Configuration of Distributed Message Converter Systems using Performance Modeling

Karl Aberer
LSIR - Distributed Information Systems Laboratory
EPFL-DSC, CH-1015 Lausanne, Switzerland
karl.aberer@epfl.ch

Thomas Risse, Andreas Wombacher
GMD – IPSI, Integrated Publication and Information Systems Institute, Dolivostraße 15,
64293 Darmstadt, Germany
{risse, wombach}@darmstadt.gmd.de

Abstract

To find 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 tools to analyse 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, using performance estimates a hardware configuration is determined and then the software configuration is incrementally optimized evaluating Layered Queueing 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 increasing importance of B2B interaction.

1. Introduction

Electronic data interchange is an important part of 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 performance of EDI converter systems.

This work is performed within the ESPRIT project POEM (No: 26.356), which is funded by the commission of the European communities. The Partners in POEM are GMD (Germany), MOSAIC SOFTWARE AG (Germany), IT Innovation Center (Great Britain), WestLB (Germany), ABN Amro Bank (Netherlands).

This problem was the motivation to investigate within the POEM (Parallel Processing Of Voluminous EDIFACT Documents) project the question of how performant parallel converter system can be built, based on the typical infrastructures currently available in large enterprises (SMP). Also distributed architectures involving different machine types need to be considered.

A critical problem in achieving this is to identify a hardware and software configuration for such a distributed system, given performance requirements and system constraints derived from the business requirements. For this problem we could not identify an existing systematic approach.

In this paper we present the solution we developed towards the question of how to configure a distributed system such that given throughput and response time goals are satisfied. 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.

For predicting the system performance we use a performance model based on *Layered Queueing Network (LQN) simulation.* Such a model allows to predict the performance of one specific system configuration. Our problem is to select among a large number of possible configurations an appropriate one. Simulating all possible configurations is prohibitively expensive, considering the large number of combinations and the high cost of a single LQN simulation.

To tackle this problem we use the following strategy:

1. *Hardware Configuration*: We make a rough approximation of the performance behavior of single hosts, which can be expressed by a few key parameters that are relatively inexpensive to obtain by single host simulations. Based on this we perform the hardware selection and determine the message distribution on the selected hosts, in a way that theoretically the required performance can be achieved.

2. *Software Configuration*: based on the selected message distribution and hardware configuration an LQN model of the complete system is built and used to determine a software configuration that actually achieves the performance that has been theoretically predicted in the hardware configuration. Since the simulations of the complete model are rather expensive, we use a heuristic algorithm, which tries to minimize the number of required simulations in finding the software configuration.

This heuristic approach to the system design does not necessarily lead to "the" optimal configuration. However since the business requirements (expected messages to process) can also only be approximately given, we only need to find a configuration that is reasonable, i.e. covers the processing requirements approximately at acceptable cost. The different optimization steps serve to avoid obvious flaws in the design, like gross over- or undersizing of the system or occurence of bottlenecks, and to optimize system performance as far as possible once certain design decisions are taken. Also, in the hardware configuration business requirements, like already existing hardware or hardware cost, will often largely influence the decisions.

1.1 Related work

A large body of related work exists in the area of performance prediction for parallel and distributed systems. The prediction methods can be distinguished into two classes: stochastic methods and deterministic methods. Stochastic models take into account the variance of execution times. But as shown in [1] stochastic models have never been evaluated with regard to their accuracy to concrete applications because they require complex solution techniques. In deterministic models the variance of the execution time is assumed to be negligible. In [1] and [2], 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 [21] are based on program execution graphs. Most of the methods are using queueing networks. In [1], [11] and [20] queueing networks are used to model communication networks, parallel or distributed systems. Also database systems have been evaluated in [17] with queueing networks.

Furthermore Fontenot describes in his paper [3] the general problem of software congestion. He noted that software bottlenecks could be avoided by using multiple parallel instances. But no method has been given which finds the necessary number of instances to achieve a specific performance goal.

Generally, most of the existing work is focusing on predicting or evaluating the performance of a concrete system. Devising a method using performance models in order to find 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 give some technical details on the motivating application, which facilitates the understanding of the subsequent presentation. In Section 3 we give some background on LQN models and describe the LQN model that has been built. Section 4 gives the key contribution of this paper, the configuration methods. In Section 5 we describe its application to our business example. Finally Section 6 gives a conclusion and outlook on future work.

2. Application Background

2.1 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 Figure 1.

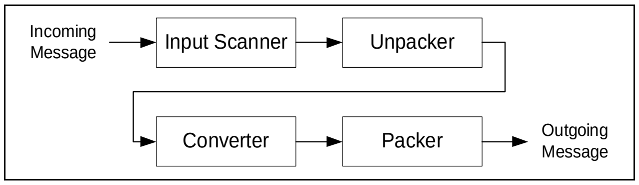

Figure 1: Sequential processing of messages

The 'Input Scanner' regularly checks for the arrival of new messages. Afterwards the 'Unpacker' analyzes the messages regarding the syntax format (e.g. EDIFACT [7], DTA [6]), message structure and size. The information about messages structure and size is used for the distribution of messages among the available hosts doing 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 and availability, and constraints on the type of 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 Figure 2. The *global scheduler* distributes incoming messages to the individual hosts. The distribution strategy it uses will based on the results from 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, the processing steps can be found in [14].

