Schedulability Analysis of Real-Time Systems with Stochastic Task Execution Times

Sorin Manolache
Department of Computer and Information Science, IDA,
Linköping University

Acknowledgements

My gratitude goes to Petru Eles for his guidance and his persevering attempts to lend his sharpness of thought to my writing and thinking. He is the person who compels you not to fall. Zebo Peng is the never-angry-always-diplomatic-often-ironic advisor that I would like to thank.

All of my colleagues in the Embedded Systems Laboratory, its "steady state" members as well as the "transient" ones, made life in our "boxes" enjoyable. I thank them for that.

Also, I would like to thank the administrative staff for their help concerning practical problems.

Last but not least, the few but very good friends provided a great environment for exchanging impressions and for enjoyable, challenging, not seriously taken and arguably fruitful discussions. Thank you.

Sorin Manolache

Abstract

Systems controlled by embedded computers become indispensable in our lives and can be found in avionics, automotive industry, home appliances, medicine, telecommunication industry, mecatronics, space industry, etc. Fast, accurate and flexible performance estimation tools giving feedback to the designer in every design phase are a vital part of a design process capable to produce high quality designs of such embedded systems.

In the past decade, the limitations of models considering fixed (worst case) task execution times have been acknowledged for large application classes within soft real-time systems. A more realistic model considers the tasks having varying execution times with given probability distributions. No restriction has been imposed in this thesis on the particular type of these functions. Considering such a model, with specified task execution time probability distribution functions, an important performance indicator of the system is the expected deadline miss ratio of tasks or task graphs.

This thesis proposes two approaches for obtaining this indicator in an analytic way. The first is an exact one while the second approach provides an approximate solution trading accuracy for analysis speed. While the first approach can efficiently be applied to mono-processor systems, it can handle only very small multi-processor applications because of complexity reasons. The second approach, however, can successfully handle realistic multi-processor applications. Experiments show the efficiency of the proposed techniques.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

This chapter briefly presents the frame of this thesis work, namely the area of embedded real-time systems. The limitations of the hard real-time systems analysis techniques, when applied to soft real-time systems, motivate our focus on developing new analysis techniques for systems with stochastic execution times. The challenges of such an endeavour are discussed and the contribution of the thesis is highlighted. The section concludes by presenting the outline of the rest of the thesis.

1.1 Embedded system design flow

Systems controlled by embedded computers become indispensable in our lives and can be found in avionics, automotive industry, home appliances, medicine, telecommunication industry, mecatronics, space industry, etc. [Ern98].

Very often, these embedded systems are reactive systems, i.e. they are in steady interaction with their environment, acting upon it in a prescribed way as response to stimuli sent from the environment. In most cases, this response has to arrive at a certain time moment or within a prescribed time interval from the moment of the application of the stimulus. Usually, the system must respond to a stimulus before a prescribed relative or absolute deadline. Such systems, in which the correctness of their operation is defined not only in terms of functionality (what) but also in terms of timeliness (when), form the class of real-time systems [But97, KS97, Kop97, BW94]. Real-time systems are further classified in hard and soft real-time systems. In a hard real-time system, breaking a timeliness requirement is intolerable as it may lead to catastrophic consequences. Soft real-time systems are considered as still functioning correctly even if some timeliness requirements are occasionally broken. In a hard real-time system, if not all deadlines are guaranteed to be met, the system is said to be unschedulable.

The nature of real-time embedded systems is typically heterogeneous along multiple dimensions. For example, an application may exhibit data, control and protocol processing characteristics. It may also consist of blocks exhibiting different categories of timeliness requirements. A telecommunication system, for example, contains a soft real-time configuration and management block and a hard real-time subsystem in charge of the actual communications. Another dimension of heterogeneity is given by the environment the system operates in. For example, the stimuli and responses may be of both discrete and continuous nature.

