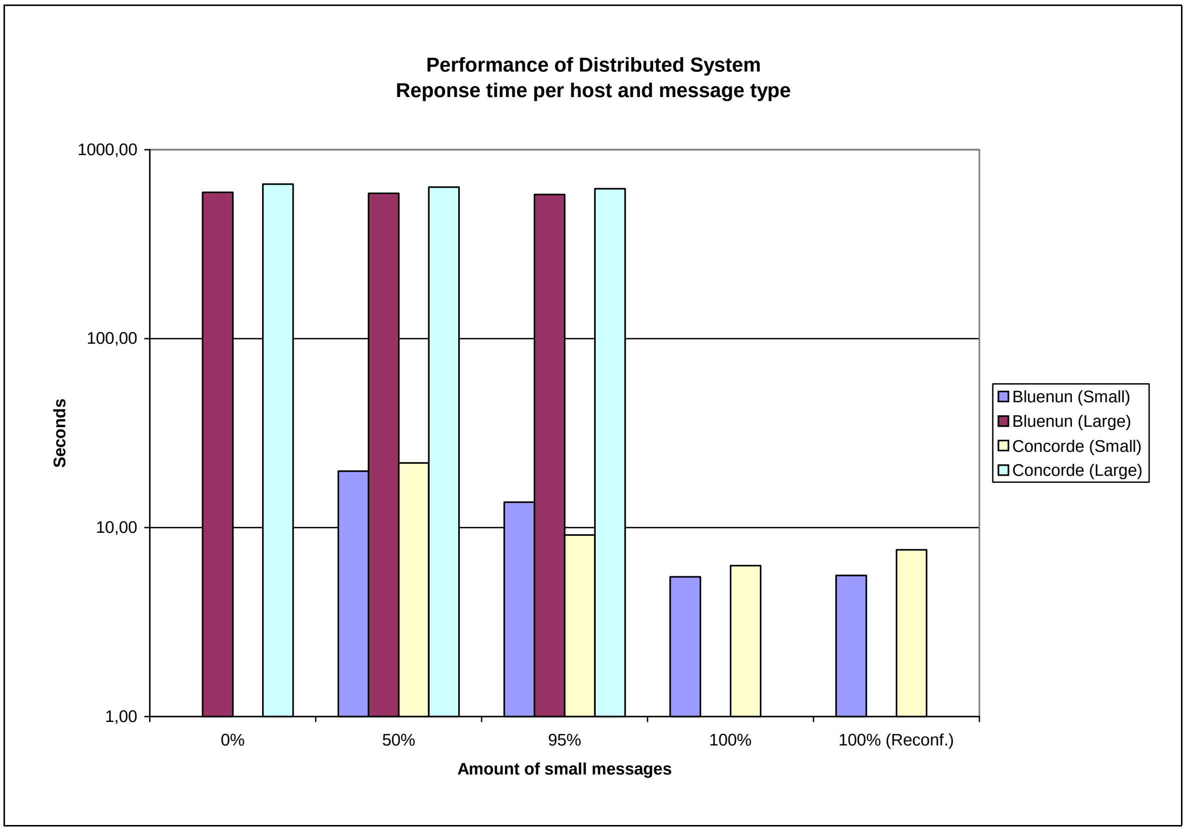

Fig. 15. System response time.

for choosing among possible hardware configurations leave room for further optimizations and are feasible to implement since the hardware configuration algorithm is very efficient.

For hardware configuration we neglected the fact that a global scheduler is required, that has to run on one specific host. This leads to the slight inaccuracy of the performance determined for the fast host. However, this does not critically influence the results, since the resource requirements are extremely low for the global scheduler.

It can be seen that the business goals are not achieved accurately. The average response time for small messages is 25% above the goal for small messages and the response time for large messages lies 19% below the goal for large messages. The other values are extremely close to the goal values. As stated in [20] up to 30% estimation error for the response time and 10% for the throughput are reasonable expectations for the validation of queuing network models results on real systems. Hence we consider this as an acceptable and good result, since also all of the assumptions on the system are approximations of the reality and our goal was to find a reasonable configuration avoiding major flaws.

The sensitivity analysis shows that variations in the message distribution have only slight impact on the performance of the system for our example application since large messages dominate. Hence for variations in the message distributions, which occur in practice, most likely no reconfiguration will be necessary. The only case where problems can occur was the case of 100% small messages. In this special case the processing overhead has a large impact on the performance of the system and a reconfiguration would be necessary. But the software constraints, e.g. the limited number of packers, also limit the possible software configurations. Hence only a slight performance increase was possible for that case.