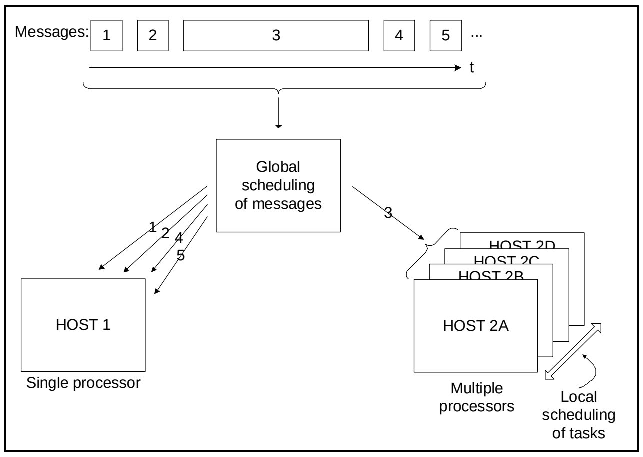

Figure 2: System architecture

3. System Performance Modeling

Response time and throughput are key factors for defining the quality of a system [5]. 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 handle different 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.

Commonly used systems for performance modeling are based on Queueing Networks (QN) [8][9][10]. The standard model of queueing networks is restricted to the modeling of hardware servers. Hence the model was extended by Woodside et al ([18][19]) to Stochastic Rendezvous Networks (SRVN) which allow to model hardware and software servers. SRVNs differ from the classical QNs in two ways. First, each node can act both as client and as server. Second, SRVNs distinguish between two execution phases of a call (Rendezvous) to model synchronous and asynchronous calls.

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 any real system [12][20]. Other approaches approximate the solution by adapting the *Mean Value Analysis* (MVA) of Queueing Networks to SRVNs [12]. The 'Method of Layers' introduced by Riola [15] divides the complete model into several sub models (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 all combined the SRVN and the method of layers to *Layered Queueing Networks* [5]. Further enhancements and generalizations can be found in [20].

3.1 Layered Queueing Networks

The LQN model consists of several components. The core of LQN models are directed acyclic graphs whose nodes are tasks with service entries. A task has one waiting queue and is assigned to one or more processors. This technique is also known as single respectively multiple server [10][11]. Hence the number of entries executed in parallel depends on the number of assigned processors. The arcs between the tasks represent requests from one entry to another. As it is possible that an entry calls several other tasks, each call has a calling probability. The execution of entries is divided into *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

Two characteristics of a real system cannot be modeled directly within LQN, cycles and open arrivals. Cycles can occur often in software systems, 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. Open arrival means that messages arrives from the outside at rate λ.

We use the following transformation rules, as defined in [13] and [16], to transform a system model into a valid LQN model.

- A cyclic relation in a system can be transformed to an acyclic model by the use of semaphores. The idea is to model the read and write access to a source by the usage of a semaphore, which allows an exclusive read or write access to the source by the user of the source. This is shown in Figure 3.

- 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 [16]. 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.

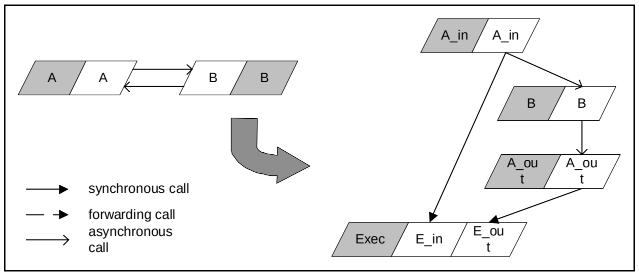

Figure 3: Transformation of cyclic graphs

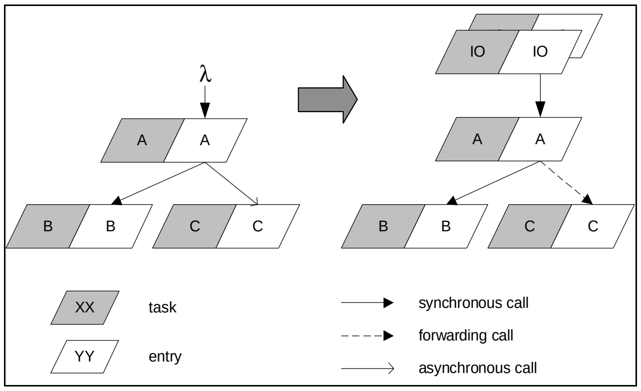

Figure 4: Transformation of open arrivals

3.3 Modelling of a converter system

Within the LQN model of the converter 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 building of the LQN model is based on the data flow within the system. Figure 5 shows the data flow within the converter system. The system has an open arrival IO 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. So 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 Figure 6. To build the complete model for two message clusters all processing steps have to be duplicated. The model does not include cycles and has been transformed into a closed network model.