The heterogeneity in the nature of the application itself on one side and, on the other side, constraints such as cost, power dissipation, legacy designs and implementations, as well as non-functional requirements such as reliability, availability, security, and safety, lead to implementations consisting of custom designed heterogeneous multiprocessor platforms. Thus, the system architecture consists typically of programmable processors of various kinds (application specific instruction processors (ASIPs), general purpose processors, DSPs, protocol processors), and dedicated hardware processors (application specific integrated circuits (ASICs), field-programmable gate arrays (FPGAs)) interconnected by means of shared buses, point-to-point links or networks of various types.

Designing such systems implies the deployment of different techniques with roots in system engineering, software engineering, computer architectures, specification languages, formal methods, real-time scheduling, simulation, programming languages, compilation, hardware synthesis, etc. Considering the huge complexity of such a design task, there is an urgent need for automatic tools for design, estimation and synthesis in order to support and guide the designer. A rigorous, disciplined and systematic approach to real-time embedded system design is the only way the designer can cope with the complexity of current and future designs in the context of high time-to-market pressure. Such a design flow is depicted in Figure 1.1 [Ele02].

The design process starts from a less formal specification together with a set of constraints. This initial informal specification is then captured as a more rigorous model formulated in one or possibly several modelling languages [JMEP00]. During the system level design space exploration phase, different architecture, mapping and scheduling alternatives are assessed in order to meet the design requirements and possibly

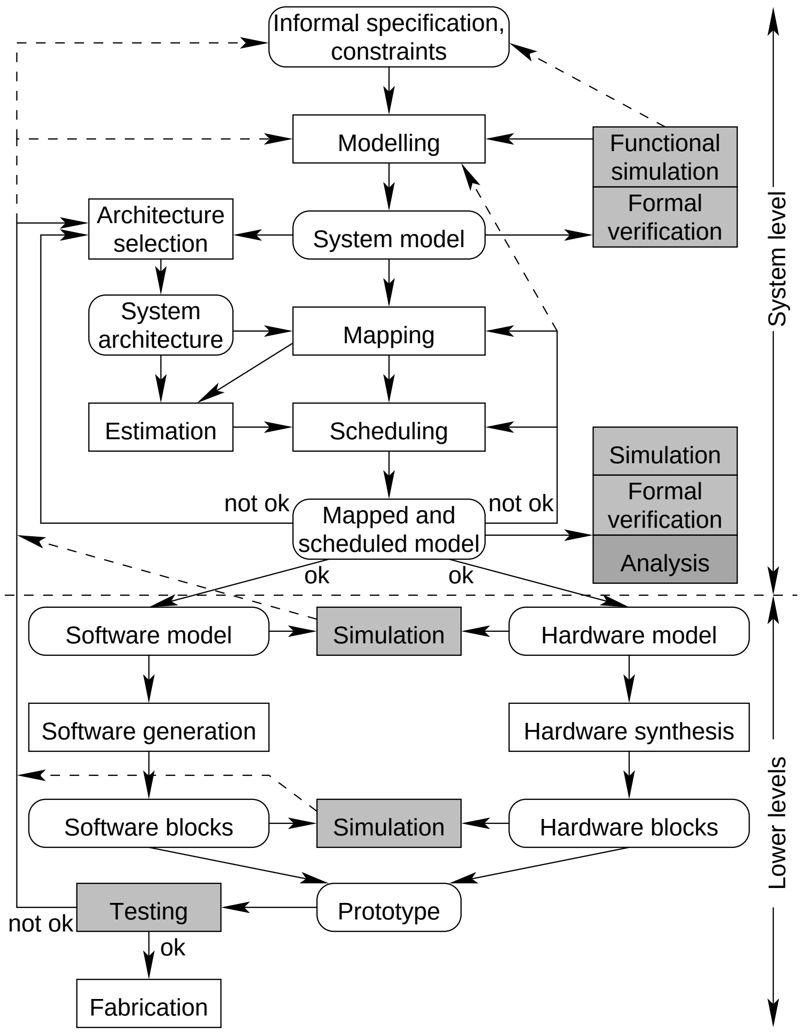


Figure 1.1: Typical design flow

optimise certain indicators. The shaded blocks in the figure denote the activities providing feedback concerning design fitness or performance to the designer. The existence of accurate, fast and flexible automatic tools for performance estimation in every design phase is of capital importance for cutting down design process iterations, time and implicitly cost.

