A class of hierarchical queueing networks and their analysis

Peter Buchholz

Department of Informatics, University of Dortmund, D-44221, Dortmund, Germany

Received 4 March 1992; revised 1 June 1993

Queueing networks are an adequate model type for the analysis of complex system behavior. Most of the more realistic models are rather complex and do not fall into the easy solvable class of product form networks. Those models have to be analyzed by numerical solution of the underlying Markov chain and/or approximation techniques including simulation. In this paper a class of hierarchically structured queueing networks is considered and it is shown that the hierarchical model structure is directly reflected in the state space and the generator matrix of the underlying Markov chain. Iterative solution techniques for stationary and transient analysis can be modified to make use of the model structure and allow an efficient numerical analysis of large, up to now not solvable queueing networks.

Keywords: Hierarchical queueing networks; Markov chains; numerical analysis; steady state solution; transient solution.

1. Introduction

Performance analysis of dynamic systems (computer systems, communication networks, manufacturing plants, etc.) is often done by means of extended queueing networks (QNs), which can be mapped on Markov chains. Based on extended QNs various software tools have been developed (e.g., Müller-Clostermann [17], Müller-Clostermann and Szczittnick [18], Sauer and McNair [22], Veran and Potier [23]) allowing a comfortable and often graphical specification and subsequent analysis of QN models. However, often a single flat model is not adequate to describe a highly structured system. Therefore techniques have been developed and integrated in different tools to specify hierarchical models (e.g., by Beilner et al. [2] and Gordon et al. [12]).

QNs are the only paradigm permitting an efficient product form solution for a well-defined subclass of models (see Baskett et al. [1] and later extensions). Nevertheless, many realistic models cannot be solved by means of product form techniques; simulative, approximative or numerical techniques have to be used.

Simulation requires a highly methodological and computational effort and as for approximative solution techniques the results are only approximations, whereas numerical techniques are faced with an explosive growth of the state space, which prohibits the analysis of larger models.

Although hierarchical model specification techniques have become available, the hierarchical structure is normally not used for model analysis. Recently it has been noticed that the model structure influences directly the structure of the state space and the generator matrix of the underlying Markov chain and that this structure can be used for numerical solution purposes (see Buchholz [4–6]). These new techniques are very efficient, especially for the analysis of large models, and extend the class of solvable models on a given hardware significantly. Nevertheless, the solution is still performed using well-established iterative solution methods from linear algebra, which have been proved to be successful for the analysis of QNs (e.g., in the overview papers by Cao and Stewart [8], Kaufman [15], Krieger et al. [16] for stationary analysis and in Reibman and Trivedi [21] for transient analysis). The difference from the conventional approach computing the overall generator matrix of the model is due to the structure of the model that allows the use of iterative techniques without computing the huge generator. Therefore the memory requirements are much smaller than in the conventional case. The number of states which has been handled on a contemporary workstation is approximately 2000000 (compared with around 200000 using conventional techniques) and additionally, a speed-up of the solution for models with 50000 or more states can be observed.

The balance of this paper is outlined as follows. In section 2 the class of hierarchical QNs is introduced. Section 3 describes the state space and the generator matrix structure. Afterwards, in section 4, numerical solution techniques for stationary and transient analysis, making direct use of the model structure, are presented. In section 5 the advantages of the new approach are shown by means of an example. The paper ends with some conclusions.

Throughout the paper vectors are denoted by boldface italic letters, matrices by boldface italic capital letters. All vectors are row vectors, $\boldsymbol{a}^T$ ($\boldsymbol{A}^T$) describes the transposed vector $\boldsymbol{a}$ (matrix $\boldsymbol{A}$). $\boldsymbol{I}$ is the identity matrix, $\boldsymbol{e}$ the vector with 1 in every position and $\boldsymbol{e}_i$ a vector with 1 in position i and 0 elsewhere. $\| \ldots \|$ denotes the cardinality of a set.

2. A hierarchical description of QNs

Models are described by a two level hierarchy. J low level submodels (*LLMs*) are specified as ordinary (extended) QNs and are numbered from 1 to J. The *LLMs* are connected via a high level model (*HLM*), specifying routing of entities between different *LLMs*. Although the hierarchy includes only two levels, it is, of course, straightforward to extend the description to an arbitrary number of levels. The

technique can be used as long as the models are tree-like structured and submodels with an "autonomous" behavior can be separated.

A *HLM* specifies the connection between *LLMs*. The dynamic behavior of the model is realized by the movement of entities belonging to classes k from a set K. Subsets of communicating classes are combined to chains. Let C be the set of chains and K_c be the set of classes belonging to chain c (i.e., $K = \cup K_c$, $K_c \cap K_{c'} = \emptyset$ for all $c \neq c'$). Since the overall model will be analyzed by numerical solution of the global balance equations, the state space has to be finite, which implies a finite population N_c for each chain $c \in C$ in the model.

The *HLM* specifies the routing of entities between *LLMs*. As common in QNs the movement of entities is timeless. Since one goal of hierarchical modelling is the hiding of information among the levels, the *HLM* is not aware of the internal *LLM* structure, which can be a complex QN or a single station. The only available information about the state of a specific *LLM* j is the population of entities in this *LLM* described by a vector $\boldsymbol{n}_j \in \mathbb{N}^{\|K\|}$. A state of the *HLM* is given by a vector $\boldsymbol{n}$, as specified in the following equation:

$$\boldsymbol{n} = (\boldsymbol{n}_1 \ldots \boldsymbol{n}_J), \quad \boldsymbol{n}_j \in \mathbb{N}^{\|K\|} \quad (1 \leqslant j \leqslant J),$$

$$\sum_{j=1}^{J} \sum_{k \in K_c} n_j(k) = N_c \quad \text{for all } c \in C. \tag{1}$$

Let Z_0 be the set of all possible states of the *HLM*, which will be further described below. The movement of entities in the *HLM* is quantified with routing probabilities, which are allowed to depend on the whole information available in the *HLM* (i.e., the state vector $\boldsymbol{n}$). $r(i,k,j,l,\boldsymbol{n})$ $(1 \leqslant i,j \leqslant J, k, l, \in K_c, c \in C, \boldsymbol{n} \in Z_0)$ is the routing probability of a class k entity leaving *LLM* i and entering immediately *LLM* j as class l entity, if the *HLM* is in state $\boldsymbol{n}$.

$$\sum_{j=1}^{J} \sum_{l \in K_c} r(i,k,j,l,\boldsymbol{n}) = 1.0 \quad \text{for all } i \in \{1 \ldots J\}, c \in C, k \in K_c, \boldsymbol{n} \in Z_0. \tag{2}$$

The specification of routing probabilities between *LLMs* completes the *HLM* specification. *LLMs* are specified as extended QNs. The possible specification elements depend on the used tool. We have practical experience with QNAP2 (Veran and Potier [23]) and MACOM (Müller-Clostermann and Sczittnick [18]), both providing a very general class of QNs.

A *LLM* j has to provide a clearly defined interface to its environment realized by sets of input and output ports, one input and one output port belong to exactly one entity type. When j is embedded in a *HLM*, the entity types of the *LLM* are related to the classes of the *HLM*. One entity class is mapped exclusively on exactly one entity type (i.e., the behavior of an entity inside a *LLM* is determined by its type identity, the behavior in the *HLM* is determined by its class identity). Inside j,

an entity type k is interpreted as an entity chain including the classes $\{k_1 \ldots k_X\}$. After leaving the *LLM* an entity is characterized only by its class identity in the *HLM*. Obviously the class identity of an entity in the *HLM* does not change during its sojourn in a *LLM*. Let k be one entity type in *LLM* j. For the isolated specification of j and the generation of the state space, j is combined with a pseudo environment. The pseudo environment consists, independent of the real environment, of a finite capacity source per entity type. Entities are generated by the source with exponentially distributed interarrival times with rate $\lambda > 0.0$, as long as the number of entities does not exceed the capacity $N_{j,k}$ for type k. If j includes $N_{j,k}$ type k entities, the source for k is switched off, until a type k entity leaves j. After leaving, entities are absorbed by a sink.