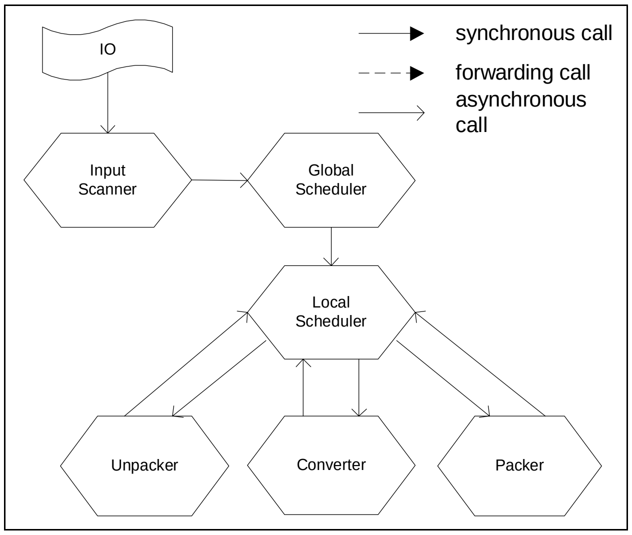

Figure 5: Simple model of the system

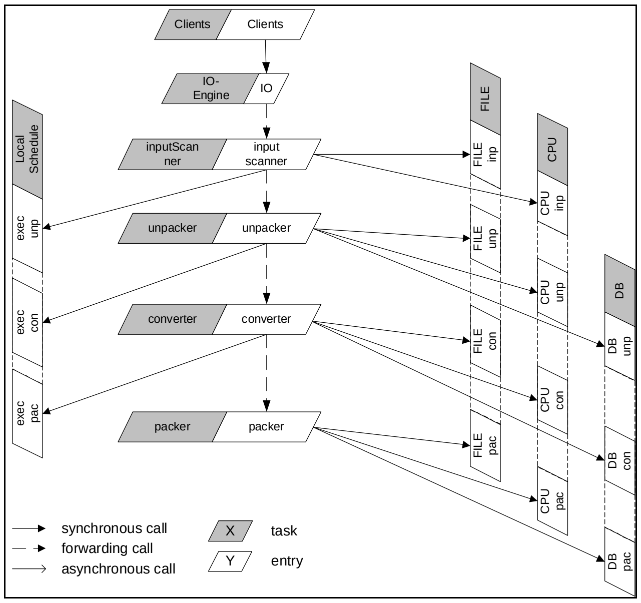

Figure 6: LQN model of one host and one message type

3.4 Simulation tools

As already mentioned several approaches exist for solving a LQN model. The 'TimeBench' tool [4] helps to construct and simulate LQN models and to generate code for different target environments. The LQNS tool (Layered Queueing Network Solver) [4] 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 analyse different system configuration or for sensitivity analysis. All solvers are able to use different methods for solving LQN models, e.g. 'Mean Value Analysis' [12] or 'Method of Layers' [15]. The LQNS is used for our simulations in the example in Section 5.

Even if the solvers approximate the solution the cost of each simulation can vary a lot. The simulation of a single host can be done within a few minutes. The simulation of a complex model with e.g. three hosts as presented in Section 5 can require up to 40 hours. So it is necessary to minimize the number of simulation runs. Hence a system design method has been developed.

3.5 Model parameters

The model parameters are the service times of each entry within a task. The service time represents in the model described in Section 3.3 the total time a processing step uses a resource during the execution. In our case we were using timestamps that were integrated into the program code. These measurements have been performed for each message and host type.

3.6 Evaluation of the LQN model

We have evaluated the performance parameters of the LQN model with the behavior of the real system that is discussed in more detailed in Section 5, 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 model is sufficiently precise and the deviations are within the range of measurement precision.

4. System design method

In this Section we describe the method that we have used in order to configure the hardware and software for a converter system.

4.1 Goal parameters

The goal parameters are derived from the business requirements. Specifically they are

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

Hardware constraints: The hardware configuration is a multi-set *H* containing 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 *minh* and maximal *maxh* number of hosts that are allowed ($minh \leq |H| \leq maxh$). 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 the LQN model that has been described in Section 3, and which is based on model parameters that are derived from the real system. Since the simulation of a complete converter system is fairly expensive, we have to strictly limit the number of simulations that are run during the system configuration process. Thus we proceed as follows.

From the simulation of the system model we determine for each combination of message type $m \in MT$ and for each host type $k \in HT$ approximate values for the minimal response time $R_0^{k,m}$ and the maximal throughput $T_{\max}^{k,m}$ by running two simulations, once with an extremely low utilization and once with extremely high utilization. These values can be obtained efficiently as only single hosts are simulated. We use these two values to approximate the response-time-throughput behavior of each host for each message type. Based on this estimation we select the hardware configuration and obtain a distribution of the workload on the different hosts. Using this hardware configuration we iteratively modify and simulate the software configuration on the hosts until they achieve the response-time-throughput behavior that has been predicted by using the approximation function.

