Hybrid Performability Analysis of Logistic Networks

Falko Bause, Peter Buchholz, Markus Fischer, Peter Kemper
Informatik IV, Universität Dortmund, D-44221 Dortmund, Germany
{falko.bause, peter.buchholz, markus.fischer, peter.kemper}@udo.edu

Abstract

Resources in large logistic networks are occasionally unavailable or malfunctioning. This implies that perfomability becomes an issue for quantitative analysis of logistic networks. Different time scales between failures and normal operation often justify the decomposition of a performability model into a single availability model that considers failures and recovery of resources and a family of performance models whose individual instances depend on the state of resources. In this paper, we present an approach that simulates a set of performance models independently and in a distributed manner on a network of workstations. We propose to optimize the achievable quality of results for a given total amount of CPU time by minimizing the confidence intervals for performability measures. This is possible by an adaptive assignment of CPU time to simulate those models whose results have the largest impact on the width of confidence intervals.

1 Introduction

Quantitative analysis of a large logistic network is an important and challenging task during its development and operation. Key measures like throughput, sojourn times, cost of operation or usage of resources are usually computed by simulating a model of the system rather than performing experiments with the system itself. Among the many modeling formalisms known for dynamic systems, process-oriented notations are particularly useful in that application area. The ProC/B approach in [1] is one example of such a modeling formalism for logistic networks. The corresponding ProC/B toolset provides a variety of analysis techniques including simulation for the analysis of ProC/B models. Quantitative results of logistic networks or process chains depend on two parts, the load and the available resources to process that load. In practice, the perceived performance varies over time due to failure, malfunction, maintenance, repair and replacement of resources. In computer or communication systems, the concept performability [7] reflects these issues. A transfer of performability methods for computer and communication systems [4] towards logistic networks is not straightforward. The major difference is that the performance part of a performability model for a computer system is often adequately described by a product form queueing network but a model for a logistic network usually requires features that exceed what can be expressed by queueing networks. In consequence, the efficient algorithms employed for queueing network analysis of models of computer systems do not apply and simulation becomes the method of choice. On the other hand, there are similarities as well. For instance, normal operation and failure/repair actions typically perform on different time scales. In addition, the dependancy between performance of a system and availability is often only one-sided, i.e., the performance depends on availability of resources but not vice versa. These two observations lead us to a time scale decomposition approach where the availability part of a performability model is analyzed numerically as a continuous time Markov chain. The results of the availability analysis determine a family of performance models that are subsequently analyzed by simulation. We propose a distributed simulation on a network of workstations and describe how the work load is distributed in order to minimize confidence intervals of reward estimators.

The paper is stuctured as follows. Section 2 briefly introduces performability modeling and analysis of logistic networks. It proposes the decomposition approach and discusses issues of distributed simulation. In Section 3, we describe a case study. A model of warehouse is analyzed and we discuss benefits and limits of our approach.

2 Performability Models

From an application point of view, logistic networks handle orders that are issued by customers. Procurement of orders takes place by a single company or a network of companies that collaborate. A single company can consider its way of handling an order as a service that consists of a number of individual actions or steps that are either taken care of by own units (divisions, departments, groups, individual members of staff) or by some external partners (outsourcing). The notion of a service that is provided someone and requested by someone else serves the purpose of structuring

a complex world. It gives a separation of concerns as well as a separation of resources and resource consumption.

The ProC/B formalism supports hierarchically structured models based on the notion of service in the aforegoing manner. The hierarchy is based on service calls (like function calls in a programming language) and inclusion of resource in the sense that a functional unit provides services whose detailed description resembles actions which in turn are performed as services of a (contained) resource. Basic resources for time consumption (an active resource like a server in a queueing network) and space consumption (a passive resource like a storage area) terminate this recursive description. Special source and sink nodes to produce or terminate service calls at the top level node of the hierarchy provide the load description. This process-oriented way of modeling has gained a lot of attention in logistic systems to identify inefficient organizational structures. The motivation for modeling logistic systems are manifold, for instance, there is interest in the identification of bottlenecks, in technical measures like utilization of resources, production lead times, the quality of service, the ability to meet deadlines. Furthermore, economic measures like an assignment of production costs to individual orders according to resource usage are of interest.

Obviously, the performance of real world systems varies due to failures of resources and delays due to the repair or replacement of faulty resources. This kind of dynamic behavior does not necessarily formalize well as a process. It is rather a behavior that a resource performs internally (and often independently from the state of its environment). Availability of resources is thus often modeled in a formalism like a Markov chain, a stochastic automaton or a GSPN.