If the *HLM* state space Z_0 is known during specification of *LLM* j, the capacity $N_{j,k}$ is calculated as shown in eq. (3). Otherwise $N_{j,k}$ is set to an arbitrary value and j can be embedded in any environment, where the possible population of type k entities in j does not exceed $N_{j,k}$.

$$N_{j,k} = \max_{\boldsymbol{n} \in Z_0}(n_j(k)) \,. \tag{3}$$

Additionally, j might include internal entity types (resp. chains for the isolated *LLM*), which do not leave j and contain a finite number of entities. The internal description of j might include all constructs provided by the used tool. Two restrictions are made here for notational convenience; first, only a single entity departs at one time from j; second, an entity is not allowed to arrive and depart instantaneously in j. We assume that the state space of j, in combination with the pseudo environment, generated starting with the empty *LLM*, is irreducible.

3. State space and generator matrix structures

The state space of a hierarchical model is, like the model itself, structured in the state space of the *HLM* and the J state spaces of the *LLMs*. The dynamic behavior of the *HLM* and each *LLM* can be described by several isolated transition matrices including transition rates or conditional probabilities. These isolated transition matrices provide a complete specification of the generator matrix of the Markov chain underlying the complete hierarchical QN, which is necessary for the analysis of the model. The sequel of this section describes the state spaces and transition matrices of the isolated model parts and introduces the structure of the generator matrix.

The *HLM* is specified by the routing probabilities between *LLMs*. The state space Z_0 can be constructed starting from any initial state $\boldsymbol{n} \in Z_0$ by generation of all successor states defined by the routing probabilities. One state of the *HLM* describes a fixed distribution of entities over the *LLMs*. To generate all possible successor states, it has to be known which entities can depart from each *LLM*; how-

ever, this information depends on the local *LLM* state in general, which is not transparent in the *HLM*. A necessary condition for a class k entity to depart is, of course, $n_j(k) > 0$. This condition is obviously not sufficient. There are two ways to generate Z_0; the first is to use information about the internal *LLM* structure for *HLM* state space generation; the second is to assume that, whenever $n_j(k) > 0$, a class k entity can possibly depart. In the latter case states might be generated which are never reached in the overall model (e.g., if the *LLM* is a single station with priority preemptive scheduling, only entities with the highest priority can depart; assuming that all entities can possibly depart might yield non-reachable *HLM* states). This does not matter much, since usually not many of these states are generated and an efficient algorithm for the functional analysis of hierarchical models has been developed (see Buchholz [7]) and can be used to delete non-reachable states. We assume here that Z_0 forms an irreducible state space.

Transitions between states of the *HLM* are quantified with routing probabilities. Two states $\boldsymbol{n}$ and $\boldsymbol{n}'$ from Z_0 can potentially communicate via one transition, if they are equal or distinct in two positions by one entity. This is expressed by a measure $\Delta(\boldsymbol{n}, \boldsymbol{n}')$.

$$\Delta(\boldsymbol{n}, \boldsymbol{n}') = \begin{cases} (0,0,0,0) & \text{if } \boldsymbol{n} = \boldsymbol{n}', \\ (i,k,j,l) & \text{if } \boldsymbol{n} = \boldsymbol{n}' + \boldsymbol{e}_{i,k} - \boldsymbol{e}_{j,l}, \\ (\infty,\infty,\infty,\infty) & \text{else}, \end{cases} \tag{4}$$

where $\boldsymbol{e}_{i,k}$ describes the situation with one additional entity of class k in i.

States from Z_0 should be ordered in a well-defined way (e.g., lexicographically), let $f(\boldsymbol{n})$ be a function assigning a unique number from $\{1 \ldots \|Z_0\|\}$ to state $\boldsymbol{n}$. For notational convenience we define $p(f(\boldsymbol{n}), f(\boldsymbol{n}'))$ as $r(i,k,j,l,\boldsymbol{n})$ if $\Delta(\boldsymbol{n}, \boldsymbol{n}') = (i,k,j,l)$ and 0.0 else. Furthermore, $p_{i,k}(f(\boldsymbol{n}))$ is defined as $r(i,k,i,k,\boldsymbol{n})$. For each state $\boldsymbol{n} \in Z_0$ the following sets of successor states are needed:

$$\begin{aligned} \Gamma(\boldsymbol{n}) &= \{\boldsymbol{n}' | \Delta(\boldsymbol{n}, \boldsymbol{n}') = (i,k,j,l) \text{ or } (0,0,0,0)\}, \\ \Gamma_{\pm}(\boldsymbol{n}) &= \{\boldsymbol{n}' | \Delta(\boldsymbol{n}, \boldsymbol{n}') = (i,k,j,l) \text{ and } i = j, k \neq l\}, \\ \Gamma_{-}(\boldsymbol{n}) &= \{\boldsymbol{n}' | \Delta(\boldsymbol{n}, \boldsymbol{n}') = (i,k,j,l) \text{ and } i < j\}, \\ \Gamma_{+}(\boldsymbol{n}) &= \{\boldsymbol{n}' | \Delta(\boldsymbol{n}, \boldsymbol{n}') = (i,k,j,l) \text{ and } i > j\}, \\ \Gamma_{<}(\boldsymbol{n}) &= \{\boldsymbol{n}' | \boldsymbol{n}' \in \Gamma(\boldsymbol{n}) \text{ and } f(\boldsymbol{n}') < f(\boldsymbol{n})\}, \\ \Gamma_{>}(\boldsymbol{n}) &= \{\boldsymbol{n}' | \boldsymbol{n}' \in \Gamma(\boldsymbol{n}) \text{ and } f(\boldsymbol{n}') > f(\boldsymbol{n})\}. \end{aligned} \tag{5}$$

The state space and transition matrices of *LLM* j are generated from a model j and the finite capacity source defined in section 2. Let $N_j = (N_{j,1} \ldots N_{j,\|K\|})$ be the vector of maximum populations per entity type and Z_j the state space of this model. The set of possible populations inside j is given by

$$NM_j = \{\boldsymbol{n}_j \in \mathbb{N}^{\|K\|} \mid \boldsymbol{0} \leqslant \boldsymbol{n}_j \leqslant \boldsymbol{N}_j\}. \tag{6}$$

According to the population in j, the state space Z_j is decomposed into disjoint subspaces $Z_j(\boldsymbol{n}_j)(\boldsymbol{n}_j \in NM_j)$, including all states with population $\boldsymbol{n}_j$ in j.

$$Z_j = \bigcup_{\boldsymbol{n}_j \in NM_j} Z_j(\boldsymbol{n}_j). \tag{7}$$

Let $\boldsymbol{Q}_j$ be the generator matrix of j in combination with the pseudo environment, where the states of one subspace are grouped together and the subspaces are ordered according to the lexicographical ordering of the population vector $\boldsymbol{n}_j$. For a single type LLM with maximum population N, $\boldsymbol{Q}_j$ has the following structure:

$$\boldsymbol{Q}_j = \begin{pmatrix} \boldsymbol{Q}_j^0 - \lambda \boldsymbol{I} & \lambda \boldsymbol{U}_j^{1-1} & & & \\ \boldsymbol{S}_j^{1-1} & \boldsymbol{Q}_j^1 - \lambda \boldsymbol{I} & \lambda \boldsymbol{U}_j^{2-1} & & \\ & \ddots & & & \\ & & \boldsymbol{S}_j^{(N-1)-1} & \boldsymbol{Q}_j^{N-1} - \lambda \boldsymbol{I} & \lambda \boldsymbol{U}_j^{N-1} \\ & & & \boldsymbol{S}_j^{N-1} & \boldsymbol{Q}_j^N \end{pmatrix}, \tag{8}$$

where λ is the service rate of the finite capacity source.

For *LLMs* including more than one type, the matrix is no longer block tridiagonal, but the structure is quite similar. All Q-submatrices are situated in the main diagonal, the S-submatrices are located in the lower triangle and the U-submatrices in the upper triangle. The three different types of matrices define the behavior of *LLM* j.