4.3 Response time estimation

Our system model uses single and multiple service centers of queueing type. A multiple service center consists of a single message queue and several processors, which can process as many messages in parallel as processors are available.

We use in the following the definitions from [10] and [11]:

C	:	Set of all tasks

S	: Set of all single server tasks with $S \subseteq C$.
M	: Set of all multiple server tasks with $M \subseteq C$.
$R_k(t)$	: Response 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 $D_k \leq D_{max}$.
T_k	: Throughput of task $k \in C$.
$A_k(t)$	: Average number of messages in queue of task $k \in M$.

$U = D \cdot T$: Utilization law (1)

$T = \frac{m}{D}$ Throughput of a server with m≥1 processors (2)

The response time for single servers is defined as

$$R_k(t) = \frac{D_k}{1-U_k(t)} = \frac{D_k}{1-D_k \cdot t} \tag{3}$$

The response time law for multiple servers is defined in [11] as

$$R_k(t) = \frac{D_k \cdot (1+A_k(t))}{\min(1+A_k(t), m)} \tag{4}$$

The average number of messages in a queue is equal to the time averaged queue length [10]:

$$A_k(t) = t \cdot R_k(t) \tag{5}$$

So (4) can transformed to

$$R_k(t) = \begin{cases} D_k & 1+t \cdot R_k(t) < m \\ \dfrac{D_k}{m - t \cdot D_k} & 1+t \cdot R_k(t) \geq m \end{cases} \tag{6}$$

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

The response time of a system is the sum of the response times of all tasks:

$$R(t) = \sum_{k \in C} R_k(t) \tag{7}$$

Derivation of an upper bound

For the upper bound we use the response time of the system at low utilization, i.e. a measurement $R_0 \approx R(0)$, which can be obtained by simulating the system with throughput $T \approx 0$, as well as the maximal service time D_{max} , which can be obtained by simulating the system at utilization $U \approx 1$.

Distinguishing the different types of service centers we obtain for the response time:

$$R(t) = \sum_{k \in S} \frac{D_k}{1-t \cdot D_k} + \sum_{\substack{k \in M \wedge \\ 1+t \cdot D_k \geq m_k}} \frac{D_k}{m_k - t \cdot D_k} + \sum_{\substack{k \in M \wedge \\ 1+t \cdot D_k < m_k}} D_k \tag{8}$$

Using $D_k \leq D_{max}$ and (2) we obtain:

$$R(t) \leq \frac{1}{1-t \cdot D_{\max}} \sum_{k \in S} D_k + \frac{1}{m_{\max} - t \cdot D_{\max}} \sum_{\substack{k \in M \wedge \\ 1+t \cdot D_k \geq m_k}} D_k + \sum_{\substack{k \in M \wedge \\ 1+t \cdot D_k < m_k}} D_k$$

$$= \frac{1}{1-\frac{t}{T_{\max}}} \sum_{k \in S} D_k + \frac{1}{m_{\max}\left(1-\frac{t}{T_{\max}}\right)} \sum_{\substack{k \in M \wedge \\ 1+t \cdot D_k \geq m_k}} D_k + \sum_{\substack{k \in M \wedge \\ 1+t \cdot D_k < m_k}} D_k$$

$$\leq \frac{1}{1-\frac{t}{T_{\max}}} \sum_{k \in C} D_k \leq \frac{1}{1-\frac{t}{T_{\max}}} R_0 = R_{up}(t) \tag{9}$$

This bound has the property that for

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

Derivation of a lower bound

For the lower bound we estimate the response time of the service center k with the maximal response time D_{max}, which we can obtain by simulating the system at utilization $U \approx 1$. We assume this service center has $m_{\max}$ processors. In the case this is a single processor center we have

$$R(t) \geq \frac{D_{\max}}{1-t \cdot D_{\max}} = \frac{1}{T_{\max}} \cdot \frac{1}{1-\frac{t}{T_{\max}}} \tag{10}$$

In case this is a multiple processor center and we have t such that $1+t \cdot D_{\max} \geq m_{\max}$ we have

$$R(t) \geq \frac{D_{\max}}{m_{\max} - t \cdot D_{\max}} = \frac{1}{T_{\max}} \cdot \frac{1}{1-\frac{t}{T_{\max}}} \tag{11}$$

In case this is a multiple processor center and we have t such that $1+t \cdot D_{\max} < m_{\max}$ we have

$$R(t) \geq D_{\max} = \frac{m_{\max}}{T_{\max}} \tag{12}$$

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

$$t < \frac{(m_{\max}-1)}{m_{\max}} T_{\max} \tag{13}$$

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

$$R_{down}(t) = \begin{cases} \dfrac{m_{\max}}{T_{\max}} & \text{if } t < \dfrac{m_{\max}-1}{m_{\max}} \cdot T_{\max} \\ \dfrac{1}{T_{\max}} \cdot \dfrac{1}{1-\frac{T}{T_{\max}}} = \tilde{R}_{down}(t) & \text{otherwise} \end{cases} \tag{14}$$

This bound has the property that for

$T \rightarrow T_{max}$: $R_{down}(t) \rightarrow R(t)$.