If both aspects are present in a single model, we can study the performance degradation of a model due to failure of resources. Since performance and availability imply different types of dynamic behaviour, a model should reveal a structure that allows a partition into two distinct parts, one for performance aspects and one for availability aspects. In our case, this is accomplished by combining two different modeling formalims. In general, the timing behavior of both parts is mutually dependent. Trivially, the performance of a systems depends on the availability of resources. On the other hand, failure rates may depend on the load of a resource, a failure of one resource might cause a simultaneous failure of some other resource and repairs may be done by repairmen which are also shared resources. However, it is often the case, that there is only a one-sided dependancy present in a model, i.e., the performance model depends on the availability model, but not vice versa. This case is particularly advantageous for analysis and will be considered in the following.

We employ the ProC/B formalism to describe the performance part and GSPN for the availability part. It is straightforward to identify the number of available resources from the marking of the GSPN. In our case of a one-sided dependancy, the state of the availability model provides input parameters for the performance model.

2.1 Analysis Based on Decomposition

More formally, we consider a class of models that share state variables which describe the current configuration of resources - and one (the availability) submodel reads and writes those variables, while the other (the performance) submodel only reads that information. In order to distinguish the different degrees in the number of available resources, we define a set $\mathcal{C}$ of configurations, where a single configuration $c \in \mathcal{C}$ describes which resources are available and which are not. For realistic examples with resources of different types, the set $\mathcal{C}$ may contain up to several thousand configurations.

We assume that any measure that is of interest is formally described by a reward R [8] . Rewards can be used to express a wide variety of performance and performability measures including throughputs, sojourn times, rejections probabilities of orders or the filling of stocks. Let $E[R_c]$ be the first moment of the reward value if the performance model is in configuration c. Furthermore let p_c be the probability that the current configuration is c. The value

$$E[R] = \sum_{c \in \mathcal{C}} p_c E[R_c] \tag{1}$$

has to be computed.

The probabilities p_c are computed from the availability submodel. With our assumptions that the failure rates are independent of the state of the performance model, the probabilities can be computed with a high precision by numerical analysis of the Markov chain described by the GSPN.

In addition to this structural property, we observe different time scales for the dynamic behavior of performance and availability submodels. Note that, in a well designed system failure and repair rates are orders of magnitude smaller than service rates at resources. Usually the difference between service rates and failure rates is at least 3 to 4 orders of magnitude. This large difference has two implications. First of all, a model of the complete system, if it exists, cannot be analyzed using standard simulation. Since failures are rare events, a huge number of service events has to be simulated before the first failure occurs. To obtain statistically significant results for the performance of the system including failures and repairs, the number of necessary events would be much too large. However, the large difference between the rates also assures that between failures and repairs the performance model reaches quasi steady state such that independent simulation runs can be performed for the configurations of the performance model.

A time-scale decomposition of performability models is often considered in literature, however simulation is hardly used to compute an estimate of $E[R_c]$ for individual configurations c. It is more common to consider queueing networks for the performance submodel and perform product-form analysis for the price of hard restrictions on what can be expressed in the performance part. For modeling logistic networks, queueing networks are often too restricted to be of general use. Thus, the challenge of the simulative performance analysis is to analyze a large number of different performance models such that the confidence interval for $E[R]$ is sufficiently small. Before we consider this issue in depth, we discuss a side issue that is also of practical relevance. For many systems, a minimal number of resources must be available to be operational. Since the configurations determine the number of available resources, we need to ensure that only configurations that correspond to well-defined models are considered for simulative analysis. This is not trivial for complex models. It may require a functional pre-analysis of the configurations. Since the ProC/B toolset [1] allows a transformation of a performance model into a Petri Net (PN), we compose a PN that results from the performance model with the GSPN of the availability model and subsequently apply analysis techniques for PNs on the resulting net. Such functional techniques include model checking or invariant analysis. Currently, the composition of both PNs is performed manually useing features of the APNN toolbox [3] to manipulate hierarchical and compositional models. Let $\mathcal{C}^o \subseteq \mathcal{C}$ be the set of operational configurations among the set of configurations.

Performance models that are non-stationary form a second class of malfunctioning models. If a model is non-stationary, our assumption that the performance model reaches quasi steady state in each configuration is no longer justified. Since the detection of non-stationary behavior is not trivial, we assume in the sequel that all configurations of the performance model are stationary. This can always be achieved by limiting the number of processes in the system, as it is also done in practice whenever the number of arrivals exceeds the capacity of a logistic network.