Matrices $\boldsymbol{Q}_j^{n_j}$ include all transitions inside j (e.g., an entity travels from one station to another without leaving j) and in the main diagonal the negative rates out of a state caused by transitions originated in j.

$$\boldsymbol{Q}_j^{n_j} \in \mathbb{R}^{\|Z_j(\boldsymbol{n}_j)\| \times \|Z_j(\boldsymbol{n}_j)\|}, \quad \boldsymbol{Q}_j^{n_j} \boldsymbol{e}^T \leqslant \boldsymbol{0}^T. \tag{9}$$

Matrices $\boldsymbol{S}_j^{n_j-k}$ include the departure rates of type k entities from j.

$$\boldsymbol{S}_j^{n_j-k} \in \mathbb{R}_+^{\|Z_j(\boldsymbol{n}_j)\| \times \|Z_j(\boldsymbol{n}_j - \boldsymbol{e}_k)\|} \quad \text{for all } \boldsymbol{n}_j, \boldsymbol{n}_j - \boldsymbol{e}_k \in NM_j, \tag{10}$$

where $\mathbb{R}_+$ is the set of non-negative real numbers.

The third matrix type $\boldsymbol{U}_j^{n_j-k}$ describes the arrival of type k entities. The arrival of an entity is something passive from the *LLMs* viewpoint. Therefore we cannot define arrival rates independently from the environment. However, the behavior of j under the condition of a type k arrival can be quantified by conditional probabilities included in the following matrix:

$$\boldsymbol{U}_j^{n_j-k} \in \mathbb{R}_+^{\|Z_j(\boldsymbol{n}_j - \boldsymbol{e}_k)\| \times \|Z_j(\boldsymbol{n}_j)\|}, \quad \boldsymbol{U}_j^{n_j-k} \boldsymbol{e}^T = \boldsymbol{e}^T, \quad \text{for all } \boldsymbol{n}_j, \boldsymbol{n}_j - \boldsymbol{e}_k \in NM_j. \tag{11}$$

Let Z_G be the state space of the overall model, a single state $z \in Z_G$ is given by

the states of all *LLMs* according to one state of the *HLM* (i.e., one distribution of the entities among the *LLMs*). Z_G can be decomposed into subspaces $Z_G(\boldsymbol{n})(\boldsymbol{n} \in Z_0)$, which include all states with population n_j in *LLM* j.

$$Z_G = \bigcup_{\boldsymbol{n} \in Z_0} Z_G(\boldsymbol{n}), \quad Z_G(\boldsymbol{n}) = \bigtimes_{j=1}^{J} Z_j(n_j). \tag{12}$$

Z_G has a structure that is defined by the structure of the model, the same holds for the generator matrix of the overall model $\boldsymbol{Q}_G$. Before we investigate the structure of $\boldsymbol{Q}_G$, tensor operations, building a base for matrix generation, have to be introduced briefly. Complementary information can be found in Davio [11] and Plateau [19].

DEFINITION 1

The tensor product of two matrices $\boldsymbol{A}_1 \in \mathbb{R}^{r_1 \times c_1}$ and $\boldsymbol{A}_2 \in \mathbb{R}^{r_2 \times c_2}$ is defined as $\boldsymbol{C} = \boldsymbol{A}_1 \otimes \boldsymbol{A}_2$, $\boldsymbol{C} \in \mathbb{R}^{r_1 r_2 \times c_1 c_2}$,
where

$$C((i_1 - 1) * r_2 + i_2, (j_1 - 1) * c_2 + j_2) = A_1(i_1, j_1) A_2(i_2, j_2)$$
$$(1 \leqslant i_x \leqslant r_x, 1 \leqslant j_x \leqslant c_x, x \in \{1, 2\}).$$

The tensor sum of two matrices $\boldsymbol{B}_1 \in \mathbb{R}^{n_1 \times n_1}$ and $\boldsymbol{B}_2 \in \mathbb{R}^{n_2 \times n_2}$ is defined as

$$\boldsymbol{D} = \boldsymbol{B}_1 \oplus \boldsymbol{B}_2 = \boldsymbol{B}_1 \otimes \boldsymbol{I}_{n_2} + \boldsymbol{I}_{n_1} \otimes \boldsymbol{B}_2,$$

where $\boldsymbol{I}_n$ is the $n \times n$ identity matrix.

DEFINITION 2

The generalized tensor product is defined as $\boldsymbol{C} = \otimes_{j=1}^{J} \boldsymbol{A}_j = (\otimes_{j=1}^{J-1} \boldsymbol{A}_j) \otimes \boldsymbol{A}_J$. The generalized tensor sum is defined as $\boldsymbol{C} = \oplus_{j=1}^{J} \boldsymbol{A}_j = (\oplus_{j=1}^{J-1} \boldsymbol{A}_j) \oplus \boldsymbol{A}_J$, where $\otimes_{j=1}^{1} \boldsymbol{A}_j = \oplus_{j=1}^{1} \boldsymbol{A}_j = \boldsymbol{A}_1$.

Transitions in the overall model are distinguished by being internal or external to a *LLM*. Internal transitions describe the movement of one entity inside one *LLM* and are collected in a matrix $\boldsymbol{Q}_{IG}$. External transitions describe the movement of an entity leaving one *LLM* and entering another or the same one. Only external transitions can potentially change the *HLM* state.

$$\boldsymbol{Q}_G = \begin{pmatrix} \boldsymbol{Q}_{IG}^{1} & & \\ & \ddots & \\ & & \boldsymbol{Q}_{IG}^{\|Z_0\|} \end{pmatrix} + \begin{pmatrix} \boldsymbol{Q}_{EG}^{1,1} & \cdots & \boldsymbol{Q}_{EG}^{1,\|Z_0\|} \\ \vdots & & \vdots \\ \boldsymbol{Q}_{EG}^{\|Z_0\|,1} & \cdots & \boldsymbol{Q}_{EG}^{\|Z_0\|,\|Z_0\|} \end{pmatrix}. \tag{13}$$

$\boldsymbol{Q}_{IG}$ includes all transitions inside *LLMs* with a fixed population in each *LLM* j. Since transitions inside *LLMs* are independent of one another by assump-

tion, each submatrix of Q_{IG} can be expressed by the tensor sum of the *LLM*-matrices describing internal transitions.

$$Q_{IG}^{f(n)} = \oplus_{j=1}^{J} Q_j^{n_j} . \tag{14}$$

Q_{EG} includes transition rates between *LLMs*. Submatrix $Q_{EG}^{f(n),f(n')}$ contains the transition rates between states from $Z(n)$ and $Z(n')$. The matrix only contains non-zero entries if n can be transformed into n' by an entity leaving one *LLM* and entering another or the same one immediately. All non-zero submatrices have similar structures. A S-matrix describes the transition in the *LLM* where the entity departs, a U-matrix describes the transition in the *LLM* where the entity arrives, and identity matrices show that the states of the remaining *LLMs* do not change. Since the departure and arrival of an entity are directly dependent, the resulting submatrices are built using the tensor product (resp. the ordinary product if the source and destination *LLM* are identical) for submatrix construction (see Buchholz [4,5]). In particular the order of the different matrix operations depends on the numbers of the involved *LLMs*.

$$Q_{EG}^{f(n),f(n')} = \begin{cases} \sum_{j=1}^{J} I_{l_j(n)} \otimes \left(\sum_{k \in K} p_{j,k}(f(n)) S_j^{n_j - k} U_j^{n_j - k} \right) \otimes I_{u_j(n)} & \text{if } n = n' , \\ p(f(n), f(n')) I_{l_i(n)} \otimes S_i^{n_i - k} \otimes I_{d_{ji}(n)} \otimes U_j^{n'_j - 1} \otimes I_{u_j(n)} & \text{if } n' \in \Gamma_-(n) , \\ p(f(n), f(n')) I_{l_j(n)} \otimes U_j^{n'_j - 1} \otimes I_{d_{ij}(n)} \otimes S_i^{n_i - k} \otimes I_{u_i(n)} & \text{if } n' \in \Gamma_+(n) , \\ p(f(n), f(n')) I_{l_i(n)} \otimes S_i^{n_i - k} U_i^{n'_i - 1} \otimes I_{u_i(n)} & \text{if } n' \in \Gamma_\pm(n') , \\ 0 & \text{else} , \end{cases} \tag{15}$$

where

$$l_i(n) = \prod_{j=1}^{i-1} \| Z_j(n_j) \| , \quad u_i(n) = \prod_{j=i+1}^{J} \| Z_i(n_j) \| , \quad d_{ij}(n) = l_i(n) - l_{j+1}(n) .$$

Q_G has, indeed, a structure, which is related to the model structure and easy operations to generate the matrix from the isolated descriptions of the *LLMs* and the *HLM* are available. We assume Q_G to be irreducible.