6. Real system measurements

The predictions were tested using experimental measurements on the "Bluenun + Concorde" system. Tests were carried out to study the following aspects of the system:

- What are the loads on each of the system resources (CPU, Database and File System) represented in the LQN model?
- Is the theoretically best software configuration (including scheduling algorithm and process configuration) actually the best option?
- Where are the bottlenecks in the real experimental system?

The results of these tests are discussed in the following.

6.1. LQN model calibration

The LQN models were calibrated on a distributed workstation system using an instrumented version of the POEM software. The distributed system consisted of the two non-equivalent IBM F50 workstations, one with four processors (Bluenun) and one with two processors (Concorde).

Unfortunately, it was not possible to get direct measurements of the time spent in each process, because the machines were multi-processor systems, so that much of the work was done in parallel. To solve this problem, the software was also run on an old IBM SP2 system comprising single-processor RS/6000 nodes. By comparing the results between these different systems, it was possible to extract machine independent measures of the work done by the POEM software using each primitive resource in each process, and also the speed of that resource on each target machine. These were then combined to get effective times spent using each resource for each of the three processes of the message converter system. The results have been summarized in Table 1. These results were used as parameters for the LQN performance models.

6.2. Scheduler and configuration tests

Experiments were conducted to check that the optimal software configuration predicted by the performance models were actually the best. The software configuration includes both the scheduling method to be used and the process configuration, i.e. the number of IPP and converter processes to be used. The test results for 95% small and 5% large messages are summarized in Table 13.

The scheduling algorithms used are simple strategies like Round Robin, First Come First Serve (FCFS) and a random distribution. Furthermore our modified bin-stretching approach [4] which uses processing time estimation, is also included in the analyses. The initial results from these tests suggested that the FCFS scheduler was the most efficient, which contradicted predictions that the bin-stretching method would be better. However, the bin-stretching algorithm uses an estimate of the runtime of each job on each machine, which is combined with load data to decide which machine should be used for that job.

Table 13
Scheduler test results

Scheduler	Number of converter	Transactions per hour		Speedup
		Bluenun (four CPUs)	Bluenun (four CPUs)+ Concorde (two CPUs)	
N/A	3	173595	N/A	1.00
Round Robin	3	N/A	207707	1.20
First-come, first-served	3	N/A	242989	1.40
Bin-stretching	3	N/A	219700	1.27
Random	3	N/A	239020	1.38
Bin-stretching (modif. load estimates)	3	N/A	267972	1.54

Table 14
Software configuration test results

Bin-stretching scheduler		Transactions per hour	
Small messages (%)	Number of converter	Bluenun (four CPUs)	Bluenun (four CPUs) + Concorde (two CPUs)
0	3	193133	272315
50	3	192885	265891
95	3	173595	267972
100	3	57000	73813
0	4	–	252545
50	4	–	252503
95	4	–	225962
100	4	–	72254

The load estimates initially assumed that the four-CPU Bluenun would complete all jobs twice as quickly as the two-CPU Concorde, but this was too crude because it failed to account for non-CPU loads. The run-time estimates were then re-weighted to take account of the non-CPU loads and the bin-stretching algorithm outperformed the other methods as predicted.

Next, tests were made using 3 or 4 converter processes:

The results in Table 14 show that using three converters is optimal, as suggested by the LQN models. This result is counter-intuitive, since the converter process is the most computationally intensive, and one might expect that four such processes would have made better use of the four CPUs on Bluenun. This result show that a naive adaption of the configuration does not deliver the best result.

6.3. Bottleneck analysis

Next, a simple analysis of system bottlenecks was made, using the results from Table 1 to calculate saturation load conditions. This suggested that the main bottleneck for both Bluenun and Concorde was database access time—with four or even two CPUs, the computational load was not the rate-determining step. The LQN analysis suggests that this effect is most significant in the Packer process, due to the fact that the POEM system can only support a single Packer process on each machine, and so cannot overlap other work with the database access time.

6.4. Discussion of experimental results

Overall, the experimental results support the main conclusions of the LQN analysis. The best software configuration is to have only three converter processes, and the bottleneck is the single Packer process on each machine. The overall throughput as measured is also generally consistent with the LQN results—except for the case where we have 100% small messages, when the LQN model seems to over-predict the throughput.

