Schedulability Analysis of Multiprocessor Real-Time Applications with Stochastic Task Execution Times

Sorin Manolache, Petru Eles, Zebo Peng
Department of Computer and Information Science, Linkoping University, Sweden
{sorma, petel, zebpe}@ida.liu.se

Abstract

This paper presents an approach to the analysis of task sets implemented on multiprocessor systems, when the task execution times are specified as generalized probability distributions. Because of the extreme complexity of the problem, an exact solution is practically impossible to be obtained even for toy examples. Therefore, our methodology is based on approximating the generalized probability distributions of execution times by Coxian distributions of exponentials. Thus, we transform the generalized semi-Markov process, corresponding to the initial problem, into a continuous Markov chain (CTMC) which, however, is extremely large and, hence, most often is impossible to be stored in memory. We have elaborated a solution which allows to generate and analyze the CTMC in an efficient way, such that only a small part has to be stored at a given time. Several experiments investigate the impact of various parameters on complexity, in terms of time and memory, as well as the trade-offs regarding the accuracy of generated results.

1 Introduction

Systems controlled by embedded electronics become indispensable in our lives and can be found in avionics, automotive industry, home appliances, medicine, telecommunication industry etc. [4]. Due to the application nature itself, as well as to the high demands in terms of computation power and constraints, such as cost and energy dissipation, these systems are mainly custom designed heterogeneous multiprocessor platforms. Their high complexity makes the design process a challenging activity. Hence, an accurate and efficient design process is the key to cope with the high time-to-market pressure. The enormous design space implies that accurate estimation and analysis tools in all design stages are of capital importance for guiding the designer and reducing the design iterations and cost.

The present work focuses on an analytic approach to schedulability analysis of multiprocessor real-time systems. We consider applications modelled as sets of task graphs with the tasks statically mapped on a set of processors. Most of the previous work in this area considers worst case execution times (WCET) [5][17] and provides answers whether the tasks will meet or not all their deadlines. Such an approach is well suited for particular critical applications. However, it leads to expensive and inefficient implementations in the case of many application classes like soft real-time systems and multimedia applications.

Our work considers the case when the task execution times are of stochastic nature and their probabilities are distributed according to given arbitrary time probability distribution functions. Because meeting a deadline in this case becomes also a stochastic event, our method generates the probability of task deadline misses.

The variability of the task execution time may stem from several sources: input data characteristics (especially in differently coded video frames), hardware architecture hazards (caches and pipelines), environmental factors (network load), or insufficient knowledge regarding the design (a task running on a not yet manufactured processor, for example).

Leung and Whitehead [14] showed that the schedulability analysis is an NP-complete problem in the case of fixed task execution times and more than two processors. Obviously, the problem is much more challenging in the case of stochastic task execution times.

The sequel of the paper is organized as follows: The next section surveys some related approaches. Section 3 formulates the problem. Section 4 presents the approach outline while Section 5 introduces our intermediate representation based on Generalized Stochastic Petri Nets. Section 6 presents the technique for approximation of arbitrary probability distributions and Section 7 details our analysis method. Experiments are presented in Section 8. The last section draws the conclusions.

2 Related work

Lehoczky has pioneered the “heavy traffic” school of thought in the area of real-time queueing [12][13]. The theory was later extended by Harrison [7], Williams [18] and others. The application is modelled as a multiclass queueing network. This network behaves as a reflected Brownian motion with drift under heavy traffic conditions, i.e. when the processor utilizations approach 1, and therefore it has a simple solution.

0-7803-7607-2/02/$17.00

This approach, to our knowledge, fails yet to handle systems where a task has more than one immediate successor task. Moreover, the heavy traffic assumption implies an almost infinite queue in the case of input distributions with non-negligible variation. This leads to an unacceptably high deadline miss ratio and limits the applicability of such an approach in real-time systems.

In our earlier work, we considered the particular case of monoprocessor systems [15]. The analysis is based on solving a generalized semi-Markov process by means of the auxiliary variable method. Although we concurrently construct and analyse the process, saving a significant amount of memory, this method is of limited applicability to multiprocessor systems due to the exploding complexity.

Hu et al. [8] do not target individual task deadline miss probabilities, but rather assess the feasibility of the entire application, by defining and computing a feasibility metric. However, their approach is limited to monoprocessor systems.

Kalavade and Moghê [9] address the problem of individual task deadline miss probabilities in the context of multiprocessors. Their approach is based on solving the underlying generalized semi-Markov process. In order to be able to manage such a complex problem, they restrict their approach to task execution times that assume values from a discrete set.

Other researchers, such as Kleinberg et al. [11] and Goel et al. [6], apply approximate solutions to problems exhibiting stochastic behaviour but in the context of load balancing, bin packing and knapsack problems. Moreover, the probability distributions they consider are limited to a few very particular cases.

Kim and Shin [10] modelled the application as a queueing network, but restricted the task execution times to exponentially distributed ones which reduces the complexity of the analysis. The tasks were considered to be scheduled according to a particular policy, namely FIFO. The underlying mathematical model is then the appealing continuous time Markov chain (CTMC).

In our approach we address the extremely complex problem of tasks running on multiprocessor systems and which have execution times with arbitrary probability distributions. Our approach is also based on a CTMC in order to keep the appealing character of the solution procedure and to avoid the time and memory consuming convolutions implied by solving the otherwise generalized semi-Markov process (GSPM). Tasks are scheduled according to a fixed priority non-preemptive policy. We overcome the limitation of the exponentially distributed execution time probabilities by approximating arbitrary real-world distributions by means of Coxian distributions, i.e. weighted sums of convoluted exponentials. The resulting CTMC is huge, but by exploiting certain regularities in its structure, we have elaborated a solution such that the infinitesimal generator needs not to be stored explicitly. As a consequence, the memory complexity, which is the most critical aspect of the analysis, is drastically reduced. This makes the method applicable to real-world applications.