4. Numerical solution techniques

The numerical analysis of non-product form QNs is performed by analyzing the set of global balance equation defined in Q_G. Since the matrix is normally rather

large and sparse, direct solution techniques, modifying the non-zero structure during the solution process, are not usable for the analysis of even moderate sized models. More appropriate are iterative techniques leaving the sparse matrix structure as it is. Nevertheless, the space requirement rather than the time requirement is the most limiting factor during analysis. This is true even for machines with virtual memory, since the access time to secondary memory often exceeds primary memory access time by the order of a magnitude, causing a non- acceptable solution time for models that do not fit into primary memory. For hierarchical QNs iterative solution techniques can be modified to make use of the special structure of generator matrix by avoiding the generation of Q_G, which yields a significant reduction in the space requirements for the solution and enlarges therefore the class of solvable models.

Various iterative techniques for the calculation of stationary or transient results in QNs exist, for an overview see the papers by Kaufman [15], Krieger et al. [16] or Reibman and Trivedi [21]. We introduce in the remainder of this section only a few techniques in the context of hierarchical QNs. However, these techniques are very efficient, from our experience, and the integration of other techniques in the framework is straightforward as shown by Buchholz [4,5].

Let π be the stationary solution vector of the Markov chain described by the hierarchical QN (i.e., $\pi Q_G = 0$, $\pi e^T = 1.0$). From π the means of all stationary performance quantities like throughputs, populations, sojourn times, etc., can be computed easily. π is unique since Q_G is an irreducible generator. According to the structure of the model and the state space, π can be decomposed into subvectors $\pi_{f(n)}$ including the stationary probabilities of states from subset $Z_G(n)$.

The idea of all iterative solution methods for stationary analysis is to start with an arbitrary initial vector $\pi^0 > 0$ and to multiply this vector with an iteration matrix T_G, which can be easily determined from Q_G, until the iteration vector π^m is sufficiently near to π. Of course, the difference $|\pi^m - \pi|$ can only be estimated using assumptions about the convergence behavior. The convergence behavior and estimation of the convergence for different iterative solution algorithms has been considered in various papers (e.g., by Cao and Stewart [8], Gross et al. [13], Kaufman [15], and Krieger et al. [16]) and will not be further investigated here.

One of the most efficient iterative techniques is the Gauss–Seidel method (see Krieger et al. [16]), which will be used to describe the integration of iterative solution techniques with the structured generator matrices of hierarchical QNs. For our purposes Gauss–Seidel has to be modified slightly, yielding a method between point and block Gauss–Seidel. Let $\tilde{Q}$ be the matrix Q with the diagonal elements set to zero and let D_G be a diagonal matrix including the diagonal elements of Q_G. In the modified Gauss–Seidel method a subvector $\pi^m_{f(n)}$ is determined from π^{m-1} and π^m as

$$\pi^m_{f(n)} = \Bigg(\sum_{n' \in \Gamma_>(n)} \pi^{m-1}_{f(n')} Q^{f(n'),f(n)}_{EG} + \sum_{n' \in \Gamma_<(n)} \pi^m_{f(n')} Q^{f(n'),f(n)}_{EG} + \pi^{m-1}_{f(n)} (\tilde{Q}^{f(n)}_{IG} + \tilde{Q}^{f(n),f(n)}_{EG}) \Bigg) (D^{f(n)}_G)^{-1}. \quad (16)$$

Before we include the structure of the matrices in the iteration steps, the compatibility of tensor sums/products with regular matrix sums/products is introduced in (17) (see also Buchholz [5], Davio [11], or Plateau [19]).

$$\oplus_{j=1}^J Q_j = \sum_{j=1}^J I_{c_{j-1}} \otimes Q_j \otimes I_{r_{j+1}}, \quad \otimes_{j=1}^J Q_j = \prod_{j=1}^J I_{c_{j-1}} \otimes Q_j \otimes I_{r_{j+1}}, \quad (17)$$

where

$$c_j = \prod_{i=1}^j col(Q_j), \quad r_j = \prod_{i=j}^J row(Q_j),$$

$col(Q_j)(row(Q_j))$ is the number of columns (rows) of Q_j.

A single matrix $I_l \otimes Q \otimes I_u$ can be determined easily from Q, since

- it contains l non-zero diagonal blocks,
- each diagonal block consists of the modified matrix Q, where every element $Q(z,z)$ is substituted by a diagonal matrix of order $u \times u$, with $Q(z,z)$ in the main diagonal, and
- all other elements are zero.

Using the above properties and the structure of the matrices given in (14) and (15), the multiplication of the iteration vector with the submatrices from Q_{IG} and Q_{EG} in (16) becomes

$$\pi^m_{f(n)} \tilde{Q}^{f(n)}_{IG} = \pi^{m-1}_{f(n)} \sum_{j=1}^J I_{l_j(n)} \otimes \tilde{Q}^{n_j}_j \otimes I_{u_j(n)},$$

$$\pi^m_{f(n)} Q^{f(n),f(n')}_{EG} =$$

$$\begin{cases} \pi^m_{f(n)} p(f(n),f(n'))(I_{l_i(n)} \otimes S_i^{n_i-k} \otimes I_{u_i(n)})(I_{l_j(n')} \otimes U_j^{n'_j-l} \otimes I_{u_j(n)}) & \text{if } n' \in \Gamma_-(n) \cup \Gamma_\pm(n), \\ \pi^{m-1}_{f(n)} p(f(n),f(n'))(I_{l_j(n)} \otimes U_j^{n'_j-l} \otimes I_{u_j(n)})(I_{l_i(n')} \otimes S_i^{n_i-k} \otimes I_{u_i(n)}) & \text{if } n' \in \Gamma_+(n), \\ \pi^{m-1}_{f(n)} \sum_{i,k} p_{i,k}(f(n))(I_{l_i(n)} \otimes \widetilde{SU}_i^{n_i-k} \otimes I_{u_i(n)}) & \text{if } n = n', \\ 0 & \text{else}, \end{cases} \quad (18)$$

where $SU_i^{n_i-k} = S_i^{n_i-k} U_i^{n_i-k}$.

The matrices D_G and $SU_i^{n_i-k}$ can be either generated before the iteration, or the

elements are computed directly in each iteration step, which needs more effort but less space. In any case the iteration is performed by the repeated use of a function $mult(\pi, Q, l, u)$ calculating $\pi(I_l \otimes Q \otimes I_u)$ without generating the matrix resulting from the tensor product. This approach is possible, since all elements of the resulting matrix are determined by the elements of Q and the integers l and u. A specification of such a procedure for sparse matrix structures is given in the appendix, the implementation needs less than 20 lines of code. The solution procedure is not changed through the tensor operations, therefore the results of the analysis are exact and the convergence behavior of the algorithms is not affected by the tensor operations. The advantage of the tensor based multiplication concerning the amount of storage is significant as shown below.

The structure of the model can be used for further improvements of the solution technique by integrating aggregation steps to determine an a priori guess of the initial vector or to speed up to convergence of the solution technique as pointed out by Cao and Stewart [8]. The initial vector π^0 is normally chosen as a uniform distribution, since no other a priori information is available. However, the initial vector has a significant influence on the number of iterations and the time needed for the solution. In hierarchical models a priori information can be gained by introduction of an aggregation step to determine π^0. An aggregation step consists of two parts; first, the distribution inside each *LLM* j for each population n_j is approximated, afterwards an aggregated generator matrix Q_{AG} is computed and used for the determination of the distribution between the aggregated states. This approach is known from decomposition and aggregation techniques by Courtois [9].