We do not simulate a non-operational configuration c, nevertheless we need $E[R_c]$ in (1). If the requested reward describes the throughput of the system, then the reward in a non-operational configuration becomes 0. The determination of rewards that describe sojourn times or rejection probabilities is more complex. For those cases, we assume that no service occurs in a non-operational configuration and that all arrivals are rejected during a non-operational period. The latter assumption will only approximately match a real situation. It is justified, if the duration of a non-operational configuration is much longer than the interarrival time of new processes and the capacity is not too large. Let d_c be the mean duration of configuration $c \in \mathcal{C} \setminus \mathcal{C}^o$, which can be computed from the results of the numerical analysis of the availability model, and let λ_k be the mean arrival rate of processes of type k. If R describes the rejection rate of type k processes, then $R_c = d_c \cdot \lambda_k$ and if R describes the sojourn time, then $R_c = d_c$ because processes in the system have to wait the whole period. In this way rewards can be assigned to non-operational configurations. The assigned rewards are only approximations, but the approximation error tends to 0 for an increasing difference between service rates and failure/repair rates.

As sketched so far, the proposed approach can be considered as the solution of a Markov reward model where rewards of individual states are determined by simulation. This raises a number of interesting observations. For a given reward evaluation, we may want to design a system in a way that $E[R]$ obtains some optimal value which yields an optimization problem in order to change parameters of the availability model in a way that distribution of p_c over $\mathcal{C}$ increases (or decreases) the value of $E[R]$. As long as we assume that the behavior of the availability model is not directly influenced by the state of the performance model, this optimization can be done by modifying the availability model only and reusing the simulation results for the performance models. On the other hand, for a given $c \in \mathcal{C}$, we can adjust the quality of the simulation according to the relative importance of R_c for $E[R]$ as indicated by the value of p_c. The potential of this observation is considered in the following for distributed simulation on a workstation cluster.

2.2 Distributed Simulation

Simulation is used to determine an estimate $\bar{R}(n)$ for $E[R]$ where n denotes the number of sample points used for the estimate. Let S_n^2 be the sample variance of the estimator, then a confidence interval can be computed by

$$\bar{R}(n) \pm z_{1-\alpha/2}\sqrt{\frac{S_n^2}{n}}$$

for an n that is sufficiently large and where $z_{1-\alpha/2}$ is the $1-\alpha/2$ quantile of the normal distribution [6].

In our case, $E[R]$ is given as a weighted sum of $E[R_c]$ such that

$$\bar{R}(n) = \sum_{c \in \mathcal{C}^o} p_c \bar{R}_c(n) + \sum_{c \in \mathcal{C} \setminus \mathcal{C}^o} p_c R_c \; .$$

If the availability model is analyzed numerically, then the values for p_c are exact up to the numerical precision of the machine. Additionally, the values R_c are fixed for $c \in \mathcal{C} \setminus \mathcal{C}^o$. Thus, the variance of the estimator $\bar{R}(n)$ results from the variance of $\bar{R}_c(n)$ for $c \in \mathcal{C}^o$. If we assume that the sample values to determine $\bar{R}_c(n)$ are all independent we obtain for the sample variance of the estimator $\bar{R}(n)$

$$\bar{S}^2 = \sum_{c \in \mathcal{C}^o} p_c^2 \frac{S_c^2(n_c)}{n_c} \qquad (2)$$

where n_c is the number of sample values to determine $\bar{R}_c(n)$ and $S_c^2(n_c)$ is the estimator for the variance of R_c. The relation holds because for independent random variables X_i with variance $S^2(X_i)$, $S^2(\sum X_i) = \sum S^2(X_i)$ and $S^2(cX_i) = c^2 S^2(X_i)$ [6]. The confidence interval for $\bar{S}^2$ is computed as

$$\bar{R}(n) \pm z_{1-\alpha/2}\sqrt{\bar{S}^2} \; .$$

Since the simulation models are independent for different configurations of the performance model, an approach based on independent replications can be used [6] for the estimation of $\bar{R}_c(n)$. Different simulation runs can be distributed on different machines or processors and can be performed in parallel. We apply the approach in a workstation network and assume that M workstations are available. For simplification we assume that all workstations are identical, but the following approach can be easily extended to non-homogeneous machines. We further assume that we can start and stop a simulation run on some processor and obtain as a result of some run i for configuration c the values $n_{c,i}$, $R_{c,i}$ and $S_{c,i}^2(n_{c,i})$ for the number of observations, the estimated mean and variance, respectively. The variance $S_{c,i}^2(n_{c,i})$ can be estimated using some appropriate method for dependent observations like batch means, regenerative simulation or an autoregressive method [6].