3 Problem formulation

Informally, the problem can be formulated as follows: given a set of task graphs where the task execution time probabilities are distributed according to given arbitrary continuous distributions, as well as the mapping of the tasks to a set of processors, the analysis generates the task deadline miss ratios. More formally, the input to the analysis procedure consists of:

- A set of task graphs $TG = \{g_1, \ldots, g_G\}$. Graph nodes are tasks from the set $\mathcal{T}=T \cup CT$, where $T=\{t_1, \ldots, t_M\}$ are the actual tasks specified by the designer, and $CT=\{t_{M+1}, \ldots, t_N\}$ represent communication activities. Graph edges capture data dependencies. A communication task is inserted on each node connecting tasks from the set T, that are mapped to different processors. In the paper, where we use the term “task”, without any particular specification, we refer to both actual tasks from the set T and communication tasks;
- A set of processors $P = \{p_1, p_2, \ldots, p_r\}$, and a set of buses $BU=\{l_{r+1}, \ldots, l_s\}$;
- Two surjective mappings, $MapP : T \rightarrow P$, mapping tasks on processors, and $MapB : CT \rightarrow BU$, mapping communication tasks on buses;
- A set of continuous execution time probability distribution functions $E = \{e_1, \ldots, e_N\}$, $e_i : [0{:}\infty) \rightarrow \Re$, statistically independent, where e_i is the execution time (communication time in the case of communication tasks) probability distribution function of task t_i;
- A mapping that associates a static priority to each task, $Prior : T \rightarrow \aleph$; two tasks mapped on the same processor are not allowed to have the same priority[1];
- A set of periods, $A = \{a_1, a_2 \ldots, a_G\}$, where a_i is the period of task graph g_i;
- Each task graph has an associated deadline which equals its period;
- If all deadlines are satisfied, there exists at most one active instantiation of a task graph in the system at a certain moment. If the execution of any task graph g_i exceeds its deadline, then and only then two or more instantiations of g_i can be simultaneously active at a time. A set of integers $B=\{b_1, \ldots, b_G\}$ is given by the designer, where b_i indicates the maximum number of simultaneously active instantiations of the task graph g_i that are acceptable. If the number of instantiations of g_i equals b_i when a new instantiation of g_i arrives, then the new instantiation will be rejected and a new deadline miss will be noted.

The analysis outputs the deadline miss ratios of the task graphs, $F = \{f_1, f_2, \ldots, f_G\}$,

$$f_i = \lim_{t \to \infty} \frac{D_i(t)}{A(t)}$$

where $D_i(t)$ denotes the number of missed deadlines of graph g_i until time t, and $A_i(t)$ denotes the number of arrivals of graph g_i until time t.

4 Approach outline

The underlying mathematical model of the application to be analysed is the stochastic process. The process has to be constructed and analysed in order to extract the desired performance metrics. When considering arbitrary execution time probability distribution functions (ETPDFs), the resulting process is a generalized semi-Markov process, making the analysis extremely demanding in terms of memory and time. If the execution time probabilities were exponentially distributed, as assumed for instance by Kim and Shin [10], the process would be a CTMC which is easier to solve.

The outline of our approach is depicted in Figure 1. As a first step, we generate a model of the application as a Generalized Stochastic Petri Net (GSPN). We use this term in a broader sense than the one defined by Balbo [1], allowing arbitrary probability distributions for the firing delays of the timed transitions. This step is detailed in the next section.

The second step implies the approximation of the arbitrary real-world ETPDFs with Coxian distributions, i.e. weighted sums of convoluted

1.We consider that bus conflicts are arbitrated based on priorities associated to messages (as is the case, for example, with the popular CAN bus[2])

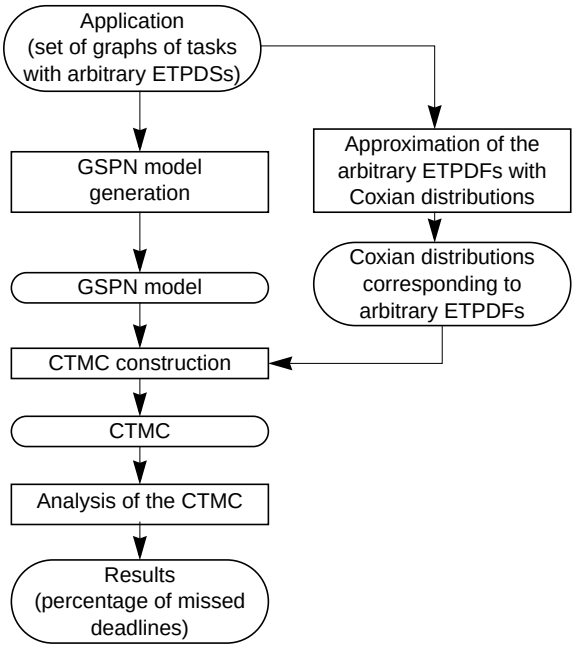

Figure 1. Approach outline

exponential distributions. Some details regarding Coxian distributions and the approximation process follow in Section 6.

In the third step, the tangible reachability graph of the GSPN is obtained. Its nodes correspond to states in a generalized semi-Markov process (GSMP). Directly analysing this process is practically impossible (because of time and memory complexity) for even small toy examples if they are implemented on multiprocessor systems. Therefore, the states of this process are substituted by sets of states based on the approximations obtained in the second step. The transitions of the GSMP are substituted by transitions with exponentially distributed firing interval probabilities from the Coxian distributions. What results is a CTMC, however much larger than the GSMP. The construction procedure of the CTMC is detailed in Section 7.

As a last step, the obtained CTMC is solved and the performance metrics extracted.