The distribution inside the *LLMs* can be calculated by short-circuiting the *LLMs* for all population vectors. Let $y_j^{n_j}$ be the approximated distribution of j with population n_j, which is determined from the following equation:

$$y_j^{n_j}\left(Q_j^{n_j} + \sum_{k \in K} S_j^{n_j - k} U_j^{n_j - k}\right) = 0, \quad y_j^{n_j} e^T = 1.0. \tag{19}$$

If some of the *LLMs* are product form networks, the short circuit vectors are calculated more efficiently using a product form algorithm. The vector $y_j^{n_j}$ for *LLMs* with a non- reasonable short circuit behavior (e.g., the above equation is not uniquely solvable) is chosen as a uniform distribution or any distribution which can be derived from the isolated *LLM* matrices. The aggregated system is constructed by substituting each subset of states $Z_G(n)$ by a single aggregate state yielding values for the aggregated transition rates as given in the following equation:

$$Q_{AG}(f(n), f(n')) = \begin{cases} p(f(n), f(n')) y_i^{n_i} S_i^{n_i - k} e^T & \text{if } n' \in \Gamma(n)/n, \\ 0 & \text{else}, \end{cases} \tag{20}$$

$$Q_{AG}(f(n), f(n)) = -\sum_{n' \in \Gamma(n)/n} Q_{AG}(f(n), f(n')).$$

The stationary distribution of the aggregated system is calculated from the following equation:

$$\boldsymbol{x}\boldsymbol{Q}_{AG} = \boldsymbol{0}, \quad \boldsymbol{x}\boldsymbol{e}^T = 1.0. \tag{21}$$

From a higher view, the aggregated system equals the *HLM*, where every *LLM j* is substituted by an exponential station with population dependent service rate $\mu_{jk}(\boldsymbol{n}_j)$ for class k entities. The initial vector $\boldsymbol{\pi}^0$ of the whole system is determined by multiplying the conditional distribution inside the subset $Z_G(\boldsymbol{n})$ with the probability to be in that subset.

$$\boldsymbol{\pi}^0_{f(\boldsymbol{n})} = x(f(\boldsymbol{n}))(\otimes_{j=1}^{J} \boldsymbol{y}_j^{\boldsymbol{n}_j}). \tag{22}$$

It is known (see Courtois [9,10]) that $\boldsymbol{\pi}^0$ is the exact stationary distribution, if the whole model is a product form network, and a good approximation, if the model violates only slightly product form or the coupling between the *LLMs* is loose. To speed up the convergence of iterative solution techniques in models with loosely coupled *LLMs*, aggregation steps can also be used. The underlying methods are known as aggregation/disaggregation (a/d) algorithms described in Cao and Stewart [8] or Krieger et al. [16]. However, a/d algorithms have been described on the level of the generator matrix without taking care of the model structure. In hierarchical QNs a natural interpretation for these techniques is given.

Aggregation steps can be combined with any iterative solution technique. Let $\boldsymbol{\pi}^m_{f(\boldsymbol{n})}$ be the actual solution vector for the states in $Z_G(\boldsymbol{n})$ reached during iteration and $\bar{\boldsymbol{\pi}}^m_{f(\boldsymbol{n})} (= \boldsymbol{\pi}^m_{f(\boldsymbol{n})}/(\boldsymbol{\pi}^m_{f(\boldsymbol{n})}\boldsymbol{e}^T))$ its normalized version. Before performing the next iteration step, the iteration vector can be redirected by an aggregation step. The elements of the aggregated matrix are calculated as shown in the following equation:

$$Q_{AG}(f(\boldsymbol{n}), f(\boldsymbol{n}')) = \begin{cases} p(f(\boldsymbol{n}), f(\boldsymbol{n}'))\bar{\boldsymbol{\pi}}^m_{f(\boldsymbol{n})} \boldsymbol{I}_{l_i(\boldsymbol{n}_i)} \otimes \boldsymbol{S}_i^{\boldsymbol{n}_i - k} \otimes \boldsymbol{I}_{u_i(\boldsymbol{n}_i)} \boldsymbol{e}^T & \text{if } \boldsymbol{n}' \in \Gamma(\boldsymbol{n})/\boldsymbol{n}, \\ 0 & \text{else}, \end{cases} \tag{23}$$

$$Q_{AG}(f(\boldsymbol{n}), f(\boldsymbol{n})) = -\sum_{\boldsymbol{n}' \in \Gamma(\boldsymbol{n})/\boldsymbol{n}} Q_{AG}(f(\boldsymbol{n}), f(\boldsymbol{n}')).$$

The stationary distribution of the aggregated system is computed using eq. (21) and a new iteration vector $\hat{\boldsymbol{\pi}}^m_{f(\boldsymbol{n})}$ is given by

$$\hat{\boldsymbol{\pi}}^m_{f(\boldsymbol{n})} = x(f(\boldsymbol{n}))\bar{\boldsymbol{\pi}}^m_{f(\boldsymbol{n})}; \tag{24}$$

$\hat{\boldsymbol{\pi}}^m_{f(\boldsymbol{n})}$ can then be used for further iteration steps.

The aggregation step has an interpretation in the *HLM*, since the aggregated system describes the *HLM* where every *LLM j* is substituted by an exponential station with service rate $\mu_{jk}(\boldsymbol{n}_1, \ldots, \boldsymbol{n}_J)$ depending on the whole state of the *HLM*. The idea of a/d steps can be extended by using subnets of *LLMs* instead of single

LLMs yielding an improvement, if the subnets are loosely couple. However, the approach is very similar and need not be described here in detail.

The goal of a transient analysis of a hierarchical QN is the determination of $\pi(t)$, the state vector at time $t \geq 0$, which, of course, depends on the initial state distribution $\pi(0)$. The vector $\pi(t)$ is described by the Kolmogorov differential equations $\pi'(t) = \pi(t)Q_G$, with the general solution $\pi(t) = \pi(0)e^{Qt}$. However, instead of using standard differential equation solvers, it is often preferable to exploit the special structure of the system for the solution. The most efficient solution method is the so-called "randomization" or "uniformization" method as pointed out by Gross and Miller [14], or Reibman and Trivedi [21]. The method is described in the following equation:

$$\pi(t) = \sum_{m=0}^{\infty} \tau^m \mathrm{e}^{-\alpha t} \frac{(\alpha t)^m}{m!}, \tag{25}$$

where

$$\tau^m = \tau^{m-1} + \frac{1}{\alpha}\tau^{m-1}Q_G, \quad \alpha \geq \max_{z \in Z_G} |Q_G(z,z)|, \quad \tau^0 = \pi(0).$$

The idea behind "randomization" is to reduce the continuous time Markov chain to a chain in discrete time and a subordinated Poisson process. The evaluation of the discrete time chain is realized by the vectors τ^m, specifying the state after m steps. For practical implementations the infinite sum of the above equation has to be truncated after a finite number M of summations. The vector $\pi(t)$ then can be bounded as shown in (26) and by Gross and Miller [14].

$$\begin{aligned} \pi_{min}(t) = \sum_{m=1}^{M} \tau^m \mathrm{e}^{-\alpha t} \frac{(\alpha t)^m}{m!} \leq \pi(t) \leq \pi_{max}(t) = \pi_{min}(t) \\ + e\left(1.0 - \mathrm{e}^{-\alpha t} \sum_{m=0}^{M} \frac{(-\alpha t)^m}{m!}\right). \end{aligned} \tag{26}$$

The structure of the generator matrix is used for the computation of the vector τ^m as described in (27) for a subvector $\tau^m_{f(n)}$. The tensor sums/products in the equation can be transformed similarly to (18), the only difference is that the diagonal elements need not be skipped (i.e., the operator $\sim$ is not needed).

$$\tau^m_{f(n)} = \tau^{m-1}_{f(n)} + \frac{1.0}{\alpha}\left(\tau^{m-1}_{f(n)} Q^{f(n)}_{IG} + \sum_{n' \in \Gamma(n)} \tau^{m-1}_{f(n')} Q^{f(n'),f(n)}_{EG}\right). \tag{27}$$

Although transient solutions have been introduced here on the original generator, it should be obvious that the approach works in exactly the same way when modifying the matrix by making some states absorbing. Such a modification is used to calculate quantities like first passage time distributions and probability flows (see Gross and Miller [14], or Reibman and Trivedi [21] for applications).