We have defined a method to compute $\bar{R}(n)$ and its confidence interval from a simulation using M processors. We further assume that runs are performed in phases of length Δ. At the beginning of each phase on each workstation a simulation run is started which runs for Δ times units and produces results of the above form. We assume that T phases are simulated such that overall MT simulations runs are made. A schedule describes the assignment of simulation models to processors in the different phases. Since we assume identical processors, we only have to define which configurations are simulated in a phase and not the assignment of models to processors. Phases are numbered from 1 through T and $\mathcal{S}_t$ is the multiset of configurations which are simulated during phase t on some processor. $\mathcal{S}_t$ contains M elements, one model for each processor, and may contain the same elements several times since one configuration can be simulated on different processors in the same phase. A schedule is defined as

$$\mathcal{S} = \cup_{t=1}^{T} \mathcal{S}_t \; .$$

Let $occ_c(\mathcal{S})$ be the number of occurrence of c in $\mathcal{S}$ which describes the number of phases of length Δ in which configuration c has been simulated. Let $\bar{S}_c^2(a_c)$ be the estimator for the variance of $\bar{R}_c(n)$ if configuration c has been simulated for a_c intervals of length Δ. With this formalization we can define the optimization problem

$$\min_{\mathcal{S}} \left(\sum_{c \in \mathcal{C}^o} p_c^2 S(occ_c(\mathcal{S})) \right)$$

to find a schedule with a minimal variance of the estimator for $\bar{R}(n)$ and therefore also with the smallest confidence interval.

Of course, the computation of the optimal schedule requires knowledge of S_c^2 which is not available. Consequently, a schedule $\mathcal{S}_{t+1}$ is computed using the information which is available from the phases 1 through t. Let $a_c(t) = occ_c(\cup_{s=1}^{t} \mathcal{S}_s)$ and $\sigma_c(a_c(t)) = \sqrt{(\bar{S}_c^2(a_c(t))/a_c(t))}$. For the moment we assume $a_c(t) > 0$ for all $c \in \mathcal{C}^o$. Since $\bar{S}_c^2(a_c(t))$ is an unbiased estimator for S_c^2, $\sigma_c(t)$ is proportional to the width of the confidence interval for $\bar{R}_c(n)$ after t phases.

The gain of performing in phase $t+1$, i parallel runs of configuration c is proportional to the estimated reduction of the confidence interval width due to these simulation runs and is given by

$$g_c^i(t+1) = p_c^2 \left(\sigma_c(a_c(t)+i) - \sigma_c(a_c(t)) \right) \; .$$

Since we do not know $\sigma_c(a_c(t)+i)$ a priori we assume

$$S_c^2 = a_c(t) \left(\sigma_c(a_c(t)) \right)^2 \; \Rightarrow \sigma_c(a_c(t)+i) = \sqrt{\frac{S_c^2}{(a_c(t)+i)}}$$

and

$$g_c^i(t+1) = p_c^2 \sigma_c(a_c(t)) \left(\sqrt{\frac{a_c(t)}{a_c(t)+i}} - 1 \right) \; .$$

The following algorithm computes $\mathcal{S}_t$ from the available gains $g_c^i(t)$.

1. $\mathcal{S}_t = \emptyset$;
2. for ($m = 1$ to M) do
3. $\quad c = \min_c \left(g_c^{occ_c(\mathcal{S}_t)+1}(t) - g_c^{occ_c(\mathcal{S}_t)}(t) \right)$;
4. $\quad \mathcal{S}_t = \mathcal{S}_t \cup \{c\}$;

The only remaining point is the estimation of the initial values for $\sigma_c(.)$. One possibility is to run each configuration at least once at the beginning to obtain some result and start afterwards with the algorithm. If the number of configurations is huge, only some configurations are simulated and other configurations receive values of similar configurations which have been simulated. However, the latter approach introduces an additional approximation and usually requires the use of some form of a metamodel which goes beyond the goal of the current paper.

2.3 Tool Support

Performability analysis of realistic models of logistic networks requires adequate tool support. We propose the ProC/B toolset that provides a graphical user interface for the ProC/B formalism. Models are either mapped onto HIT-models [2] for a simulative analysis or onto Generalized Stochastic Petri Nets (GSPNs) for functional analysis based on invariant analysis or model checking, numerical analysis of the associated CTMC or simulation. HIT-models are supported by the tool HIT [2]; GSPN models are supported by the APNN toolbox [3]. The numerical analysis of the availability model is limited by the size of the resulting Markov chain; currently models in the order of $10^6 - 10^7$ states can be analyzed in the APNN toolbox.