5 Intermediate model generation

As a first step, starting from the task graph model given by the designer, an intermediate model based on Generalized Stochastic Petri Nets (GSPN) [1] is generated. Such a model allows an efficient and elegant capturing of the characteristics of the application and of the scheduling policy. It constitutes also an appropriate starting point for the generation of the CTMC to be discussed in the following sections.

A classical GSPN, as introduced by Balbo [1], contains timed transitions with exponentially distributed firing delay probabilities and immediate transitions, with a deterministic zero firing delay. The immediate transitions may have associated priorities. A *tangible marking* is one in which no immediate transitions are enabled. Such a marking can be directly reached from another tangible marking by firing exactly one timed transition followed by a possibly empty sequence of immediate transition firings, until no more immediate transitions are enabled. The tangible reachability graph (TRG) contains the tangible markings of the

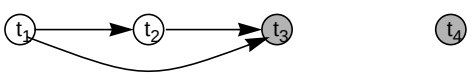

Figure 2. Task graphs

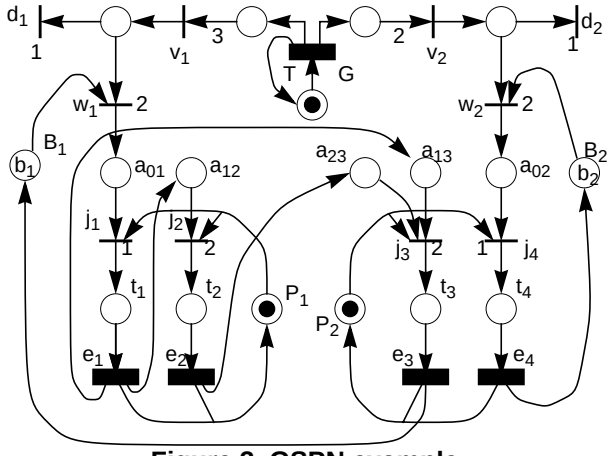

Figure 3. GSPN example

GSPN. An edge in the TRG is labelled with the timed transition that triggers the marking change. Each marking in the TRG corresponds to a state in the underlying stochastic process. If all the timed transitions have exponentially distributed firing delay probabilities, as it is the case in the classical definition of the GSPN, then the underlying stochastic process is a CTMC.

We use the term GSPN in a broader sense, allowing arbitrary probability distributions of the transition firing delays. In this case, the TRG of the net corresponds to a generalized semi-Markov process.

We illustrate the construction of the GSPN based on an example. Let us consider the task graphs in Figure 2. Tasks t_1, t_2 and t_3 form graph g_1 while g_2 consists of task t_4. t_1 and t_2 are mapped on processor p_1 and t_3 and t_4 on processor p_2. The task priorities are 1, 2, 2, 1 respectively. The task graph g_1 has period π_1 and g_2 has period π_2. For simplicity, in this example, we ignore the communication tasks. The GSPN corresponding to the example is depicted in Figure 3. Timed transitions are depicted as solid rectangles. Immediate transitions appear as lines possibly annotated by the associated priority. If a timed transition e_i is enabled, it means that an instantiation of the task t_i is running. The probability distribution of the firing delay of transition e_i is equal to the ETPDF of task t_i. As soon as e_i fires, it means that the instantiation of t_i completed execution and leaves the system. The task priorities are modelled by prioritizing the immediate transitions j_i.

The periodic arrival of graph instantiations is modelled by means of the transition G with the deterministic delay T. G fires every T time units, where T is the greatest common divisor of the graph periods. As soon as G has fired π_i/T times, the transition v_i fires and a new instantiation of task graph g_i demands execution. (In our example, we considered π_1/T=3 and π_2/T=2.) If there are already b_i active instantiations of g_i in the system, where b_i is the initial marking of place B_i, then B_i is not marked, and the transition d_i fires, meaning that the new graph instantiation is rejected. Otherwise, w_i fires, a token from B_i is consumed and the new instantiation is accepted in the system. The transitions w_i have a higher priority than the transitions d_i, as indicated by the integers near the transitions, so an instantiation is always accepted if possible.

The mutual exclusion of the execution of tasks mapped on the same processor is modelled by means of the places P_1 and P_2. The data dependencies among the tasks are modelled by the arcs $e_2 \rightarrow a_{23}$, $e_1 \rightarrow a_{13}$ and $e_1 \rightarrow a_{12}$. Once a task graph instantiation leaves the system (the sink nodes in the task graph complete execution), a token is added to B_i.

Consider two arbitrary tangible markings M_1 and M_2, such that M_2 is directly reachable from M_1 by firing the timed transition U. The sets of timed transitions e_i that are enabled in M_1 and M_2 are W_1 and W_2. (Observe that W_1 and W_2 can be seen as sets of tasks, as each transition e_i corresponds to a task t_i.) An important property that is easily detected

from the Petri Net is that the net does not contain competitively enabled timed transitions. This means that no transition in W_1 can be disabled when firing U, except possibly U itself ($W_1 \setminus \{U\} \subseteq W_2$). In other words, if the cardinality of $W_1 \setminus W_2$ is greater than 1, then we are guaranteed that M_2 is not directly reachable from M_1. In the underlying stochastic process, this implies that there can be no edge from a state in which a set W_1 of tasks is running to a state in which a set W_2 of tasks is running, if $|W_1 \setminus W_2| > 1$. This property is used in Section 7 to determine the states of the underlying stochastic process that might be directly connected with an edge labelled with a given transition.

The underlying GSMP is extracted from the Petri Net by building its TRG. The GSMP is then approximated with a CTMC by replacing the arbitrary probability distributions of the task execution times (firing delay probability distributions of the timed transitions e_i) with Coxian distributions. This is further discussed in Section 6 and Section 7.