Like in the stationary case, aggregation can be introduced for QNs with loosely coupled *LLMs* yielding a method similar to the one introduced by Bobbio and Trivedi [3]. However, since this approach is an approximative method, we will not introduce it in detail here.

In the remainder of this section a brief comparison of the space and time requirements of the new technique and the conventional approach on the overall generator is given. To simplify the analysis we assume that the cycle probabilities $r(i,k,i,k,\boldsymbol{n})$ are all zero. However, non-zero cycle probabilities normally have only minor effects.

Since the generator matrices of Markov chains resulting from QNs are normally very large and sparse, the use of sparse matrix storage schemes is necessary. Therefore the number of non-zero elements is proportional to the storage needed for the matrix. Let $nz(\boldsymbol{Q})$ be the number of non-zero elements of $\boldsymbol{Q}$. The number of non-zeros of $\boldsymbol{Q}_G(nz(\boldsymbol{Q}_G))$ equals

$$\sum_{\boldsymbol{n}\in Z_0}\left(\sum_{j=1}^{J}((nz(\boldsymbol{Q}_j^{\boldsymbol{n}_j})-\|Z_j(\boldsymbol{n}_j)\|)\prod_{i=1,i\neq j}^{J}\|Z_j(\boldsymbol{n}_j)\| + \prod_{i=1}^{J}\|Z_j(\boldsymbol{n}_j)\| + \sum_{\boldsymbol{n}'\in\Upsilon(\boldsymbol{n})}(nz(\boldsymbol{S}_i^{\boldsymbol{n}_i-k})nz(\boldsymbol{U}_j^{\boldsymbol{n}'_j-l})\prod_{\iota=1,\iota\neq i,j}^{J}\|Z_\iota(\boldsymbol{n}_\iota)\|)\right), \tag{28}$$

where $\Upsilon(\boldsymbol{n})=\{\boldsymbol{n}'|\boldsymbol{n}'\in\Gamma(\boldsymbol{n})\text{ and }p(f(\boldsymbol{n}'),f(\boldsymbol{n}))>0.0\}$.

The first term in the above equation includes the non-zeros of $\boldsymbol{Q}_{IG}$, the second the diagonal elements of $\boldsymbol{Q}_{IG}$ and the third the non-zeros of $\boldsymbol{Q}_{EG}$. The number of elements to be stored for the isolated matrices is given by

$$\sum_{j=1}^{J}\left(\sum_{\boldsymbol{n}_j\in NM_j}nz(\boldsymbol{Q}_j^{\boldsymbol{n}_j})+\sum_{k,\boldsymbol{n}_j(k)>0}(nz(\boldsymbol{S}_j^{\boldsymbol{n}_j-k})+nz(\boldsymbol{U}_j^{\boldsymbol{n}_j-k}))\right). \tag{29}$$

Additionally, the routing probabilities $r(i,k,j,l,\boldsymbol{n})$ have to be stored, but the number of routing probabilities between the *LLMs* is normally negligible compared with the storage of the *LLM* matrices. Comparing (28) and (29), one can notice that the memory needed for the isolated matrices is only a fraction of the memory for the generator matrix. In both cases additionally the iteration and solution vectors have to be stored. For the new techniques the memory for these vectors becomes the limiting factor, whereas conventional techniques are limited by the memory for the generator matrix. This difference allows the handling of models which are larger, approximately by the order a magnitude, of course, depending on the concrete model structure.

The number of operations needed for a vector matrix multiplication is in the con-

ventional case proportional to the number of non-zeros in Q_G. The number of operations for a tensor based vector matrix multiplication is proportional to

$$\sum_{n\in Z_0}\left(\sum_{j=1}^{J}(nz(Q_j^{n_j})\prod_{i=1,i\neq j}^{J}\|Z_j(n_j)\|\right) + \sum_{n'\in\Upsilon(n)}((nz(S_i^{n_i-k})\|Z_j(n'_j)\| + nz(U_j^{n'_j-l})\|Z_i(n_i)\|)\prod_{\iota=1,\iota\neq i,j}^{J}\|Z_\iota(n_\iota)\|)). \quad (30)$$

The number of operations is not reduced by the new approach and a single operation is slightly more complex since address transformations are performed. Therefore we cannot expect faster solutions for small models. However, for larger models the time for solution is largely influenced by the time needed for paging and the new techniques allow much larger models to be held in primary memory, yielding a significant faster solution. Additionally, the structure of the matrices and vectors allows a natural parallelization of the solution. The conventional and the new approach can be combined by constructing the generator matrix only partially until primary memory is occupied. This technique provides the possibility of choosing an optimal relation between the available space and the efficiency of the solution.

5. An example

In this section we report the analysis of an example model to compare the new solution approach with the conventional one. The example model used here is a modified version of a multi-echelon repairable item inventory system, which had been analyzed in great detail by Gross et al. [13], results are documented in the cited paper and the references therein. It should be noticed that the model is analyzed by Gross et al. [13] using software which is particularly designed for this special kind of problem, and the analysis is performed on very powerful machines like Cray 1.0 or Cyber 1.9. Our techniques are implemented in standard C on a workstation to handle a general class of submodels, which are defined using the MACOM package [18]. Of course, matrices are all stored as sparse matrices (i.e., only non-zero elements are considered), but the structure of a particular model is not further used. To get comparison results we analyze each configuration of the model with the new techniques and by building Q_G (of course, using also sparse matrix storage schemes) and analyzing the system with a conventional Gauss–Seidel solver.

Before results are documented the example should be explained in some detail (see also Gross et al. [13] for the slightly different original version of the model). The *HLM* contains two bases and a depot with a central repair facility (see fig. 1). Items in one of the bases fail and might need a repair in the depot; in this case the failed item leaves the base and enters the depot. Gross et al. [13] assume zero travel

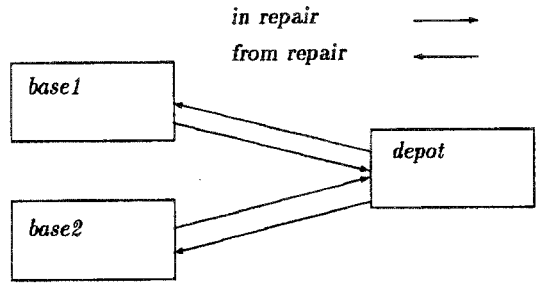

Fig. 1. High level model structure.

time between depot and bases and vice versa, here we assume that traveling from a base to the depot and from the depot to a base takes time and that there exists one channel from each base to the depot and from the depot to each base. Notice that this assumption enlarges the state space of the model significantly. We assume that channels to a *LLM* are modeled inside the *LLM*. If a failed item reaches the depot, it goes into repair and is immediately substituted by a spare, if one is available at the depot. If no spare is available, a backorder is established, which is filled after an item comes out of depot repair. Thus, the behavior of the *HLM* is characterized by two classes of entities traveling between one base and the depot. Items going from the base to the depot are failed and need a depot repair. Items from the depot to the base are repaired and can be used in the base.

We now consider in some more detail the *LLMs*. *base1* and *base2* only differ through the parameter values (see fig. 2). Each base includes three stations with *FCFS* scheduling. Station *work* is a multiserver station with O_b server ($b = 1, 2$), the items actually in service describe the working items in the base (i.e., O_b is the number of operating times at base b). Items waiting at *work* are local sparse. The termination of a service in *work* equals the failing of an item. With probability p_b

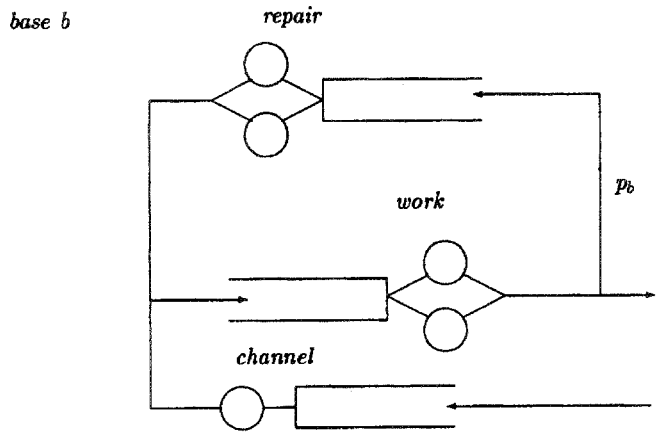

Fig. 2. *base* submodel.

the failed item can be repaired locally in the station *repair*, with probability $1.0 - p_b$ a repair at the depot is necessary; the item leaves the base *LLM*. The probability p_b can depend on the state of the base, however, in the examples documented here we assume a state independent probability. The station *repair* is also a multiserver with C_b server, describing the number of local repair facilities. The third station *channel* is entered by items immediately after entering the *LLM*, the station is a single server and models the communication channel between the *depot* and *base b*. All service times are assumed to be exponential.