3 Case Study: Warehouse

In this section, we analyze a model for the management of a warehouse at a manufacturing site for cars. The model contains three processes that describe the operational activities. The first process, denoted as `Truck_Arrival`, describes the supply and storage of sub-assemblies (engines etc.) and the return of buffered empties (boxes). The second process, labeled `In-Plant_Transport`, puts sub-assemblies (including their packages) on-demand into a second intermediate buffer that serves the actual production line. Finally, the third process, labeled `Demand`, describes the demand of the production line and moves freed shipping boxes back into the buffers. Since involved buffers have a finite capacity, certain activities remain blocked until sufficient inventory or free space becomes available. Fork-lifts access buffers for loading and unloading operations. They are statically assigned to specific buffers and exclusively-used to manipulate their inventory. Furthermore, fork-lifts are used to load and unload trucks that deliver sub-assemblies or take empties. Since the number of fork-lifts is limited (2 per buffer and 21 for trucks), processes may experience queuing and additional waiting times due to non-availability of requested fork-lifts. Fork-lifts occasionally fail and albeit the system remains operational, its overall performance degrades. Since failures happen rarely, the maintenance policy is as follows. A single worker is responsible for repairing fork-lifts on-site in case of a breakdown. In addition, an external company sends additional staff to perform regular maintenance checks in fixed time intervals to keep the number of breakdowns at an acceptable level. A maintenance check implies that all operational fork-lifts are not available for a short period of time, the maintenance interval.

In order to evaluate the impact of failures on performance measures we have created two models, a dependability model that captures the failure, maintenance and repair behavior of involved fork-lifts and a performance model that captures the three processes that the normal operation of the warehouse and measures lead times of the involved processes. Lead times are typically used to evaluate the performance of the system. Application experts are interested in the determination of optimal maintenance frequencies. Notice, exorbitant maintenance effort would prevent from failures (and non-availability), but this is at the expense of more frequent non-availability caused by maintenance interrupts. In the system, the time to failure of maintained fork-lifts is assumed to be the same as the time to failure of repaired fork-lifts.

In what follows, we briefly introduce a ProC/B performance model and a GSPN dependability model, cf. Fig. 1 and 2. Due to space limitation, we highlight only certain parts of the performance model and restrict ourselves to the `Truck_Arrival` process. The upper part of Figure 1 (separated by a dashed line) shows a flow-chart of activities (arrow-like hexagons) that make up the process of incoming trucks and their handling at the warehouse. The lower part gives the resources that are demanded by activities for their execution. Incoming trucks first go to an entrance office and experience a simple delay, then they register at this office. After registration, a specific storage area is selected (timeless activity) and the process forks into two subprocesses modeling the arrival of trucks (upper chain) and the arrival of requested fork-lifts (lower chain) at the loading ramp. Afterwards, the process performs unloading activities (represented by 5 arrow-like hexagons) and reaches a boolean branch. It is assumed, that only certain trucks carry sub-assemblies that need to be buffered. Therefore, in the upper branch the process puts the freight into the main store, proceeds to the empties store and unloads empties from the empties store by calling the service `unload`, whereas in the lower branch, the process immediately reaches a sink for finalization. Notice, the model is hierarchically structured and the description of services in terms of involved activities and demanded resources is delegated to the submodel and invisible from the calling activity denoted `Unload_Empties`.

We proceed in this section with a description of the GSPN model capturing the dependability aspects of the system under investigation. In our model we assume that the time to failure, the repair time and the maintenance time are all Erlang-2 distributed. Figure 2 shows a colored GSPN model of the described scenario. It consists of three parts:

- the upper part (subnet $p_7, \ldots, p_{10}, t_6, \ldots, t_8$) describes the maintenance process of all operational fork-lifts,
- the middle part (dotted box, subnet $p_{11}, \ldots, p_{13}, t_9, \ldots, t_{13}$) models the elapsed time between maintenance intervals, and
- the lower part (subnet $p_1, \ldots, p_6, t_1, \ldots, t_5$) describes operational fork-lifts, being subject to failures, and repair.