6 Coxian distribution approximation

Coxian distributions were introduced by Cox [3] in the context of queueing theory. A Coxian distribution of r stages is a weighted sum of convoluted exponential distributions. The Laplace transform of the probability density of a Coxian distribution with r stages is given below:

$$X s) = \sum_{i=1}^{r} \alpha_i \cdot \prod_{k=1}^{i-1} (1-\alpha_i) \cdot \prod_{k=1}^{i} \frac{\mu_k}{s+\mu_k}$$

X(s) is a strictly proper rational transform, implying that the Coxian distribution may approximate a fairly large class of arbitrary distributions with an arbitrary accuracy provided a sufficiently large r.

Figure 4 illustrates the way we are using Coxian distributions in our approach. Let us consider the timed transition with a certain probability distribution of its firing delay in Figure 4a. This transition can be replaced by the Petri Net in Figure 4b, where hollow rectangles represent timed transitions with exponential firing delay probability distribution. The annotations near those transitions indicate their average firing rate. In this example, three stages have been used for approximation.

Practically, the approximation problem can be formulated as follows: given an arbitrary probability distribution, find μ_i, $i=\overline{1,r}$, and α_i, $i=\overline{1,r-1}$ ($\alpha_r=1$) such that the quality of approximation of the given distribution by the Coxian distribution with r stages is maximized. This is usually done in the complex space by minimizing the distance between the Fourier transform X(jω) of the Coxian distribution and the computed Fourier transform of the distribution to be approximated. The minimization is a typical interpolation problem and can be solved by various numerical methods [16]. We use a simulated annealing approach that minimizes the difference of only a few most significant harmonics of the Fourier transforms which is very fast, if provided with a good initial solution. We choose the initial solution in such way that the first moment of the real and approximated distribution coincide.

By replacing all generalized transitions of the type depicted in Figure 4a with subnets of the type depicted in Figure 4b the GSMP underlying the Petri Net becomes a CTMC. It is obvious that the introduced additional places trigger an explosion in the TRG and implicitly in the resulted CTMC. The next section details on how to efficiently handle such a complexity increase.

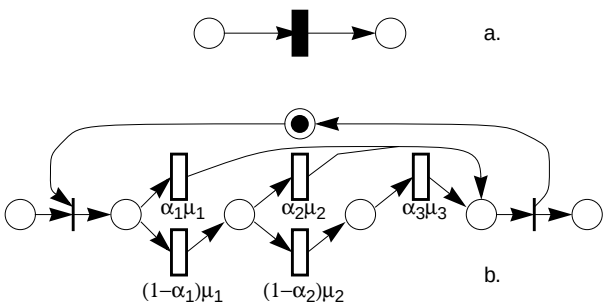

Figure 4. Coxian approximation with 3 stages

7 CTMC construction and analysis

Let S be the set of states of the GSMP underlying the Petri Net. Let $M=[m_{ij}]$ be a square matrix of size $|S| \times |S|$ where m_{ij} denotes the average transition rate from state i to state j in the GSMP, i.e. the average firing rate of the transition that labels the edge from i to j in the GSMP. We first partition the set of states S in clusters such that states in the same cluster have outgoing edges labelled with the same set of transitions. A cluster is identified by a binary combination that indicates the set of transitions that are enabled in the particular cluster. The clusters are sorted according to their corresponding binary combination and the states in the same cluster are consecutively numbered.

Consider an example application with four independent tasks, each mapped on a different processor. In this case, 16 clusters can be formed, each corresponding to a possible combination of simultaneously running tasks. Note that if the tasks were not independent, the number of combinations of simultaneously running tasks, and implicitly of clusters, would be smaller. Figure 5 depicts the matrix M corresponding to the GSMP of this example application. The rows and columns in the figure do not correspond to individual rows and columns in M. Each row and column in Figure 5 corresponds to one cluster of states. Thus, each cell in the figure does not correspond to a matrix element but to a submatrix. The row labelled with 1101, for example, as well as the column labelled with the same binary number, indicate that the tasks 1, 2, and 4 are running in the states belonging to the cluster labelled with 1101. The shaded cells of M indicate those submatrices that may contain non-zero elements. The blank ones are null submatrices. For example, one such null submatrix appears at the intersection of row 1101 and column 1000. Due to the non-preemption assumption, a task arrival or departure event may not stop the running of another task. If the submatrix (1101, 1000) had non-zero elements it would indicate that an event in a state where the tasks 1, 2, and 4 are running, triggers a transition to a state where only the task 1 is running, and *two* of the previously running tasks are not running anymore. This is not possible in the case of non-preemption. The submatrix (1000, 0000) may contain non-zero elements, because, if the task 1 completes execution, the stochastic process may transit to a state in which no task is running.

Once we have the matrix M corresponding to the underlying GSMP, the next step is the generation of the CTMC using the Coxian distribution for approximation of arbitrary probability distributions of transition delays. When using the Coxian approximation, a set of new states is introduced for each state in S resulting an expanded state space S', the

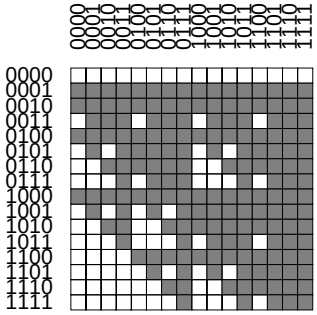

Figure 5. The matrix corresponding to a GSMP

Figure 6. Part of a GSMP

state space of the approximating CTMC. We have to construct a matrix Q of size |S'|×|S'|, the so called infinitesimal generator of the approximating CTMC. The construction of Q is done cell-wise: for each cell of matrix M, a corresponding cell of matrix Q will be generated. Furthermore, blank cells (null submatrices) of M will result in null submatrices of Q, while shaded cells will result in submatrices that contain at least one non-zero element. A cell (U, V) of Q will be of size X×Y, where

$$X = |U| \cdot \prod_{i \in EnU} r_i \qquad Y = |V| \cdot \prod_{i \in EnV} r_i$$

and U and V are clusters of states, EnU={k : transition t_k is enabled in U}, EnV={k : transition t_k is enabled in V}, and r_k is the number of stages we use in the Coxian approximation of the ETPDF of task t_k.