Derivation of an approximation function

In the next step we define an approximation function $R_{approx}(t)$ for the response time such that $R_{down}(t) \leq R_{approx}(t) \leq R_{up}(t)$.

$$\begin{aligned} R_{approx}(t) &\approx \frac{t}{T_{\max}} \cdot \tilde{R}_{down}(t) + (1 - \frac{t}{T_{\max}}) \cdot R_{up}(t) \\ &= R_0 + \frac{t}{T_{\max}^2} \cdot \frac{1}{1 - \frac{t}{T_{\max}}} \end{aligned} \tag{15}$$

By this construction $R_{approx}(t)$ has the property that $R_{approx}(t) \rightarrow R(t)$ for $T \rightarrow 0$ and $T \rightarrow T_{max}$. Using $\tilde{R}_{down}(t)$ instead of $R_{down}(t)$ can be justified as for small values of t $R_{up}(t)$ is dominating in $R_{approx}(t)$. And we get rid of the unknown quantity $m_{\max}$. However, we have to check that the property $R_{down}(t) \leq R_{approx}(t)$ still holds (which would have been trivial if we used $R_{down}(t)$ in the definition of $R_{approx}(t)$). But this also clear because of

$$R_0 + \frac{t}{T_{\max}^2} \cdot \frac{1}{1 - \frac{t}{T_{\max}}} \geq \frac{m_{\max}}{T_{\max}} = D_{\max} \tag{16}$$

since $R_0 > D_{\max}$. The relationship between the different response functions is illustrated in the following figure.

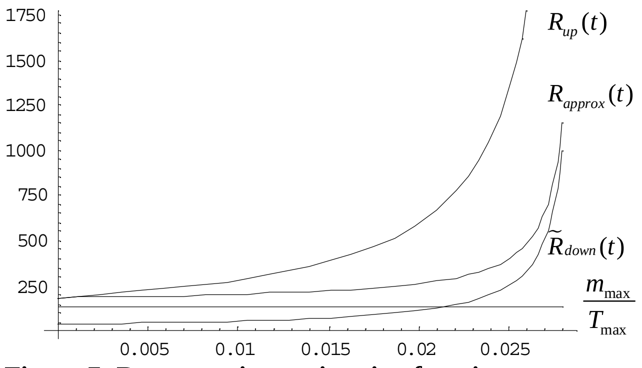

Figure 7: Response time estimation functions

4.4 Configuration algorithm

Configuration of a single host

For each host of type k and for each message type $m \in MT$ that is processed on this host, we have to chose the desired throughput and response time values. Using our approximation function for the response time, we chose values such that an optimal throughput (and thus an optimal utilization) is achieved. Thus the throughput $T^{k,m}$ is chosen such that

$$R_{approx}(T^{k,m}) = R_{req}^m \tag{17}$$

Since usually a host will not devote its whole capacity to a single message only a fraction $p^{k,m}$ of the total processing time will be devoted to a message type. The sum of these values cannot exceed the total processing time of the host, thus

$$\sum_{m \in MT} p^{k,m} \leq 1 \tag{18}$$

Configuration of a distributed system

Next we give an algorithm that creates 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 $T^{k,m}$, thus in the configuration algorithm we have to select the hosts, such that they provide for each message type m sufficient throughput

$$capacity(m) = \sum_{h \in H_s} T^{h,m} \cdot p^{h,m} \geq T_{req}^m \tag{19}$$

The algorithm proceeds as follows:

```
$H_s = \emptyset$
For all $m \in MT$
    distribute the workload on hosts with free capacity
    ($\sum_{m \in MT} p^{k,m} < 1$) and recompute capacity(m)
    While $capacity(m) < T_{req}^m$
        Add a host $h_a$ to $H_s$ that can satisfy the response
        time requirement
        Set $p^{h_a,m} = \min(1, \frac{T_{req}^m - capacity(m)}{T^{h_a,m}})$
        recompute capacity(m)
```

The result of the algorithm in H_s is the required host configuration. In addition $p^{k,m}$ gives a distribution of messages among the hosts that guarantees the throughput and response time goals under the assumed approximate model of the system behavior. The resulting distribution can be unbalanced since the last selected host can have a very low load.