Performance estimation tools can be classified in simulation and analysis tools. Simulation tools are flexible, but there is always the danger that unwanted and extremely rare glitches in behaviour, possibly bringing the system to undesired states, are never observed. The probability of not observing such an existing behaviour can be decreased at the expense of increasing the simulation time. Analysis tools are more precise, but they usually rely on a mathematical formalisation which is sometimes difficult to come up with or to understand by the designer. A further drawback of analysis tools is their often prohibitive running time due to the analysis complexity. A tool that trades, in a designer controlled way, analysis complexity (in terms of analysis time and memory, for example) with analysis accuracy or the degree of insight that it provides, could be a viable solution to the performance estimation problem. Such an approach is the topic of this thesis.

The focus of this thesis is on the analytic performance estimation of soft real-time systems. Given an architecture, a mapping, a task scheduling alternative, and the set of task execution time probability distribution functions, such an analysis (the dark shaded box in Figure 1.1) would provide important and fairly accurate results useful for guiding the designer through the design space.

1.2 Stochastic task execution times

Historically, real-time system research emerged from the need to understand, design, predict, and analyse safety critical applications such as plant control and aircraft control, to name a few. Therefore, the community focused on hard real-time systems, where the only way to ensure that no real-time requirement is broken was to make conservative assumptions about the systems. In hard real-time system analysis, each task instantiation is assumed to run for a worst case time interval, called the worst case execution time (WCET) of the task.

This approach is sometimes the only one applicable for the class of safety critical embedded systems. However, for a large class of soft real-time systems this approach leads to significant underutilisation of computation resources, missing the opportunity to create much cheaper products with low or no perceived service quality reduction. For ex-

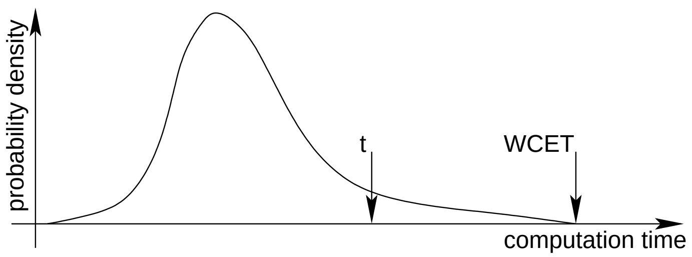


Figure 1.2: Execution time probability density function

ample, multimedia applications like JPEG and MPEG encoding, sound encoding, etc. exhibit this property.

The execution time of a task depends on application dependent, platform dependent, and environment dependent factors. The amount of input data to be processed in each task instantiation as well as its type (pattern, configuration) are application dependent factors. The type of processing unit that executes a task is a platform dependent factor influencing the task execution time. If the time needed for communication with the environment (database lookups, for example) is to be considered as a part of the task execution time, then network load is an example of an environmental factor influencing the task execution time.

Input data amount and type may vary, as for example is the case for differently coded MPEG frames. Platform dependent characteristics, like cache memory behaviour, pipeline stalls, write buffer queues, may also introduce a variation in the task execution time. Thus, obviously, all of the enumerated factors influencing the task execution time may vary. Therefore, a model considering variable execution time would be more realistic as the one considering fixed, worst case execution times. In the most general model, task execution times with arbitrary probability distribution functions are considered. Obviously, the fixed task execution time model is a particular case of such a stochastic one.

Figure 1.2 shows the execution time probability density of such a task. An approach based on a worst case execution time model would implement the task on an expensive system which guarantees the imposed deadline for the worst case situation. This situation, however, will occur with a very small probability. If the nature of the application is such that a certain percentage of deadline misses is affordable, a cheaper system, which still fulfils the imposed quality of service, can be designed. For example, such a cheaper a system would be one that would guarantee the deadlines if the execution time of the task did not exceed a time moment t (see Figure 1.2). It can be seen from the figure, that there is a low probability that the task execution time exceeds t and therefore, missing a deadline is a rare event leading to an acceptable service quality.