We will illustrate the construction of a cell in Q from a cell in M using an example. We consider a cell on the main diagonal, as it is the most complex case. Let us consider three states in the GSMP as depicted in Figure 6. Two tasks, u and v, are running in the states X and Y. These two states belong to the same cluster, labelled with 11. Only task v is running in state Z. State Z belongs to cluster labelled with 10. If task v finishes running in state X, a transition to state Y occurs in the GSMP. This corresponds to the situation when a new instantiation of v becomes active immediately after the completion of a previous one. When task u finishes running in state X, a transition to state Z occurs in the GSMP. This corresponds to the situation when a new instantiation of u is not immediately activated after the completion of a previous one. Consider that the probability distribution of the execution time of task v is approximated with the three stage Coxian distribution depicted in Figure 4b and that of u is approximated with the two stage Coxian distribution depicted in Figure 7. The resulting CTMC corresponding to the GSMP in Figure 6 is depicted in Figure 8. The edges between the states are labelled with the average firing rates of the transitions of the Coxian distributions.

Let us construct the cell $Q_{(11),(11)}$ on the main diagonal of Q, situated at the intersection of the row and column corresponding to the cluster labelled with 11. The cell is depicted in Figure 9. The matrix $Q_{(11),(11)}$ contains the average transition rates between the states X_{ij} and Y_{ij} of the CTMC in Figure 8 (only states X and Y belong to the cluster labelled with 11). The observed regularity in the structure of stochastic process in Figure 8 is reflected in the expression of $Q_{(11),(11)}$ as shown in Figure 9. Because Q is a generator matrix (sum of row elements equals 0), there appear some negative elements on the main diagonal that do not correspond to transitions in the chain depicted in Figure 8. The expression of $Q_{(11),(11)}$ is given below:

$$Q_{(11),(11)} = (A_u \oplus A_v) \otimes I_{|11|} + I_{r_u} \otimes B_v \otimes e_{r_v} \otimes D_v$$

where

$$A_u = \begin{bmatrix} -\lambda_1 & (1-\beta_1)\lambda_1 \\ 0 & -\lambda_2 \end{bmatrix} \qquad B_v = \begin{bmatrix} \alpha_1\mu_1 \\ \alpha_2\mu_2 \\ \alpha_3\mu_3 \end{bmatrix} \qquad D_v = \begin{bmatrix} 0 & 1 \\ 0 & 0 \end{bmatrix}$$

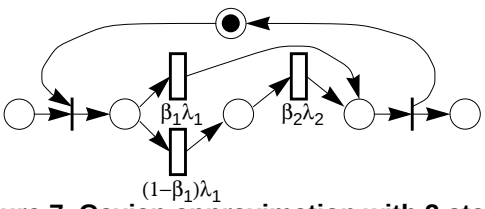

Figure 7. Coxian approximation with 2 stages

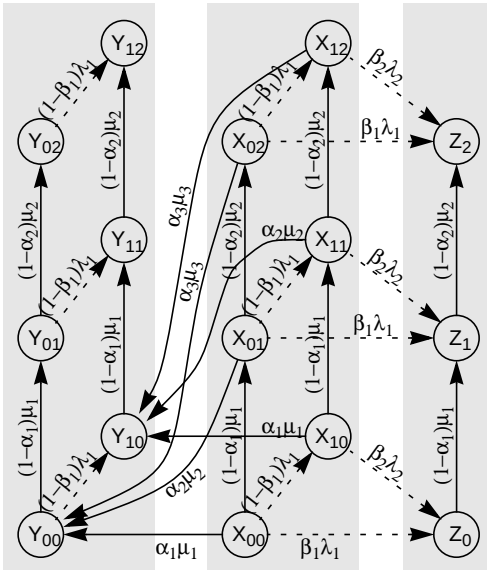

Figure 8. Expanded Markov chain

and

$$A_v = \begin{bmatrix} -\mu_1 & (1-\alpha_1)\mu_1 & 0 \\ 0 & -\mu_2 & (1-\alpha_2)\mu_2 \\ 0 & 0 & -\mu_3 \end{bmatrix} \qquad e_{r_v} = \begin{bmatrix} 1 & 0 & 0 \end{bmatrix}$$

|11| denotes the size of the cluster labelled with 11. I_i is the identity matrix of size i×i, r_i indicates the number of stages of the Coxian distribution that approximates the ETPDF of task t_i. ⊕ and ⊗ are the Kronecker sum and product of matrices, respectively.

In general, a matrix $A_k=[a_{ij}]$ is a $r_k \times r_k$ matrix, and is defined as follows:

$$a_{ij} = \begin{cases} (1-\alpha_{ki})\mu_{ki} & \text{if } j = i+1 \\ -\mu_{ki} & \text{if } j = i \\ 0 & \text{otherwise} \end{cases}$$

where α_{ki} and μ_{ki} characterize the i^{th} stage of the Coxian distribution approximating transition t_k.