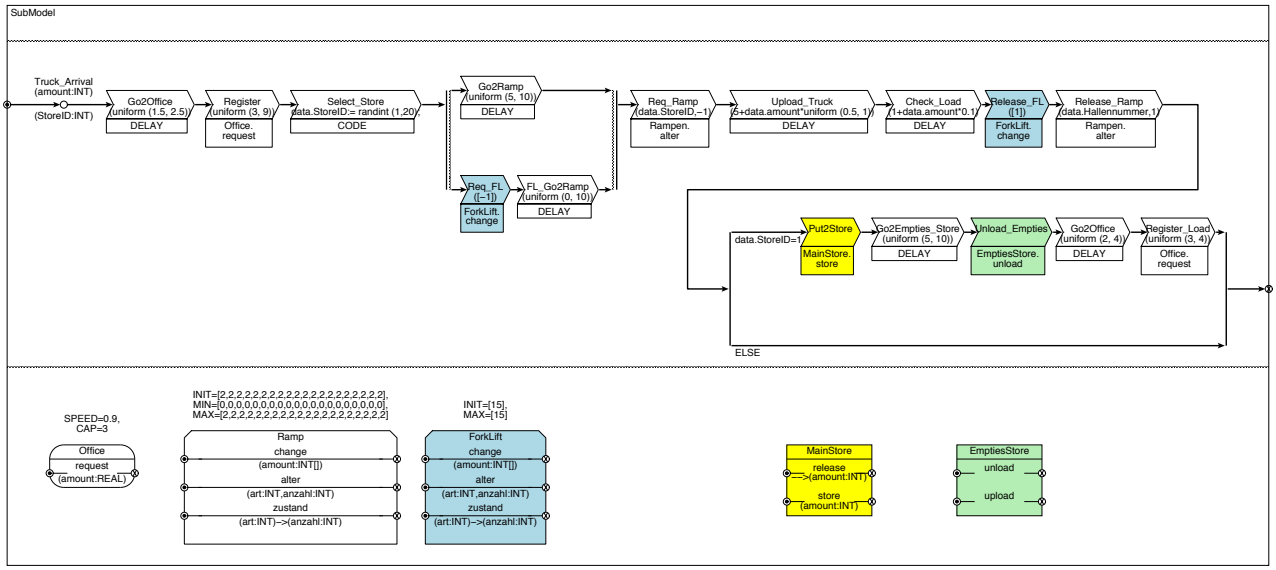

Figure 1. ProC/B performance model: `Truck_Arrival` process

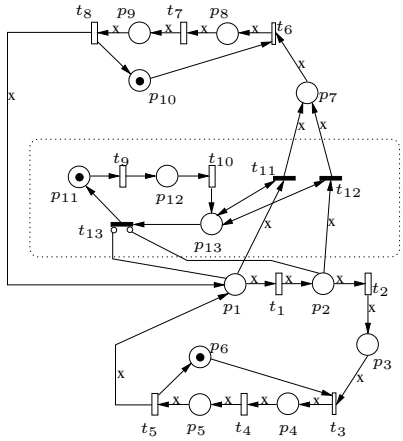

Figure 2. GSPN Dependability Model

Places $p_1, \ldots, p_5, p_7, \ldots p_9$ are colored places representing the four different types of fork-lifts and similarly transitions $t_1, \ldots, t_8, t_{11}, \ldots, t_{13}$ are colored transitions. All incidence functions are identity functions, indicated by a letter "x" in case of colored transitions, using the notation proposed in [5].

lower part: Places p_1 and p_2 model operational fork-lifts and transitions t_1 and t_2 represent the Erlang-2 distribution for the time to failure. A (colored) token at p_3 indicates that the corresponding fork-lift is broken an needs repair. After a negative exponentially distributed preparation delay the worker (p_6) starts repairing a fork-lift. Transitions t_4, t_5 model the Erlang-2 distributed repair time.

middle part: This part of the GSPN represents the fixed time interval between maintenance processes. We approximate this time interval by an Erlang-2 distribution (transitions t_9, t_{10}). The beginning of maintenance is indicated by a token at place p_{13}. While p_{13} is marked, transitions t_{11} and t_{12} empty the two places p_1 and p_2 thus "moving" all tokens to place p_7. Once the places p_1, p_2 have been emptied, transition t_{13} becomes enabled (note the inhibitor arcs from p_1, p_2) and the time interval between maintenance processes is restarted.

upper part: A token at place p_7 indicates a fork-lift waiting for maintenance. After a negative exponentially distributed preparation delay the mechanic starts with maintenance of the fork-lift, provided he is available (p_{10}). The time for maintenance is given by an Erlang-2 distribution represented by transitions t_7 and t_8. The fork-lift becomes available after maintenance at place p_1.

Since places p_1 and p_2 model the availability of operational fork-lifts we are interested in the probability $p_{i,x} := P[M(p_1)(x) + M(p_2)(x) = i]$ that i fork-lifts of type x are available. $p_{i,x}$ surely depends on the chosen parameters for timed transitions. In reality most of these parameters are determined by the type of fork-lift and are thus given. I.e. given the time to failure and repair and maintenance times, one might be interested to find an "optimal" duration between maintenance intervals. This results in the following optimization problem: Find a parameter λ (the firing rate for t_9 and t_{10} representing an Erlang-2 distribution) such that $p_{k,x}$ is maximal, where k denotes the number of existing fork-lifts of type x. In our model, we assume the fixed parameter setting of 20000 min for time to failure, 2000 min repair time and 80 min maintenance time.