In the case of hard real-time systems, the question posed to the performance analysis process is whether the system is schedulable which means if deadlines are guaranteed to be met or not. In the case of stochastic execution times, because of the stochastic nature of the system, such an approach is not reasonable. Therefore, the analysis of such systems does not provide binary classifications but rather fitness estimates, such as measures of the degree to which a system is schedulable. One such measure is the expected deadline miss ratio of each task or task graph and is the focus of this thesis.

1.3 Solution challenges

Because an application with stochastic execution times running on an architecture can be regarded as a system with random character, a natural way to formalise it is to build its underlying stochastic process. The first problem arises as the size of the underlying process is exponential with the number of tasks and the number of processors. An additional complexity dimension is due to the nature of the probability distribution functions. There exist efficient analysis techniques for the case of exponential probability distribution functions. However, this thesis considers arbitrary probability distribution functions, which pose high demands on the analysis resources. Both problems lead to an increase in analysis time and memory and, thus, limit the range of systems amenable to analysis under constraints on analysis resources.

The analysis complexity can be reduced by means of three approaches:

1. Restricting the assumptions on the applications to analyse (which means restricting the class of applications that can be handled),

2. Solving the large stochastic process with less resources,

3. Finding intelligent ways to transform the stochastic process to an equivalent one with respect to the performance indicators of interest. The transformation is done such that the equivalent process is easier to solve.

Obviously, the first alternative is not desirable, on the contrary, the challenge consists in finding an acceptable balance between required analysis resources and the capacity to analyse applications under as unrestricted assumptions as possible. In this thesis, this challenge is taken.

1.4 Contribution

The contribution of this thesis includes:

- An analytic, exact method for obtaining the expected deadline miss ratio which can be efficiently applied to mono-processor systems [Chapter 4], [MEP01];
- An analytic, approximate method for obtaining the expected deadline miss ratio which can be efficiently applied to multi-processor systems [Chapter 5], [MEP02];
- A designer controlled way to trade analysis accuracy with required analysis resources (time and memory) [Chapter 5], [MEP02];
- A fairly general class of applications the methods are applicable to [Chapter 3]:
  - Periodic tasks possibly with precedence constraints among them,
  - Flexible late task policy,
  - Almost any non-preemptive scheduling policy is supported,
  - Task execution times with generalised probability distribution functions.
- A further relaxation of the assumptions regarding the application [Chapter 6],
- Experiments qualitatively and quantitatively describing the dependency between application characteristics (number of tasks, number of processors, dependency degree) and required analysis resources, allowing to assess the applicability of the proposed methods [Chapters 4 and 5].

This thesis not only provides tools for the analysis of real-time applications with stochastic execution times, but also extends, as much as possible, the class of applications to which the tools can be applied.

1.5 Thesis organisation

The next chapter surveys the related work in the area of performance estimation of real-time systems considering also the particular context of varying task execution time. As mentioned, an analytic approach has to rely on a mathematical formalisation of the problem. The notation used throughout the thesis and the problem formulation are introduced in Chapter 3. The first approach to solve the problem, efficiently applicable to mono-processor systems, is detailed in Chapter 4. The second approach, applicable to multiprocessor systems, is presented in Chapter 5. Chapter 6 discusses relaxations on the initial assumptions as well as extensions of the results and their impact on the analysis complexity. Finally, Chapter 7 draws the conclusions and presents directions of future work.

Chapter 2

Background and Related Work

This chapter is structured in three sections. The first provides a background in schedulability analysis. The second section surveys some of the related work in the area of schedulability analysis of real-time systems with stochastic task execution times. The third section informally presents some of the concepts in probability theory and in the theory of stochastic processes to be used in the following chapters.