	X_{00}	Y_{00}	X_{01}	Y_{01}	X_{02}	Y_{02}	X_{10}	Y_{10}	X_{11}	Y_{11}	X_{12}	Y_{12}	
X_{00}	•	c	a				f						
Y_{00}		•		a				f					a=(1−α1)μ1
X_{01}		d	•		b				f				
Y_{01}				•		b				f			b=(1−α2)μ2
X_{02}		e			•						f		
Y_{02}						•						f	c=α1μ1
X_{10}							•	c	a				
Y_{10}								•		a			d=α2μ2
X_{11}								d	•		b		
Y_{11}										•		b	e=α3μ3
X_{12}								e			•		
Y_{12}												•	f=(1−β1)λ1

• = a negative number such that the matrix is a generator matrix

Figure 9. The cell $Q_{(11),(11)}$ corresponding to the example in Figure 8

A matrix $B_k=[b_{ij}]$ is a $r_k \times 1$ matrix and $b_{i1}=\alpha_{ki}\mu_{ki}$. A matrix $e_{rk}=[e_{ij}]$ is a $1 \times r_k$ matrix and $e_{11}=1$, $e_{1i}=0$, $1<i\leq r_k$. A matrix $D_k=[d_{ij}]$ corresponding to a cluster U of size |U| is a $|U| \times |U|$ matrix defined as follows:

$$d_{ij} = \begin{cases} 1 & \text{if an edge labelled with transition k connects} \\ & \text{the i-th state of the cluster with the j-th state} \\ 0 & \text{otherwise} \end{cases}$$

In general, considering a label U, the cell $Q_{U,U}$ on the main diagonal of Q is obtained as follows:

$$Q_{U,U} = \left(\bigotimes_{i \in U} A_i\right) \otimes I_{|U|} + \sum_{i \in U} \left(\bigotimes_{\substack{j \in U \\ j > i}} I_{r_j}\right) \otimes B_i \otimes e_{r_i} \otimes \left(\bigotimes_{\substack{j \in U \\ j < i}} I_{r_j}\right) \otimes D_i$$

A cell situated at the intersection of the row corresponding to label U with the column corresponding to label V ($U \neq V$) is obtained as follows:

$$Q_{U,V} = \sum_{i \in U} \left(\bigotimes_{j \in V \cup \{i\}} F_{ij}\right) \otimes D_i$$

The matrices F are given by the following expression:

$$F_{ij} = \begin{cases} I_{r_j} & \text{if } j \in U \wedge j \neq i \\ e_{r_j} & \text{if } j \notin U \\ B_i & \text{if } j = i \wedge j \notin V \\ B_i \otimes e_{r_i} & \text{if } j = i \wedge j \in V \end{cases}$$

The solution of the CTMC implies solving for π in the following equation:

$$\pi \cdot Q = 0$$

where π is the steady-state probability vector and Q the infinitesimal generator of the CTMC.

As explained in Section 5, if there are already B_i active instantiations of a task graph g_i in the system, a new arrival is rejected and a deadline miss is noted. In the Petri Net model (see Figure 3), this event corresponds to the sequence of firings of the timed transition G, followed by the immediate transitions v_i and d_i. This sequence of firings corresponds to one transition in the stochastic process. Let us consider that X→Z (Figure 6) is such a transition. The deadline miss *rate* is approximated based on the rates of the transitions $X_{00} \rightarrow Z_0$, $X_{10} \rightarrow Z_0$, $X_{01} \rightarrow Z_1$, $X_{11} \rightarrow Z_1$, and $X_{12} \rightarrow Z_2$ and is given by the expression $(\pi_{x00}+\pi_{x01}+\pi_{x02})\cdot\beta_1\lambda_1 + (\pi_{x10}+\pi_{x11}+\pi_{x12})\cdot\beta_2\lambda_2$, where π_{xij} is the probability of the CTMC being in state ij after an infinite (very long) time. These probabilities are the result of the numerical solution of the CTMC. The deadline miss *ratio* is obtained by multiplying the *rate* with the task graph period.

We conclude this section with a discussion on the size of Q and its implications on analysis time and memory. Consider the cluster labelled 11…1 of S, i.e. the one that contains the largest number of enabled transitions. The largest Q is obtained if the cluster labelled 11…1 dominates all the other clusters of S, in the sense that it contains by far more states than all the other clusters, and that the cell $M_{(11\ldots1),(11\ldots1)}$ contains by far more non-zero entries than all the other cells of M. Thus, a pessimistic upper bound for the number of non-zero elements of Q is given by the expression:

$$|M| \cdot \prod_{i \in E} r_i$$

where $E=\{i : t_i \text{ enabled in the dominant cluster}\}$ and |M| denotes the number of non-zero elements of M, the matrix corresponding to the GSMP. In the context of multiprocessor scheduling, E may have at most s elements (s=number of processors). The above formula shows that the increase in the size of the CTMC, relative to the initial GSMP, is mainly dictated by the number of processors and the number of stages used for the approximation of the probability distribution (which means the degree of expected accuracy). The number N of tasks does not directly induce any growth in terms of the CTMC. However, the structure of the initial GSMP also depends on the number of tasks. The GSMP is reflected in the matrix M and, thus, has an influence on the dimension of the resulted CTMC.

The dimension of matrix Q, as shown above, grows quickly with the number of processing elements and the number of stages used for approximation of the probability distributions. Apparently, the analysis of applications of realistic complexity would be impossible. Fortunately, this is not the case. As can be seen from the expressions of $Q_{U,U}$ and $Q_{U,V}$, the matrix Q is completely specified by means of the matrices A, B, and D_i, hence it needs not be stored explicitly in memory, but its elements are generated on-the-fly during the numerical solving of the CTMC. This leads to a significant saving in memory demand for analysis. Even for large applications, the matrices A, B, and D_i are of negligible size. The limiting factor in terms of memory is only π, the steady-state probability vector, which has to be stored in memory. In the worst case, the vector has

$$|S| \cdot \prod_{i \in E} r_i$$

elements, where |S| is the size of the GSMP. It is easy to observe that π is as large as a row (column) of Q. The effect of the complexity increase induced by the approximation in terms of analysis time can be attenuated by deploying intelligent numerical algorithms for matrix-vector computation. Such algorithms rely on factorizations that exploit the particular structure of Q.