Performability analysis comprises numerical analyzes of

parameterized dependability models and simulative analyzes of performance models. For the later we apply the proposed scheduling algorithm, see section 2.2, and compare its effectiveness with a conventional approach.

The GSPN dependability model is solved based on the underlying Markov chain with exactly $12,246,960$ states. Using numerical steady-state analysis techniques we can compute the probability mass function (pmf) of resource configurations (p_c values) within 80 minutes. The frequency of stipulated maintenances is regarded as a parameter of the dependability model. Therefore, we must perform numerical analysis for different dependability models and obtain a set of pmf's. Figure 3 shows the pmf restricted on $\mathcal{C}^0$ for different parameter settings.

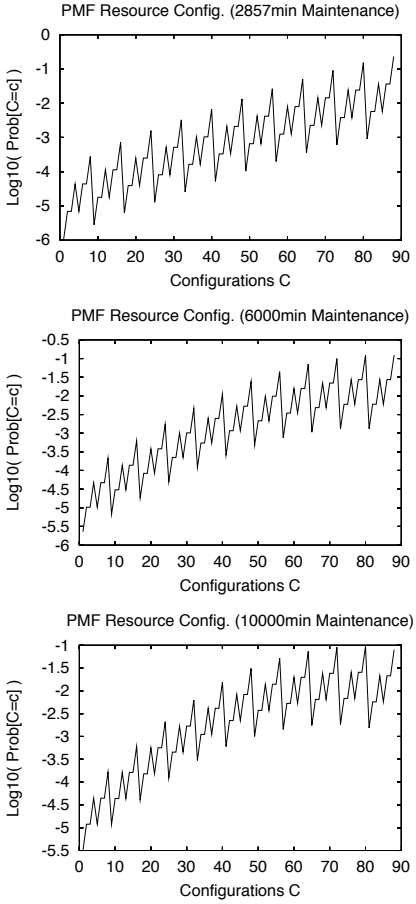

Figure 3. Pmf of resource configurations for different parameter values of the dependability model

The evaluation of the performance model requires to conduct independent simulation runs for each operational resource configuration $c \in \mathcal{C}^0$, see section 2.1. We vary the number of available fork-lifts and obtain 88 operational configurations in this particular setting. A particular simulation run determines estimates of quantities of interest (lead times of involved processes). First we consider a scenario with single simulation runs. Performance results thereby obtained based on runs that terminate at once if, either 90 percent confidence bands are attained of a width of at most 5 percent of the estimated means for all process lead times, or some maximum model time has elapsed. Totally, 88 single simulation runs need 17 hours of CPU time (measured on SUN Blade 100, Ultra Sparc IIe processor, 500 MHz). This solid stopping criterion ensures reliable results. However, the drawback is, that computational efforts for obtaining reliable performance results are by no means adapted to their relative importance (=p_c values) with respect to reliable performability results. Therefore, it is reasonable to replicate simulation runs using some adaptive control which makes use of p_c values and thus greedy reduces confidence intervals of performability quantities (see section 2.2). In our setting, the relative importance of configurations dramatically differs. Figure 3 reveals, that p_c values decreases exponentially for configurations with lower index (note the logarithmic scale). For our example, the controlled replication approach clearly outperforms its uncontrolled single run counterpart. The controlled version needs only 3-4 hours of CPU time (compared to 17 hours) in order to achieve confidence intervals that has been reached in the single run approach. Figure 4 shows the run-time behavior of the algorithm in terms of confidence intervals (percent) for performability measures (lead times of three basic processes) depending on dependability model parameters (8 settings). After 3-4 hours, charts fall below the mesh indicating confidence intervals of the single run approach. In order to gain a better understanding of the algorithms behavior we do not stop the execution at this point and proceed until 240 replications are conducted. 240 replications require 10-11 hours of CPU time and confidence intervals become fairly small ($\pm 0.1\%$), cf. Figure 4.

We conclude this section with performability results based on formula 1. Fig. 5 shows lead times of the three basic processes (`Truck Arrival`, `In-Plant Transport` and `Demand`) depending on the parameter setting of the dependability model. A time interval of about 5500 min between two maintenance inspections minimizes lead times of all processes. Lead times of processes are of interest by their own, however, their obvious interpretation necessitates deeper knowledge of the performance model.