The experiments were conducted under saturation load conditions with excess input messages. This condition was not covered directly in the LQN model, but represents an asymptotic limit of the LQN model. Large messages are able to saturate the system individually, so we might expect the LQN model to be more accurate when large messages are present, since the behavior then modeled does approach saturation conditions for part of the time. When there are no large messages, the LQN model represents a situation where the system is never saturated, so it is not surprising that the predictions are less good in this situation.

The positive verification of the LQN model results belongs also to the sensitivity analysis. As known from the sensitivity analysis in Section 5.3.3 of the model only slight performance changes exists for all message distributions except for exclusively processing small messages. This can also been seen from the measurements of the real distributed system. Interestingly, for software configurations different from the optimal one the performance depends more sensitively on the message distribution. In the 95% small and 5% large message case the performance of the single Bluenun system as well as for the distributed system with four converter instances is about 10% lower than with a high number of large messages. The reason for this phenomenon for the distributed system is found in the scheduler, which tries to balance the load of the hosts. To this end the scheduler predicts the processing time of a message. This prediction seems to match best with the real processing time in case of an optimal system configuration. For other configurations the predictions differ from the real processing time more widely. Hence the message distribution is less optimal.

The POEM system is intended for use in businesses, where transactions received by a certain deadline are processed by some later deadline. In practice, most customers tend to submit messages just before the deadline for receipt of messages, and the system operates under saturation conditions until these messages are cleared—hopefully before the deadline for processing. The measurements on the real systems show, that, ideally, the LQN models should be extended to provide better estimates for processing small messages for heavily loaded systems, as this will be of interest to operators of message converter systems.

7. Conclusion and outlook

This paper describes a method for the configuration of distributed systems that need to satisfy business-driven performance requirements. Our method allows to dramatically reduce the time needed to find configurations satisfying performance requirements. For example, it was possible to reduce the time needed for configuration of specific problem from 30 h to within minutes, if compared to exhaustive search. Though the method has been introduced in the context of distributed message converter systems we believe it is of more general applicability. The method is organized into a hardware and a software configuration step. For each of both configuration steps algorithms have been developed. These are based on queuing network models to predict the system performance. We have used the method to configure a large distributed system in order to demonstrate the scalability of the method. For a smaller system configuration we compared the predicted results with real system measurements. Furthermore a sensitivity analysis has been done. The verification on the real system shows that the method could be applied successfully to configure a distributed system to reach the maximum performance. Furthermore the performance predictions of the method are confirmed, except for a special case where the workload changes qualitatively (only small messages). So most likely for an optimal configured system a software reconfiguration should not be necessary for most message distributions. Further investigations on the effect of qualitative changes of workload are required. Progress on this question is also an economical issue as experiments in this field are very expensive.

It is obvious—and also shown in the examples—that software constraints can hinder a system to attain the maximum performance. The examples also indicate that an intuitively good configuration, e.g. running four converter processes on a four processor machine, will not necessarily result in the best performance. Hence a configuration method, like the proposed one, can help to select appropriate hardware and software configurations to reach the performance goals.

Nevertheless there are several questions that need to be addressed in the future in order to further develop and refine the method. We have not given any specific strategy of how new hosts are to be added. Here, for example, cost considerations could guide the selection strategy. Alternative methods for load balancing could be considered for optimizing the response time in case of abundant resources. However, if the performance of a system should turn out to be insufficient, incremental changes to existing system configuration can be naturally treated by the method. This could also lead to an automatic software reconfiguration to dynamically react on changes of the message distribution.

The proposed system configuration method appears to be generic enough to be also applied to other types of distributed systems. Thus investigating the wider applicability of our method is a subject for further research.

Acknowledgements

The authors would like to thank the anonymous reviewers and the editor for their valuable and extremely constructive comments on the different revisions of this paper.

References

[1] K. Aberer, T. Risse, A. Wombacher, Configuration of distributed message converter systems using performance modeling, in: Proceedings of 20th International Performance, Computation and Communication Conference (IPCCC'2001), IEEE Computer Society, 2001.

[2] Wolfram Research. http://www.wolfram.com/

[3] G. Franks, A. Hubbard, S. Majumdar, D. Petriu, J. Rolia, C. Woodside, A toolset for performance engineering and software design of client–server systems, Perf. Eval. 24 (1995) 117–135.

[4] T. Risse, A. Wombacher, M. Surridge, S. Taylor, K. Aberer, Online scheduling in distributed message converter systems, in: Proceedings of the 13th Parallel and Distributed Computing and Systems (PDCS'2001), IASTED, 2001.

[5] V.S. Adve, Analyzing the behavior and performance of parallel programs, Technical Report CS-TR-1993-1201, University of Wisconsin, Madison, 1993.