8 Experimental results

We performed four sets of experiments. All were run on an Athlon at 1533 MHz. The first set of experiments investigates the dependency of the analysis time on the number of tasks in the system. Sets of random task graphs were generated, with 9 up to 60 tasks per set. Ten different sets were generated and analysed for each number of tasks per set. The underlying architecture consists of two processors. The dependency between the needed analysis time and the number of tasks is depicted in Figure 10. The analysis time depends on the size of the stochastic process to be analysed as well as on the convergence rate of the numerical solution of the CTMC. The latter explains some non-linearities exhibited in Figure 10. The dependency of the stochastic process size as a function of the number of tasks is plotted in Figure 11.

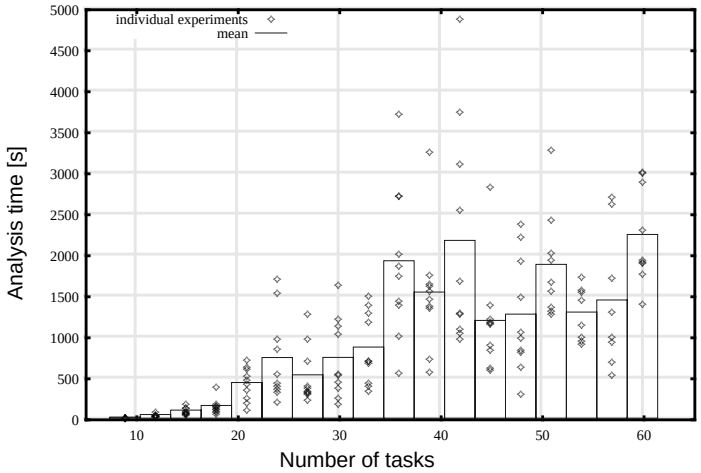

Figure 10. Analysis time vs. no. of tasks

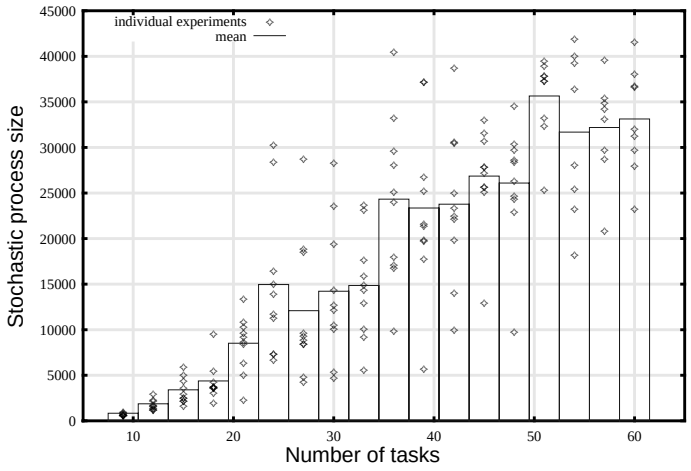

Figure 11. Stochastic process size vs. no. of tasks

In the second set of experiments, we investigated the dependency between the analysis time and the number of processors. Ten different sets of random task graphs were generated. For each of the ten sets, 5 experiments were performed, by allocating the 18 tasks of the task graphs to 2 up to 6 processors. ETPDFs were approximated by using Coxian distributions with 4 stages. The results are plotted in Figure 12.

In the third set of experiments, we investigated the increase in the stochastic process size induced by using different number of stages for approximating the arbitrary ETPDFs. We constructed 98 sets of random task graphs ranging from 10 to 50 tasks mapped on 2 to 4 processors. The ETPDFs were approximated with Coxian distributions using 2 to 6 stages. The results for each type of approximation were averaged over the 98 sets of graphs and the results are plotted in Figure 13. Recall that |S| is the size of the GSMP and |S'| is the much larger size of the CTMC obtained after approximation. As more stages are used for approximation, as larger the CTMC becomes compared to the original GSMP. As shown in Section 7, in the worst case, the growth factor is

$$\prod_{i \in E} r_i$$

As can be seen from Figure 13, the real growth factor is smaller than the theoretical upper bound. It is important to emphasize that the matrix Q corresponding to the CTMC needs not to be stored, but only a vector with the length corresponding to a column of Q. The growth of the vector length with the number of Coxian stages used for approximation can be easily derived from Figure 13. The same is the case with the growth of analysis time, which follows that of the CTMC.