The *depot LLM* is shown in fig. 3. Items entering the *depot* first enter one of the single server stations *channel-b* ($b = 1, 2$) depending on their type identity (i.e., from which base they come). After leaving *channel-b* an item puts a resource in the resource pool *failed* and tries to get a resource from the pool *repaired*. If *repaired* is empty, the item has to wait until a resource in *repaired* becomes available. The repairing of items is done by the entities from the closed chain in *depot*. We assume that C_d entities are available (C_d is the number of depot repair facilities). Entities of this type perform a closed loop in the *depot*, by getting a resource from *failed* running through an infinite server *repair* modelling the time needed for repair and putting a resource in *repaired*. If no resources in *failed* are available, entities wait for the next failed item. If a resource in *repaired* becomes available and items from both bases are waiting, the base with the largest percent deficit (i.e., largest value *waiting items*$/N_b$) gets the resource. We assume that *depot* has N_d spares (i.e., at the beginning *repaired* contains N_d resources). The service times in the *depot* are all exponentially distributed.

Various configurations of the model with state spaces ranging from 112 to 2212210 states have been analyzed. The definition of the model parameters is given in table 1. Table 2 includes the configurations which have been analyzed. In all con-

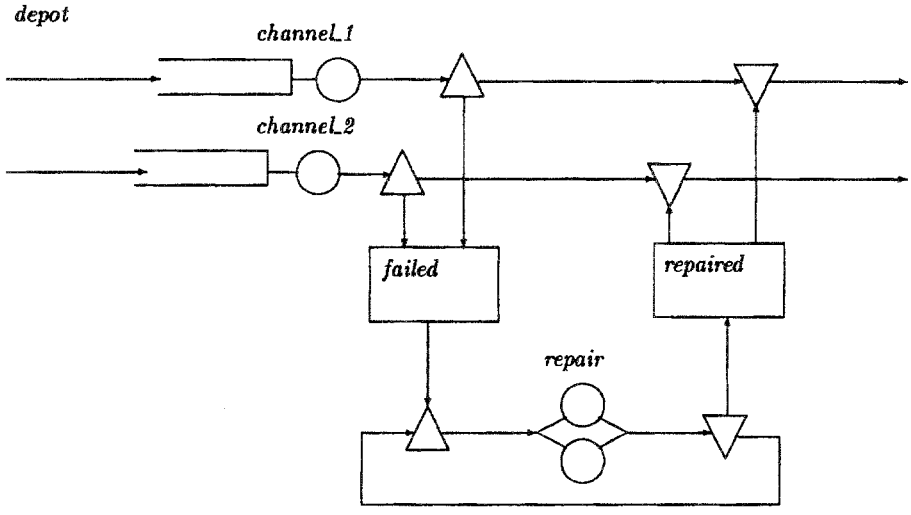

Fig. 3. *depot* submodel.

Table 1
Parameter definitions.

N_b	number of items in base b ($b = 1,2$)
O_b	number of operating items in base b
C_b	number of repair channels in base b
N_d	number of depot sparse
C_d	number of depot repair channels
p_b	probability that a failed item at base b is locally repairable
λ_b	failure rate base b
μ_b	repair rate base b
ω_b	transfer rate base b – depot and vice versa
N_d	number of depot sparse
C_d	number of depot repair channels
μ_d	depot repair rate

figurations the following values are fixed: $\mu_b = 1.0$, $\mu_d = 0.5$, $\omega_b = 10.0$, $p_b = 0.6$, $\lambda_b = 0.2\,(b = 1,2)$.

The last three columns of table 2 include the number of states (= *#states*), the number of non-zeros of the generator Q_G (= *#nz*1) and the number of non-zeros of the *LLM* and *HLM* matrices (= *#nz*2). With the conventional approach only the configurations 1 to 6 can be handled, in the remaining configurations the size of the generator and the vectors exceeds the capacity of virtual memory. All configurations except the last have been analyzed on a standard workstation with 32MB virtual memory and 16MB primary memory, for the last configuration a machine with 48MB virtual memory has been used. If we compare *#states*, *#nz*1 and *#nz*2, it can be noticed that for larger models *#nz*1 >> *states* >> *#nz*2. This, of course, confirms our theoretical results from the previous section, that the limiting factor in the conventional analysis is *#nz*1, whereas the limiting factor in the structured analysis is *#states*.

Apart from the memory requirements of a solution we are interested in the time requirements. Table 3 compares the conventional solution approach with the struc-

Table 2
Analyzed configurations.

c	N_1	O_1	C_1	N_2	O_2	C_2	N_d	C_d	*#states*	*#nz*1	*#nz*2
1	2	1	1	1	1	1	1	1	1.15e+2	5.20e+2	1.39e+2
2	5	3	2	4	2	2	2	2	1.27e+4	9.35e+4	2.25e+3
3	6	4	3	5	3	3	3	3	4.06e+4	3.20e+5	4.27e+3
4	7	5	3	5	3	3	3	3	6.17e+4	4.98e+5	5.44e+3
5	7	5	3	6	4	3	3	3	9.95e+4	8.24e+5	7.06e+3
6	8	5	3	6	3	3	4	4	1.59e+5	1.35e+6	9.10e+3
7	9	6	4	6	3	3	3	3	2.06e+5	–	1.07e+4
8	10	6	4	8	4	4	4	4	6.84e+5	–	2.03e+4
9	11	9	2	9	7	2	3	4	1.22e+6	–	2.80e+4
10	12	10	2	10	8	2	3	3	2.21e+6	–	3.88e+4

Table 3
Solver performance.

c	Conventional		Structured				
	time1	time2	time1	time2	iterations	time1	time2
1	5.00e–1	1.00e+0	6.00e–1	1.00e+0	40	6.67e–1	1.00e+0
2	7.70e+1	7.90e+1	8.93e+1	9.10e+1	110	9.93e+1	1.01e+2
3	2.40e+2	2.47e+2	2.91e+2	2.96e+2	110	3.23e+2	3.29e+2
4	3.62e+2	3.90e+2	4.44e+2	4.50e+2	130	5.77e+2	5.86e+2
5	1.81e+3	5.46e+3	7.40e+2	7.49e+2	140	1.04e+3	1.06e+3
6	2.99e+3	1.65e+4	1.19e+3	1.20e+3	110	1.32e+3	1.34e+3
7	–	–	1.55e+3	1.56e+3	160	2.47e+3	2.50e+3
8	–	–	6.82e+3	1.42e+4	150	1.02e+4	2.05e+4
9	–	–	1.28e+4	3.25e+4	110	1.41e+4	3.63e+4
10	–	–	1.54e+4	4.75e+4	150	2.26e+4	7.23e+4

tured solution. Columns 2–5 include the times needed to perform 100 iteration steps using the conventional and the new approach plus the time for the set up of the program. The set up time includes the time for reading the *LLM* and *HLM* matrices and in the conventional case also the time to generate Q_G from the isolated matrices. It should be noticed that, even with a conventional approach, it is preferable to start with the hierarchical model and the isolated matrices. Since the time needed for the generation of the isolated matrices and the generation of Q_G from these matrices using tensor operations is for larger models much smaller than the time needed for a straightforward generation of Q_G from a flat model specification. There are two times given in the table. *time1* is the CPU time the process needs, including the time needed for paging. *time2* is the elapsed time of the process running exclusively on the workstation. It should be noticed that the elapsed time is often the more important measure, since the machine is not usable when a process occupying nearly all the memory is running and it is simply the time a user has to wait until he gets results.