The earliest results in real-time scheduling and schedulability analysis have been obtained under restrictive assumptions about the task set and the underlying architecture. Thus, in the early literature, task sets with the following properties have been considered, referred in this thesis as the *restricted assumptions*: The task set is composed of a fixed number of *independent tasks* mapped on a *single processor*, the tasks are *periodically released*, each with a *fixed period*, *the deadlines equal the periods*, and *the task execution times are fixed*.

Later work was done under more relaxed assumptions. This survey of related work is limited to research considering assumption relaxations along some of the following dimensions, of interest in the context of this thesis:

- Multi-processor systems
- Data dependency relationships among the tasks
- Stochastic task execution times
- Deadlines less than or equal to the periods

- Late task policy

Note also that applications with sporadic or aperiodic tasks, or tasks with resource constraints, or applications with dynamic mapping of tasks to processors have *not* been considered in this overview.

2.1 Schedulability analysis of hard real-time systems

2.1.1 Mono-processor systems

In their seminal paper from 1973, Liu and Layland [LL73] considered sets of tasks under the restricted assumptions outlined above. The tasks are dynamically scheduled by a runtime scheduler according to an offline (static) assignment of priorities to tasks. The priority assignment is made according to the rate monotonic algorithm (the shorter the task period, the higher the task priority). Task preemption is allowed. Under these assumptions, Liu and Layland give the following processor utilisation based schedulability criterion. The task set is schedulable if

$$\sum_{i=1}^{N} \frac{C_i}{T_i} \leq N \cdot (2^{\frac{1}{N}} - 1)$$

where C_i is the worst case execution time of task τ_i, T_i is the period of task τ_i and N is the number of tasks. The left-hand side of the inequality represents U, the processor utilisation and the right-hand side represents the utilisation bound. Liu and Layland also prove that the rate monotonic (RM) priority assignment scheme is the optimal fixed (offline) priority scheme.

In the same paper, Liu and Layland analysed the set of tasks in the case they are dynamically scheduled by a runtime scheduler according to an online (dynamic) assignment of priorities to tasks. The assignment is made according to the earlier deadline first (EDF) algorithm (the closer the task deadline, the higher the task priority). Task preemption is allowed. Under these assumptions, Liu and Layland give a processor utilisation based necessary and sufficient condition for the tasks to be schedulable. Thus, the task set is schedulable if and only if

$$\sum_{i=1}^{N} \frac{C_i}{T_i} \leq 1.$$

They also prove that EDF is the optimal dynamic priority assignment algorithm. Optimality, in the context of RM and EDF scheduling, means that if there exists an offline [online] task priority assignment method under which the tasks are schedulable, then they are also schedulable under the RM [EDF] assignment method.

Recently, E. Bini *et al.* [BBB01] improved the bound given by Liu and Layland for the rate monotonic algorithm. Thus, a task set is schedulable under the same assumptions if

$$\prod_{i=1}^{N} \left(\frac{C_i}{T_i} + 1 \right) \leq 2.$$

Checking of the schedulability conditions presented above has complexity $O(N)$.

A necessary and sufficient condition for schedulability under the restricted assumptions and a fixed priority preemptive scheduling of tasks has been given by Lehoczky *et al.* [LSD89]. A task τ_i is schedulable if and only if there exists a time moment $t \in [0, T_i]$ such that

$$t \geq \sum_{j \in HP_i}^{N} C_j \cdot \left\lceil \frac{t}{T_i} \right\rceil ,$$

where $HP_i = \{j : prior(\tau_j) \geq prior(\tau_i)\}$. They have also shown that it is sufficient to check the schedulability condition only at the release times of higher priority tasks. Still, the algorithm is pseudo-polynomial.

If the task deadlines are less than the corresponding periods, a deadline monotonic task priority assignment algorithm was proposed by Leung and Whitehead [LW82]. Such a scheme assigns a higher priority to the task with the shorter relative deadline. Leung and Whitehead also proved that this algorithm is the optimal fixed priority assignment algorithm under the considered assumptions and gave a schedulability criterion. Audsley *et al.* [ABD$^+$91] extended the work of Joseph and Pandya [JP86] about response time analysis and derived a necessary and sufficient condition for schedulability under the deadline monotonic approach. Manabe and Aoyagi [MA95] gave similar schedulability conditions reducing the number of time moments that have to be evaluated, but their schedulability analysis algorithm is pseudo-polynomial as well.