The fourth set of experiments investigates the accuracy of results as a

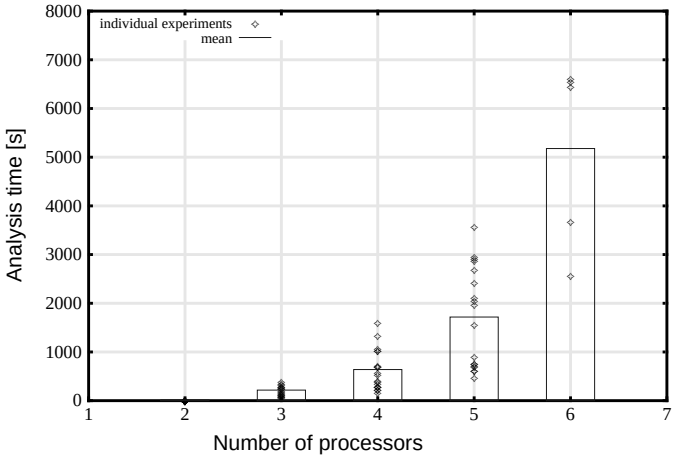

Figure 12. Analysis time vs. no of processors

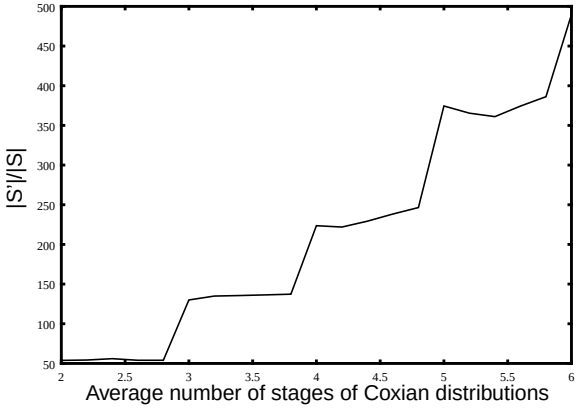

Figure 13. Increase in stochastic process size with number of stages

factor of the number of stages used for approximation. This is an important aspect in deciding on a proper trade-off between quality of the analysis and cost in terms of time and memory. For comparison, we used analysis results obtained with our approach presented in previous work [15]. As mentioned in Section 2, that approach is an exact one based on solving the underlying GSMP. However, it can handle only monoprocessor systems. Therefore, we applied the approach presented in this paper to a monoprocessor example, which has been analysed in four variants using approximations with 2, 3, 4, and 5 stages. The relative error between missed deadline ratios resulted from the analysis using the approximate CTMC and the ones obtained from the exact solution is presented in Table 1. The generalized ETPDF used in this experiment were created by drawing Bezier curves that interpolated randomly generated control points.

Table 1 Accuracy vs. no. of stages

	2 stages	3 stages	4 stages	5 stages
Relative error	8.467%	3.518%	1.071%	-0.4%

It can be observed that good quality results can be already be obtained with a relatively small number of stages.

Finally, we considered an example from the mobile communication area. A set of 8 tasks co-operate in order to decode the digital bursts corresponding to the GSM 900 signalling channel. The 8 tasks are mapped on three processors, one for modulation, one for control and one for digital signal processing. The variability in task execution times has two causes: variability in input data and pipeline hazards in the deeply pipelined DSPs. In the case of 8 tasks, the analysis reported a miss deadline ratio of 0.11% and the analysis took 3 seconds. The ETPDFs were approximated by Coxian distributions with 6 stages. If we attempt to perform the baseband processing of another channel on the same DSP processor, three more tasks are added to the task graph. As a result of the analysis, in this case 10.05% of the deadlines are missed, which is unacceptable according to the application specification.

9 Conclusions

We presented an approach to schedulability analysis of tasks with probabilistically distributed execution times, implemented on multiprocessor systems. The arbitrary probability distributions of the execution times are approximated with Coxian distributions, and the

expanded underlying Markov chain is constructed in a memory efficient manner exploiting the structural regularities of the chain. In this way we have practically pushed the solution of an extremely complex problems to its limits. Our approach also allows to trade-off between time and memory complexity on one side and solution accuracy on the other. It is worth to be mentioned that certain assumptions regarding the analysed systems could be, in principle, further relaxed. Such are, for example, the assumption regarding non-preemptability of tasks, or deadlines equal to the period. Such relaxations, however, seriously increase the complexity of the analysis and, thus, strongly reduce the size of tractable applications.

References

[1] G. Balbo, G. Chiola, G. Franceschinis, G. M. Roet “On the Efficient Construction of the Tangible Reachability Graph of Generalized Stochastic Petri Nets”, Proc 2nd Workshop on Petri Nets and Performance Models, pp. 85-92, 1987

[2] CAN (1990), “Controller Area Network CAN, an In-Vehicle Serial Communication Protocol”, SAE Handbook 1992, SAE Press, pp. 20.341-20.355

[3] D.R. Cox “A Use of Complex Probabilities in the Theory of Stochastic Processes”, Proc. Cambridge Philosophical Society, pp. 313-319, 1955

[4] R. Ernst “Codesign of Embedded Systems: Status and Trends”, IEEE Design & Test of Computers, April-June, 1998, pp. 45-54

[5] C. Fidge “Real-Time Schedulability Tests for Pre-emptive Multitasking”, Real-Time Systems, 14, 61-93 (1998)

[6] A. Goel, P. Indyk “Stochastic Load Balancing and Related Problems”, IEEE FOCS, 1999

[7] J.M. Harrison, V. Nguyen “Brownian Models of Multiclass Queueing Networks: Current Status and Open Problems”, Queueing Systems 13 (1993) 5-40

[8] X.S. Hu, T. Zhou, E.H-M. Sha, “Estimating probabilistic timing performance for real-time embedded systems”, IEEE Trans on VLSI, Vol9-6, Dec 2001, pp. 833-844

[9] A. Kalavade, P. Moghê, “A Tool for Performance Estimation of Networked Embedded End-Systems”, Proc. of DAC 1998, pp 257-262

[10] J. Kim, K.G. Shin, “Execution Time Analysis of Communicating Tasks in Distributed Systems”, IEEE Trans. on Computers, 45 No. 5, May 1996J.

[11] Kleinberg, Y. Rabani, E. Tardos “Allocating Bandwidth for Bursty Connections”, SIAM J. Computing, 30(1), 2000

[12] J.P. Lehoczky “Real-Time Queueing Theory”, Proc. of the 17th IEEE RTSS (1996)

[13] J.P. Lehoczky “Real-Time Queueing Network Theory”, Proc. of the 18th IEEE RTSS (1997)

[14] J.Y.-T. Leung, J. Whitehead “On the Complexity of Fixed-Priority Scheduling of Periodic Real-Time Tasks”, Performance Evaluation, 2(4):237-250, Dec. 1982

[15] S. Manolache, P. Eles, Z. Peng, “Memory and Time Efficient Schedulability Analysis of Task Sets with Stochastic Execution Time”, Euromicro Conf. on Real-Time Systems (2001).

[16] W. H. Press, S. A. Teukolsky, W. T. Vetterling, B. P. Flannery, “Numerical Recipes in C”, Cambridge Univ. Press, 1992

[17] J. A. Stankovic, M. Spuri, M. Di Natale, G. Butazzo, “Implications of Classical Scheduling Results for Real-Time Systems”, IEEE Computer, June 1995

[18] R.J. Williams “Diffusion Approximations for Open Multiclass Queueing Networks: Sufficient Conditions Involving State Space Collapse”, Queueing Systems 30 (1998) 27-88