[6] D. Culler, R. Karp, D. Patterson, A. Sahay, K.E. Schauser, E. Santos, R. Subramonian, T. von Eicken, LogP: towards a realistic model of parallel computation, in: Proceedings of the Fourth ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming, ACM Press, New York, 1993, pp. 1–12.

[7] Y. Yan, X. Zhang, Y. Song, An effective and practical performance prediction model for parallel computing on non-dedicated heterogeneous now, J. Parallel Distrib. Comput. 38 (1) (1996) 63–80.

[8] C.M. Woodside, J.E. Neilson, D.C. Petriu, S. Majumdar, The stochastic rendezvous network model for performance of synchronous client–server like distributed software, IEEE Trans. Comput. 44 (1) (1995) 20–34.

[9] D. Menascé, V. Almeida, L. Dowdy, Capacity Planning and Performance Modeling: From Mainframes to Client–Server Systems, Prentice-Hall, Englewood Cliffs, NJ, 1994.

[10] F. Sheikh, M. Woodside, Layered analytic performance modelling of a distributed database system, in: Proceedings of the 17th International Conference on Distributed Computing Systems (17th IDC'97), Los Alamitos, May 1997, IEEE Computer Society, pp. 482–490.

[11] M.L. Fontenot, Software congestion, mobile servers, and the hyperbolic model, IEEE Trans. Software Eng. SE-15 (8) (1989) 947–962.

[12] D. Menascé, H. Gomaa, L. Kerschberg, A performance oriented design methodology for large-scale distributed data intensive information systems, in: International Conference on the Engineering of Complex Computer Systems, Los Alamitos, November 1995, IEEE Computer Society.

[13] H. Gomaa, D. Menascé, L. Kerschberg, A software architectural design method for large-scale distributed information systems, Distrib. Syst. Eng. 3 (3) (1996) 162–172.

[14] H. El-Sayed, D. Cameron, M. Woodside, Automation support for software performance engineering, in: Proceedings of the 2001 ACM SIGMETRICS International Conference on Measurement and Modeling of Computer Systems, ACM Press, New York, pp. 301–311.

[15] United Nations, ISO 9735—Electronic Data Interchange for Administration, Commerce and Transport (EDIFACT)—Application Level Syntax Rules, United Nations, 1997.

[16] T. Risse, A. Wombacher, K. Aberer, Efficient processing of voluminous edi documents, in: Proceedings of the ECIS'2000, Vienna, Australia, Vienna University of Economics and Business Administration, 2000.

[17] G. Franks, S. Majumdar, J. Neilson, D. Petriu, J. Rolia, C. Woodside, Performance analysis of distributed server systems, in: Proceedings of the Sixth International Conference on Software Quality, 1996, pp. 15–26.

[18] L. Kleinrock, Queueing Systems: Theory, vol. 1, Wiley, New York, 1976.

[19] L. Kleinrock, Queueing Systems: Computer Applications, Vol. 2, Wiley, New York, 1976.

[20] E.D. Lazowska, J. Zahorjan, G.S. Graham, K.C. Sevcik, Quantitative System Performance, Prentice-Hall, Englewood Cliffs, NJ, 1984.

[21] C.M. Woodside, E. Neron, E.D.S. Ho, B. Mondoux, An active-server model for the performance of parallel programs written using rendezvous, J. Syst. Software 6 (1986) 125–131.

[22] C.M. Woodside, Throughput calculation for basic stochastic rendezvous networks, Perf. Eval. 9 (2) (1989) 143–160.

[23] D.C. Petriu, Approximate mean value analysis of client–server systems with multi-class requests, in: Proceedings of the 1994 ACM SIGMETRICS Conference on Measurement and Modeling of Computer Systems, ACM Press, New York, 1994, pp. 77–86.

[24] J.A. Rolia, K.C. Sevcik, The method of layers, IEEE Trans. Software Eng. 21 (8) (1995) 689–700.

[25] C. Shousha, D. Petriu, A. Jalnapurkar, K. Ngo, Applying performance modelling to a telecommunication system, in: Proceedings of the First International Workshop on Software and Performance, 1998, pp. 1–6.

[26] E.L. Lawler, D.E. Wood, Branch-and-bound methods: a survey, Oper. Res. 14 (4) (1966) 699–719.

[27] S. Kirkpatrick, C.D. Gelatt, M.P. Vecchi, Optimisation by simulated annealing, Science 220 (1983) 671–680.

[28] D. Goldberg, Genetic Algorithms, Addison Wesley, Reading, 1989.