In the case of deadlines less than periods and dynamic priority assignment according to an EDF policy, a necessary and sufficient schedulability condition has been given by Baruah *et al.* [BRH90]. Their test is pseudo-polynomial.

Less work has been carried out when considering task sets with precedence relationships among tasks. Audsley *et al.* [ABRW93] provide schedulability criteria under the assumption that the tasks are dynamically scheduled according to an offline (static) assignment of task priorities. The precedence relationships are implicitly enforced by carefully choosing the task periods, offsets and deadlines but this makes the analysis pessimistic. González Harbour *et al.* [GKL91, GKL94] consider certain types of precedence relationships among tasks and provide schedulability criteria for these situations. Sun *et al.* [SGL97] additionally extend the approach by considering fixed release times for the tasks.

Blazewicz [Bla76] showed how to modify the task deadlines under the assumptions of precedence relationships among tasks that are dynamically scheduled according to an online assignment of task priorities based on EDF. He has also shown that the resulting EDF* algorithm is optimal. Sufficient conditions for schedulability under the mentioned assumptions are given by Chetto *et al.* [CSB90] while Spuri and Stankovic [SS94] consider also shared resources.

2.1.2 Multi-processor systems

Most problems related to hard real-time scheduling on multi-processor systems under non-trivial assumptions have been proven to be NP-complete [GJ75, Ull75, GJ79, SSDB94].

The work of Sun and Liu [SL95] addresses multi-processor real-time systems where the application exhibits a particular form of precedence relationships among tasks, namely the periodic job-shop model. Sun and Liu have provided analytic bounds for the task response time under such assumptions. The results, plus heuristics on how to assign fixed priorities to tasks under the periodic job-shop model, are summarised in Sun's thesis [Sun97].

Audsley [Aud91] and Audsley *et al.* [ABR$^+$93] provide a schedulability analysis method based on the task response time analysis where the tasks are allowed to be released at arbitrary time moments. Including these jitters in the schedulability analysis provides for the analysis of applications with precedence relationships among tasks. Tindell [Tin94] and Tindell and Clark [TC94] extend this schedulability analysis by applying it to multi-processor systems coupled by time triggered communication links. Later on, Palencia Gutiérrez and González Harbour [PG98] improved on Tindell's work by allowing dynamic task offsets, obtaining tighter bounds for task response times.

The above mentioned work proved to be a fertile ground for the development of scheduling algorithms and schedulability analysis for complex real-world applications. Thus, the work of Eles *et al.* [EDPP00] and P. Pop *et al.* [PEP99, PEP00] provides heuristic scheduling algorithms for applications represented as conditional process graphs implemented on a distributed system with a time triggered communication protocol. T. Pop *et al.* [PEP02] further considered heterogeneous communication subsystems with both time triggered and event triggered protocols. Work in the area of scheduling and schedulability analysis diversified by considering particular communication protocols, like the Token Ring protocol [SM89, Ple92], the FDDI network architecture [ACZD94], the ATM protocol [EHS97, HST97], CAN bus [THW94, DF01], or TTP bus [KFG$^+$92].

For the case when the tasks are dynamically scheduled on the processors in a multi-processor system, according to an EDF task priority assignment, the reader is referred to the work of Spuri [Spu96a, Spu96b].

2.2 Systems with stochastic task execution times

The main problem that was addressed by the work surveyed in this section concerns the schedulability analysis when considering stochastic task execution times and various kinds of task models.

2.2.1 Mono-processor systems

Atlas and Bestavros [AB98] extend the classical rate monotonic scheduling policy with an admittance controller in order to handle tasks with stochastic execution times. They analyse the quality of service of the resulting schedule and its dependence on the admittance controller parameters. The approach is limited to rate monotonic analysis and assumes the presence of an admission controller at run-time.