Columns 6–8 include the number of iterations and times needed to compute π with an estimated accuracy of $1.0e^{-5}$ starting from the short-circuit initial vector. The estimator for the solution accuracy is based on an estimation of the reduction factor and the assumption of geometrical convergence behavior.

The results show that conventional techniques are faster for small models; however, increasing the size of the model, the new approach becomes more efficient in terms of CPU time and much more efficient in terms of elapsed time. Comparing the results for model 6, the largest one solvable with the conventional approach, we get a speed-up of 2.5 and 15.3 in terms of CPU time and elapsed time, respectively. But the analysis of large models still takes a lot of time, although all configurations analyzed here can be solved during a night, which is often acceptable. Of course, the concrete solution time depends on the number of iterations needed, which depend on the concrete model structure.

Gross et al. [13] state that the bi-conjugate gradient method is more efficient than Gauss–Seidel used here. It should be mentioned that our experiences do not give this impression, especially if Gauss-Seidel is combined with a/d steps. Nevertheless, also bi-conjugate gradient can be used in the given framework, but needs more space, since some additional vectors have to be stored. In the particular example here, the use of the short circuit initial vector yields between 20–40% less iterations than the initial vector with a uniform distribution. The additional integration of a/d steps has only minor effects on the solution effort. However, if the *LLMs* are loosely coupled, a/d steps significantly increase the convergence rate.

We do not give here examples for a transient analysis, since it is obvious that the comparison of the conventional with the new approach yields the same results as in the stationary case.

6. Conclusion

We have presented a new approach for the analysis of hierarchical QNs based on tensor operations and their integration in iterative numerical solution techniques. Such an approach enlarges the size of models which can be solved on a given hardware significantly, since the space requirements of the new approach are mainly influenced by the memory needed to store the iteration and solution vector, rather than the generator matrix. Nevertheless, the solution of QNs with a large number of states is still very time consuming, but it has been shown by means of a large example that even models with more than a million states can be solved during a night or weekend with an acceptable accuracy. The performance of the solution algorithms can often be increased by a priori aggregation or a/d steps during the solution, which are integrated naturally in the structured description.

The class of hierarchical QNs is rather large and of practical importance. Of course, not all models can be transformed into hierarchical models, but it is known from system sciences that many natural and well-designed artificial systems are hierarchically structured, yielding naturally hierarchically structured models.

The presentation of solution techniques in this paper is restricted to one technique for stationary and one for transient analysis. However, other iterative techniques can be and partially have been integrated in the hierarchical model world. Especially we believe that the structure of the matrices and vectors provide a base for the integration of highly parallel solution techniques.

Appendix

A procedure for tensor-based vector matrix multiplication

The following procedure $mult(\pi, Q, l, u)$ performs the multiplication $\pi I_l \otimes Q \otimes I_u$ without generating the matrix resulting from the tensor product. Let:
prob_vector be an array of an appropriate dimension

sparse_matrix a structure including only non-zero elements, elements are linked column-wise
mat_elem is one element of a sparse-matrix
$row(Q)$, $col(Q)$ the number of rows / columns of Q
The description is in C-like pseudo-code:

```
struct prob-vector mult (π, Q, l, u)
        struct prob_vector π; struct sparse_matrix {Q}; int l, u;
{
        int i1, i2, c, row_base, col_base, dim= l * u * col(Q);
        struct mat_elem q;
        struct prob_vector p;

        for (c = 1; c ≤ col(Q); c + +)
        for (q = Q[c] − > first_element; q ≤ Q[c] − > last_element;
        q = q − > successor)                                        {
                row_base = (q − > row_index −1) * u
                row_base = (q − > column_index −1) * u
                for (i1 = 1; i1 ≤ l; i1 + +){
                        for (i2 = 1; i2 ≤ u; i2 + +)
                                p[++col_base]+ = π[++row_base] * q − >value
                        col_base+ = (col(Q) − 1) * u;
                        row_base+ = (row(Q) − 1) * u; }}
        return(p);
}
```

References

[1] F. Baskett, K.M. Chandy, R.R. Muntz and G.F. Palacois, Open, closed and mixed networks of queues with different classes of customers, J. ACM 22 (1975) 248–260.

[2] H. Beilner, J. Mäter and N. Weissenberg, Towards a performance modelling environment: News on HIT, in: *Proc. 4th Int. Conf. on Modelling Techniques and Tools for Performance Evaluation*, ed. R. Puijanger (Plenum, New York, 1988).

[3] A. Bobbio and K.S. Trivedi, An aggregation technique for the transient analysis of stiff Markov chains, IEEE Trans. Comp. 35 (1986) 803–814.

[4] P. Buchholz, *The Structured Analysis of Markovian Models* (in German) (Informatik Fachberichte 282, Springer, Berlin, 1991).

[5] P. Buchholz, The numerical analysis of hierarchical queueing network models, in: *Proc. 6th GI/ITG Fachtagung Messung, Modellierung und Bewertung von Rechensystemen*, eds. A. Lehmann and F. Lehmann (Informatik Fachberichte 286, Springer, Berlin, 1991).

[6] P. Buchholz, Numerical solution methods based on structured descriptions of Markovian models, in: *Proc. 5th Int. Conf. on Computer Performance Evaluation – Modelling Techniques and Tools*, eds. G. Balbo and G. Serazzi (North-Holland, Amsterdam, 1992).

[7] P. Buchholz, A hierarchical view of GCSPNs and its impact on qualitative and quantitative analysis, J. Parallel and Distributed Comput. 15 (1992) 207–224.

[8] W.L. Cao and W.J. Stewart, Iterative aggregation/disaggregation techniques for nearly uncoupled Markov chains, J. ACM 32 (1985) 702–719.
[9] P.J. Courtois, *Decomposability: Queueing and Computer System Applications* (Academic Press, New York, 1977).
[10] P.J. Courtois, Exact aggregation in queueing networks, in: *Proc. 1st AFCET-SMF Meeting on Appl. Math.*, Ecole Polytechn. Palaiseau, France (1978).
[11] M. Davio, Kronecker products and shuffle algebra, IEEE Trans. Comp. 30 (1981) 116–125.
[12] K.J. Gordon, J.F. Kurose, R.F. Gordon and E.A. MacNair, An extensible visual environment for construction and analysis of hierarchically-structured models of resource contention systems, IBM Research Report RC 15100 (1989).
[13] D. Gross, B. Gu and R.M. Soland, The bi- conjugate gradient method for obtaining the steady-state probability distributions of Markovian multi-echelon repairable item inventory systems, in: *Proc. 1st Int. Workshop on the Numerical Solution of Markov Chains*, ed. G.W. Stewart (Marcel Dekker, New York, 1991).
[14] D. Gross and D.R. Miller, The randomization technique as a modelling tool and solution procedure for transient Markov processes, Oper. Res. 32 (1984) 343–361.
[15] L. Kaufman, Matrix methods for queueing problems, SIAM J. Sci. Stat. Comput. 4 (1983) 525–552.
[16] U. Krieger, B. Müller-Clostermann and M. Sczittnick, Modelling and analysis of communication systems based on computational methods for Markov chains, IEEE J. Sel. Areas Commun. 8 (1990) 1630–1648.
[17] B. Müller-Clostermann, NUMAS: A tool for the numerical modelling of computer systems, in ref. [20].
[18] B. Müller-Clostermann and M. Sczittnick, MACOM – A tool for the Markovian analysis of communication systems, in: *Proc. 4th Int. Conf. on Data Communication Systems and their Performance*, Barcelona (1990).
[19] B. Plateau, On the stochastic structure of parallelism and synchronisation models for distributed algorithms, in: *Proc. ACM Sigmetrics Conf. on Measurement and Modelling of Computer Systems*, Austin (1985).
[20] D. Potier, ed., *Modelling Techniques and Tools for Performance Evaluation* (North-Holland, Amsterdam, 1985).
[21] A. Reibman and K.S. Trivedi, Numerical transient analysis of Markov models, Comput. Oper. Res. 15 (1988) 19–36.
[22] C.H. Sauer and E.A. McNair, The evolution of the research queueing package, in ref. [20].
[23] M. Veran and D. Potier, QNAP2: A portable environment for queueing systems modelling, in ref. [20].