Balance the load distribution

In this last step we redistribute the distributions $p^{k,m}$ and assign spare capacities $p_{free}^{k,m}$ such that hosts of the same type get assigned the same capacities and that the spare capacities are evenly distributed among the different message types, i.e. for all hosts:

$$p^{h_1,m} = p^{h_2,m} \text{ and } p^{h_1,m}_{free} = p^{h_2,m}_{free} \quad \text{if } type(h_1)=type(h_2) \tag{20}$$

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

$$increase = \sum_{h\in H_s} p^{h,m}_{free} \cdot \frac{T^{h,m}}{T^m_{req}}, \text{ for all } m\in \text{MT} \tag{21}$$

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

$$\sum_{m\in MT} (p^{k,m} + p^{k,m}_{free}) \leq 1, \text{ for each } k\in H_s \tag{22}$$

$$capacity(m) \geq T^m_{req}, \text{ for all } m\in MT \tag{23}$$

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

4.5 Software configuration

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

The first goal is to avoid sub optimal configurations, where both the response time and the throughput do not achieve the values that have been predicted in the previous analysis ($R_{approx}(t)$). This can occur since a process instance can be busy while the hardware has still resources 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 addition useful work is achieved [3].

To avoid software bottlenecks the number of instances of each component executed in parallel must be increased. This strategy increases the throughput but on the other hand also the response time because more instances have to share the same resources.

Thus, the second goal is to achieve for each host and message type as precisely as possible the response time-throughput behavior that has been predicted during the hardware configuration.

For the software configuration, the distributions of the message loads on the hosts that have been determined in the hardware configuration are incorporated into the LQN model. In this way the complete, distributed converter system is modelled.

A simple but costly approach to find a system configuration is to simulate all possible software configurations for the extended model and select from those an appropriate one.

Though this approach is not practical for system design, we performed such an analysis once for a converter system for illustration purposes. The result is shown in Figure 8. The measurements values for throughput and response time are ordered by throughput. A step corresponds to an increase of the number of SW instances at a bottleneck. One can also observe the tradeoff between the increase in the throughput and the corresponding increase in response time and that there exist suboptimal configurations.

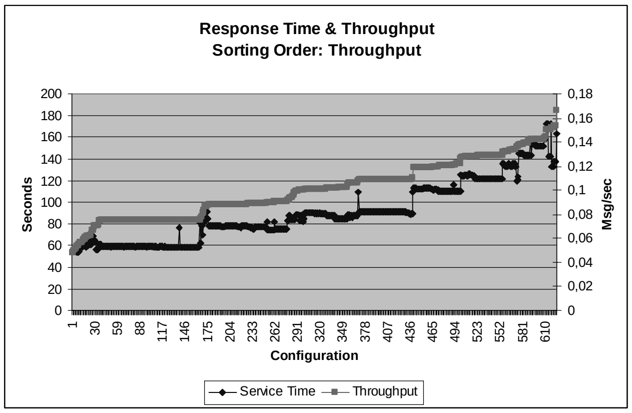

Figure 8: Throughput and response time for all configurations

4.6 A software configuration algorithm

A complete search through all possible configurations is not practical if a lot of systems have to be configured because the simulations are too time-consuming. Hence a more efficient approach is required, which reduces the number of simulations radically. The idea is to start with a minimal software configuration and increase the number of processing instances of the saturated components until the performance goals are reached.

The goals for throughput T^m_{req} and response time R^m_{req} are the same as for the hardware configuration. The number of instances of a processing step p on host $h\in H_s$ can be described as $S^{h,p}$. Informally, then the algorithm is as follows

```
Set all instances of $S^{h,p}$ to 1 and simulate the model
While not($capacity(m) \geq T^{m}_{req}$ and $R^{k,m} < R^{m}_{req}$)
  and $S^{h,p} < S_{\max}$
    Increase each $S^{h,p}$ that has a higher utilization then a certain threshold.
    Update the model description according to $S^{h,p}$ and simulate the model.
```

Of course it is possible that the algorithm does not succeed. In such a case the different design parameters need to be reviewed.

5. Real-World Example

In this example we describe the design of a financial message converter system, which has been introduced in Section 2. The message size distribution is statistically analyzed. We have simplified the real message size distribution. The messages are classified into two message classes: very large and small messages. It is assumed that the percentage of very large messages is small, about 5% of the total number of messages. Thus the percentage of small messages is 95% of the total number of messages. A small message consists of 50 financial transactions, while a large message consists of 10000 financial transactions. The customer expects a throughput of 1.5 million transactions per hour. The response time should be on average 300 sec for large message and 100 sec for small messages.

In addition it is assumed, that the customer prefers to have at least two machines for availability. Possible machine types are predefined.

5.1 System model

Table 1: Results of system measurements

Small message				
	total [s]	FILE abs [s]	DB abs [s]	CPU abs [s]
Input Scan.	0,022	0,017	0,000	0,005
Unpacker	0,125	0,047	0,025	0,053
Converter	0,840	0,095	0,159	0,586
Packer	0,293	0,032	0,013	0,249

Large message				
	total [s]	FILE abs [s]	DB abs [s]	CPU abs [s]
Input Scan.	0,030	0,026	0,000	0,004
Unpacker	1,919	0,576	0,377	0,966
Converter	138,786	13,323	26,254	99,209
Packer	43,778	4,290	1,889	37,599

For the LQN model we need the performance parameters for all system components. The relevant hardware components are the CPU, file I/O and database I/O. The software components are Input Scanner, Unpacker, Converter and Packer. The parameters are measured on a real system for each host and message type. The results of the measurement for the host type we use in the following are shown in Table 1.