[29] V.W. Mak, S.F. Lundstrom, Predicting performance of parallel computations, IEEE Trans. Parallel Distrib. Syst. 1 (3) (1990) 257–270.

[30] E.G. Coffman, M.R. Garey, D.S. Johnson, Approximation algorithms for bin packing: a survey, in: D. Hochbaum (Ed.), Approximation Algorithms, PWS, Boston, 1997.

[31] E. Coffman, J. Galambos, S. Martello, D. Vigo, Bin packing approximation algorithms: combinatorial analysis, in: D.-Z. Du, P.M. Pardalos (Eds.), Handbook of Combinatorial Optimization, Kluwer Academic Publishers, Dordrecht, 1999, pp. 151–208.
[32] J. Csirik, An on-line algorithm for variable-sized bin packing, Acta Inform. 26 (8) (1989) 697–709.
[33] F.D. Murgolo, An efficient approximation scheme for variable-sized bin packing, SIAM J. Comput. 16 (1) (1987) 149–161.
[34] D.K. Friesen, M.A. Langston, Variable sized bin packing, SIAM J. Comput. 15 (1986) 222–230.
[35] E.G. Coffman Jr., G.S. Lueker, Approximation algorithms for extensible bin packing, in: Proceedings of the 12th Annual ACM–SIAM Symposium on Discrete Algorithms, Society for Industrial and Applied Mathematics, 2001, pp. 586–588.
[36] Y. Azar, O. Regev, On-line bin-stretching, in: Randomization and Approximation Techniques in Computer Science: Second International Workshop, RANDOM'98, Barcelona, Spain, October 1998, Lecture Notes in Computer Science, vol. 1518, Springer, Heidelberg, 1998, pp. 71–81.
[37] J. Blazewicz, K. Ecker, E. Pesch, G. Schmidt, J. Weglarz, Scheduling Computer and Manufacturing Processes, Springer, Heidelberg, 1996.
[38] P. Brucker, Scheduling Algorithms, Springer, Heidelberg, 1998.
[39] J.E. Neilson, C.M. Woodside, D.C. Petriu, S. Majumdar, Software bottlenecking in client server systems and rendezvous networks, IEEE Trans. Software Eng. 21 (9) (1995) 776–782.
[40] G. Franks, Performance analysis of distributed server systems, Ph.D. Thesis, OCIEE-00-01, Carleton University, 2000.

Thomas Risse is a Division Manager of the data management group at Fraunhofer IPSI. He obtained in 1997 his diploma in computer science at the Technical University of Darmstadt, Germany. In 1998 he joined the research division OASYS at Fraunhofer IPSI in Darmstadt. Since then he worked in several European and industrial projects. His research interests are distributed data management, especially for peer-to-peer environment, performance modeling and scheduling.

Karl Aberer received the PhD degree in mathematics in 1991 from the ETH Zurich. He has been a full professor at EPFL since September 2000, where he heads the Distributed Information Systems Laboratory of the School of Computer and Communications Sciences. His main research interests are in distributed information management, P2P computing, semantic Web, and the self-organization of information systems. From 1991 to 1992, he was a postdoctoral fellow at the International Computer Science Institute (ICSI) at the University of California, Berkeley. In 1992, he joined the Integrated Publication and Information Systems Institute (IPSI) of GMD in Germany, where he became manager of the research division Open Adaptive Information Management Systems in 1996. He has published more than 80 papers on data management on the WWW, database interoperability, query processing, workflow systems, and P2P data management. Recently, he was PC chair of ICDE 2005, DBISP2P 2003, RIDE 2001, DS-9, and ODBASE 2002. He is an associate editor of SIGMOD RECORD and a member of the editorial boards of the VLDB Journal and Web Intelligence and Agent Systems.

Andreas Wombacher graduated at the Technical University Darmstadt, Germany. He has worked for 2 years at IBM on electronic commerce systems before he joined Fraunhofer IPSI. Since then he worked in several European and industrial projects. His research interests are matchmaking of workflows and decentralized workflow management applied to information commerce as well as eServices in particular Web Services.

Mike Surridge is a Operations Director, IT Innovation Centre, Southampton, UK. He holds a PhD in theoretical physics, and has been working in computer science research for 15 years, most of which has been spent at IT Innovation. His main current interests are grid and e-Science-based computing.

Stephen Taylor acquired a degree in electronic engineering in 1993 and a PhD in computer science in 1997. Since 1998 he has worked at the IT Innovation Centre, part of the University of Southampton. His main interests are grid computing and internet security.