Abeni and Butazzo's work [AB99] addresses both scheduling and performance analysis of tasks with stochastic parameters. Their focus is on how to schedule both hard and soft real-time tasks on the same processor, in such a way that the hard ones are not disturbed by ill-behaved soft tasks. The performance analysis method is used to assess their proposed scheduling policy (constant bandwidth server), and is restricted to the scope of their assumptions.

Tia *et al.* [TDS$^+$95] assume a task model composed of independent tasks. Two methods for performance analysis are given. One of them is just an estimate and is demonstrated to be overly optimistic. In the second method, a soft task is transformed into a deterministic task and a sporadic one. The latter is executed only when the former exceeds the promised execution time. The sporadic tasks are handled by a server policy. The analysis is carried out on this model.

Zhou *et al.* [ZHS99] and Hu *et al.* [HZS01] root their work in Tia's. However, they do not intend to give per-task guarantees, but characterise the fitness of the entire task set. Because they consider all possible combinations of execution times of all requests up to a time moment, the analysis can be applied only to small task sets due to complexity reasons.

De Veciana *et al.* [dJG00] address a different type of problem. Having a task graph and an imposed deadline, they determine the path that has the highest probability to violate the deadline. The problem is then reduced to a non-linear optimisation problem by using an approximation of the convolution of the probability densities.

Lehoczky [Leh96] models the task set as a Markovian process. The advantage of such an approach is that it is applicable to arbitrary scheduling policies. The process state space is the vector of lead-times (time left until the deadline). As this space is potentially infinite, Lehoczky analyses it in heavy traffic conditions, when the system provides a simple solution. The main limitations of this approach are the non-realistic assumptions about task inter-arrival and execution times.

Kalavade and Moghé [KM98] consider task graphs where the task execution times are arbitrarily distributed over discrete sets. Their analysis is based on Markovian stochastic processes too. Each state in the process is characterised by the executed time and lead-time. The analysis is performed by solving a system of linear equations. Because the execution time is allowed to take only a finite (most likely small) number of values, such a set of equations is small.

2.2.2 Multi-processor systems

Burman has pioneered the "heavy traffic" school of thought in the area of queueing [Bur79]. Lehoczky later applied and extended it in the area of real-time systems [Leh96, Leh97]. The theory was further extended by Harrison and Nguyen [HN93], Williams [Wil98] and others [PKH01, DLS01, DW93]. The application is modelled as a multi-class queueing network. This network behaves as a reflected Brownian motion with drift under heavy traffic conditions, i.e. when the processor utilisations approach 1, and therefore it has a simple solution.

Other researchers, such as Kleinberg *et al.* [KRT00] and Goel and Indyk [GI99], 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 [KS96] 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).

Our work is mostly related to the ones of Zhou *et al.* [ZHS99] and Hu *et al.* [HZS01], and Kalavade and Moghé [KM98]. Mostly, it differs by considering less restricted application classes. As opposed to Kalavade and Moghé's work, we consider *continuous* ETPDFs. Also, we accept a much larger class of scheduling policies than the fixed priority ones considered by Zhou and Hu. Moreover, our original way of concurrently constructing and analysing the underlying process, while keeping only the needed stochastic process states in memory, allows us to consider larger applications.

As far as we are aware, the heavy traffic theory fails yet to smoothly apply to real-time systems. Not only that there are cases when such a reflected Brownian motion with drift limit does not exist, as shown by Dai [DW93], but also the heavy traffic phenomenon is observed only for processor loads close to 1, leading to very long (infinite) queues of ready tasks and implicitly to systems with very large latency. This aspect makes the heavy traffic phenomenon undesirable in real-time systems.

In the context of multi-processor systems, our work significantly extends the one by Kim and Shin [KS96]. Thus, we consider arbitrary ETPDFs (Kim and Shin consider exponential ones) and we address a much larger class of scheduling policies (as opposed to FCFS considered by them).