5.2 Hardware configuration

After the definition of the model parameters the hardware configuration can be done. The configuration will be done as described in Section 4.2. Hence the first step is to define the input parameters. The input parameters are the response time for each message type at low system utilization and the maximum system throughput. For our example system these parameters can be found in the following tables.

Table 2: Hardware parameters

	Response time in sec		Max. Throughput in msg/sec	
Host type	***Small Message***	***Large Message***	***Small Message***	***Large Message***
Type 1	1.35	195	4.465	0.029
Type 2	1.87	202	1.157	0.0074

The throughput goal of 1.5 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.

Table 3: Optimization goals

	Small Message	Large Message
Throughput	0.72 msg/sec =2592 msg/h	0.037 msg/sec =134 msg/h
Response time (sec)	100 sec	300 sec

Two types of hosts are considered, large hosts with 4 processors (type 1) and small hosts with 1 processor (type 2). Based on these parameters the first phase of the hardware configuration algorithm leads to the following unbalanced distribution of the workloads.

Table 4: Unbalanced workload results

	Host type	Workload per message type	
		Small	Large
Host 1	Type 2	0%	100%
Host 2	Type 1	0%	100%
Host 3	Type 1	16.16%	55.24%

It can be seen that the distribution of messages among the host is unbalanced and host 3 has some reserve capacity. After load balancing the workload is distributed as follows.

Table 5: Balanced workload results

	Host type	Workload per message type		Throughput per message type	
		Small	Large	Small	Large
Host 1	Type 2	62,78%	24,62%	0.72	0.00077
Host 2	Type 1	0%	82,99%	0	0.01811
Host 3	Type 1	0%	82,99%	0	0.01811

5.3 Software configuration

For the software configuration we have to build the LQN model of the complete distributed system for the given hardware configuration. For that purpose we have to define the calling probabilities for each task on a per message basis. This is derived from the workload distribution determined in the hardware configuration. The resulting calling probabilities are shown in the following figure (LS is the local scheduler).

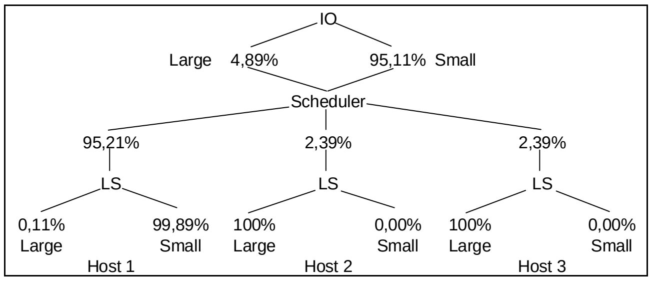

Figure 9: Message distribution of the complete system

Based on this LQN model the software configuration algorithm is executed resulting in the following software configuration and corresponding performance parameters. The gray fields indicate that no value can be given because the host does not process small messages.

Table 6: Resulting system configuration

	Inputscan.	Unpacker	Converter	Packer
Host 1	0	24	24	24
Host 2	15	1	5	2
Host 3	0	1	5	2

Table 7: Performance of the software configuration

	Response time in sec		Max. Throughput in msg/sec	
Host	***Small Message***	***Large Message***	***Small Message***	***Large Message***
Host 1	111.73	253,49	0.7259	0.00079
Host 2		305,32		0.01824
Host 3		305,09		0.01824
Total			**0.7259**	**0.03727**

The software configuration algorithm required 9 days for 25 iterations, each executing one simulation, to find an appropriate configuration. Even if 9 days for 25 simulations seems to be long, it is much faster as compared to 24^{10} simulations required to simulate all possible configurations within a given range of in maximum 24 instances per component! Hence the algorithm makes it possible to configure and measure a complete system within a reasonable amount of time.

5.4 Discussion of results

The hardware configuration algorithm has determined a configuration of two fast hosts and one slow host that can satisfy the performance goals. The actual choice of hosts in the configuration algorithm is in this version done non-deterministically. We assume that it is usually strongly driven by business constraints. Policies 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 the 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 imbalance in the performance of the two large hosts. However, this does not critically influence the results as the resource requirements are extremely low for the global scheduler.

The resulting configuration has a number of desirable and logical properties. Processing all small messages on the small host is advantageous since thus the response time for small messages does not suffer too much from waiting for the large messages. For small messages also the processing requirements for each task are almost the same, thus the even distribution of software instances on the small hosts. For large messages the conversion dominates the processing, thus relatively more instances of this process are used.

It can be seen that the business goals are not exactly achieved. The average response for small messages is even 11% above the goal, while the other values are extremely close to the goal values. We consider this however 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 with no major flaws.