4 Conclusions

This paper presents an approach for the performability analysis of large logistic networks by a combination of numerical analysis of a dependability model and simulative analysis of a possibly large set of performance models. From a simulation perspective it is particularly interesting

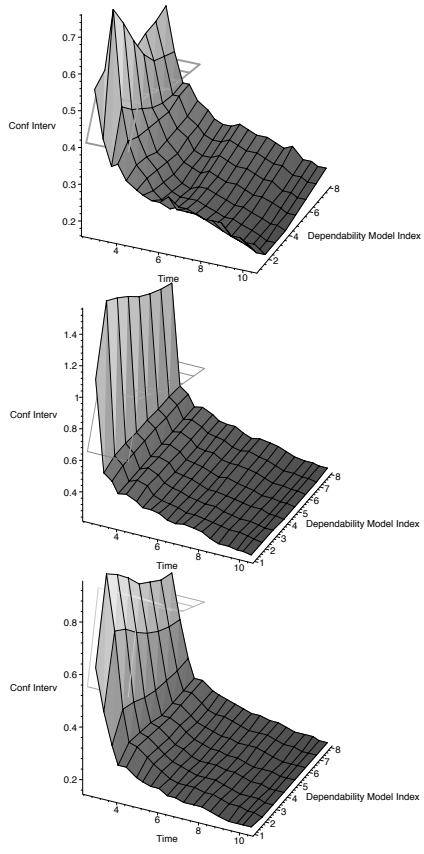

Figure 4. Run-time behavior of algorithm: confidence intervals (percent) of performability measures

how to distribute simulation time among the set of performance models to minimize the width the confidence intervals. The paper introduces an approach to perform simulation in phases and in parallel on a network of workstations. After every phase the available information is used to determine the subset of performance models which is simulated during the following phase to obtain a maximal gain measured in a maximal reduction of the confidence interval width.

References

[1] F. Bause, H. Beilner, M. Fischer, P. Kemper, and M. Völker. The Proc/B toolset for the modeling and analysis of process chains. In T. Field, P. G. Harrison, J. Bradley, and U. Harder, editors, *Computer Performance Evaluation Modeling Techniques and Tools*, pages 51–70. Springer LNCS 2324, 2002.

[2] H. Beilner, J. Mäter, and N. Weissenberg. Towards a performance modelling environment: news on HIT. In R. Puijanger, editor, *Proc. 4th Int. Conf. on Modelling Tools and Techniques for Computer Performance Evaluation*. Plenum Publishers, 1988.

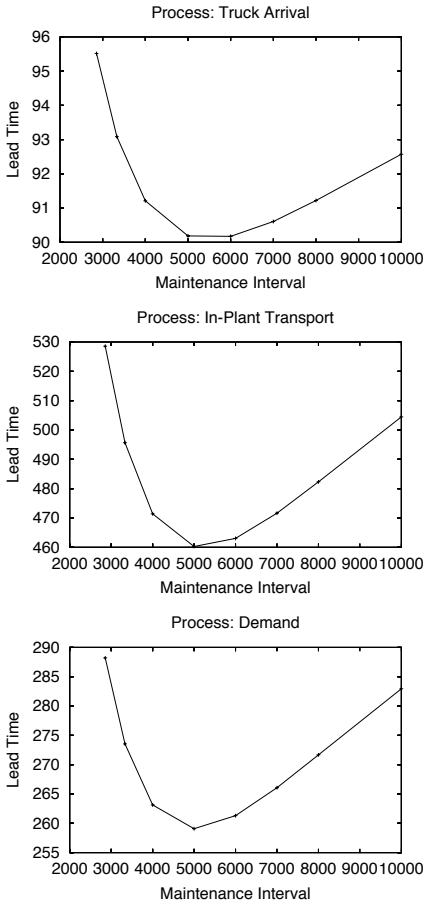

Figure 5. Performability Results: lead times of basic processes

[3] P. Buchholz, M. Fischer, P. Kemper, and C. Tepper. New features in the APNN toolbox. In P. Kemper, editor, *Tools of Aachen 2001 Int. Multiconference on Measurement, Modeling and Evaluation of Computer-Communication Systems*, pages 62–68. Universität Dortmund, Fachbereich Informatik, Forschungsbericht Nr. 760, 2001.

[4] B. R. Haverkort, R. Marie, G. Rubino, and K. Trivedi. *Performability Modelling*. Wiley, 2001.

[5] K. Jensen. *Coloured Petri Nets*. EATCS Monographs on Theoretical Computer Science. Springer, 1992.

[6] A. M. Law and W. D. Kelton. *Simulation modeling and analysis*. Wiley, 2000.

[7] J. F. Meyer. On evaluating the performability of degredable computer systems. *IEEE Transactions on Computers*, 29(8):720–731, 1980.

[8] W. H. Sanders and J. F. Meyer. A unified approach for specifying measures of performance, dependability and performability. In A. Avizienis and J. Laprie, editors, *Dependable Computing for Critical Applications*, Vol. 4 of Dependable Computing and Fault-Tolerant Systems. Springer, 1991.