6. Conclusion and outlook

This paper describes a method for the configuration of distributed systems that need to satisfy business-driven performance requirements. Though the method has been introduced in the context of distributed message converter systems, it appears to be generic enough to be also applied to other types of distributed systems.

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 the new hosts are to be added. Here, for example, cost considerations could guide the selection strategy. Alternative methods for load balancing could be considered, optimizing the response time in case of abundant resources. Finally, also the goal parameters will typically not be stable. To that extent an analysis of the sensitivity of the resulting configuration with respect to parameter changes would be desirable. 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.

Within the POEM project the resulting configurations will be tested on real systems. This will provide additional feedback on the quality of the configuration method.

7. References

[1] V. S. Adve, Analyzing the behavior and performance of parallel programs, Ph. D Thesis, University of Wisconsin-Madison, Dec. 1993

[2] D. Culler, R. Karp, D. Patterson, A. Sahay, K. E. Schauser, E. Santos, R. Subramonian and T. Eicken, "LogP:Towards a realistic model of parallel computation", Proc. Fifth ACM SIGPLAN Notices Symposium on Principles and Practices of Parallel Programming, May 1993, pp. 1-12.

[3] M.J. Fontenot; Software Congestion, Mobile Servers, and the Hyperbolic Model; IEEE Trans. Software Eng., Vol. 15, pp. 947--962, 1989

[4] G. Franks, A. Hubbard, S. Majumdar, J. Neilson, D.C. Petriu, J. A. Rolia and C. M. Woodside, "A Toolset for Performance Engineering and Software Design of Client-Server Systems", Performance Evaluation; October 1995

[5] G. Franks, S. Majumdar, j. Neilson, D. Petriu, J. Rolia and M. Woodside; Performance Analysis of Distributed Server Systems; In Proceedings of 6th International Conference on Software Quality; Ottawa; Oct. 1996;pp. 15-26

[6] J. Hergersberg; Bargeldloser Zahlungsverkehr durch Datenaustausch, Deutscher Sparkassenverlag GmbH, 1997

[7] ISO 9735-1, EDIFACT - Application level syntax Rules, ISO, 1997

[8] L. Kleinrock: Queueing Systems - Theory (Vol 1), John Wiley & Sons; 1976

[9] L. Kleinrock: Queueing Systems - Computer Applications (Vol 2), John Wiley & Sons; 1976

[10] E. D. Lazowska, J. Zahorjan, G. S. Graham, K. C. Sevcik; Quantitative System Performance; Prentice-Hall; 1984

[11] V. W. Mak, S. F. Lundstrom; Predicting Performance of Parallel Computations; IEEE Transactions on Parallel and Distributed Systems, Vol. 1, No. 3, July 1990

[12] D. C. Petriu; Approximate Mean Value Analysis of Client-Server Systems with Multi-class Requests; Proc. of the Sigmetrics Conference 1994; ACM Press, New York; 1994

[13] D. C. Petriu and Xin Wang; Deriving Software Performance Models from Architectural Patterns by Graph Transformations; 6th International Workshop on Theory and Applications of Graph Transformation TAGT'98; Paderborn; Germany; 1998

[14] Thomas Risse, Andreas Wombacher and Karl Aberer; Efficient Processing Of Voluminous EDI Documents; Proceedings of ECIS 2000; Vienna; 2000; p. 343-350

[15] J. A. Rolia, K. C. Sevcik; The Method of Layers; IEEE Trans. on Software Eng., Vol. 21, No. 8, pp. 689-700, August 1995

[16] C. Shousha, D. C. Petriu, A. Jalnapurkar, K. Ngo; Applying Performance Modelling to a Telecommunication System; Proc. of the First Int. Workshop on Software and Performance (WOSP'98), pp. 1-6, Santa Fe, New Mexico, October 1998

[17] F. Sheikh and C.M. Woodside, "Layered Analytic Performance Modelling of a Distributed Database System", Proc. 1997 International Conf. on Distributed Computing Systems, May 1997, pp. 482-490.

[18] C. M. Woodside, E. Neron, E.D.S. Ho, and B. Mondoux; An "Active-Server" Model for the Performance of Parallel Programs Written Using Rendezvous; The Journal of Systems and Software. 6, 1 & 2 (May 1986), p. 125-131

[19] C. M. Woodside. Throughput Calculation For Basic Stochastic Rendezvous Networks, Technical Report, Carleton University, Ottawa, Canada, April 1986

[20] C. M. Woodside, J. Neilson, Dorina Petriu, and S. Majumdar, The Stochastic Rendezvous Network Model for Performance of Synchronous Client-Server-like Distributed Software, IEEE Transactions on Computers, Vol. 44, No. 1, January 1995

[21] Y. Yan, X. Zhang, and Y. Song. An effective and practical performance prediction model for parallel computing on non-dedicated heterogeneous NOW. Journal of Parallel and Distributed Computing, 38:63--80, Oct. 1996