2.3 Elements of probability theory and stochastic processes

This section *informally* introduces some of the probability theory concepts needed in the following chapters. For a more formal treatment of the subject, the reader is referred to Appendix B and to J. L. Doob's "Measure Theory" [Doo94].

Consider a set of *events*. An event may be the arrival or the completion of a task, as well as a whole sequence of actions, as for example "task τ starts at moment t_1, and it is discarded at time moment t_2". A *random variable* is a mapping that associates a real number to an event. The *distribution* of a random variable X is a real function F, $F(x) = \mathsf{P}(X \leq x)$. $F(x)$ indicates what is the probability of the events that are mapped to reals less than or equal to x. Obviously, F is monotone increasing and its limit $\lim_{x\rightarrow\infty} = 1$. Its first derivative $f(x) = \frac{dF(x)}{dx}$ is the *probability density* of the random variable. If the distribution function F is continuous, the random variable it corresponds to is said to be continuous.

If $\mathsf{P}(X \leq t + u | X > t) = \mathsf{P}(X \leq u)$ then X (or its distribution) is *memoryless*. For example, if X is a finishing time of a task τ and X is memoryless, then the probability that τ completes its execution before a time moment t_2, knowing that by time t_1 it has not yet finished, is equal to the probability of τ finishing in $t_2 - t_1$ time units. Hence, if the task execution time is distributed according to a memoryless distribution, the probability of τ finishing before t time units is independent of how much it has run already.

If the distribution of a random variable X is F of the form $F(t) = 1 - e^{-\lambda t}$, then X is *exponentially* distributed. The exponential distribution is the only continuous memoryless distribution.

A family of random variables $\{X_t : t \in I\}$[1] is a *stochastic process*. The set S of values of X_t form the *state space* of the stochastic process. If I is a discrete set, then the stochastic process is a *discrete time* stochastic process. Otherwise it is a *continuous time* stochastic process. If S is a discrete set, the stochastic process is a discrete state process or a *chain*. The interval between two consecutive state transitions in a stochastic process denotes the state holding interval.

Consider a continuous time stochastic process $\{X_t : t \geq 0\}$ with arbitrarily distributed state holding time probabilities and consider I to be the ordered set $(t_1, t_2, \ldots, t_k, \ldots)$ of time moments when a state change occurs in the stochastic process. The discrete time stochastic process $\{X_n : t_n \in I\}$ is the *embedded* discrete time process of $\{X_t : t \geq 0\}$.

We are interested in the steady state probability of a given state, i.e. the probability of the stochastic process being in a given state in the long run. This indicates also the percentage of time the stochastic process spends in that particular state. The usefulness of this value

[1]Denoted also as $\{X_t\}_{t\in I}$

is highlighted in the following example. Consider a stochastic process $\{X_t\}$ where $X_t = i$ indicates that the task τ_i is currently running. State 0 is the idle state, when no task is running. Let us assume that we are interested in the expected rate of a transition from state 0 to state i, that is, how many times per second the task τ_i is activated after a period of idleness. If we know the average rate of the transition from idleness (state 0) to τ_i running (state i) given that the processor is idle, the desired value is computed by multiplying this rate by the percentage of time the processor is indeed idle. This percentage is given by the steady state probability of the state 0.

Consider a stochastic process $\{X_t : t \geq 0\}$. If the probability that the system is in a given state j at some time $t + u$ in the future, given that it is in state j at the present time t and that its past states X_s, $s < t$, are known, is independent of these past states X_s ($\mathsf{P}(X_{t+u} = j|X_t = i, X_s, 0 \leq s < t, u > 0) = \mathsf{P}(X_{t+u} = j|X_t = i, u > 0)$), then the stochastic process exhibits the *Markov property* and it is a *Markov process*. Continuous time Markov chains are abbreviated CTMC and discrete time ones are abbreviated DTMC. If the embedded discrete time process of a continuous time process $\{X_t\}_{t\geq 0}$ is a discrete time Markov process, then $\{X_t\}_{t\geq 0}$ is a *semi-Markov* process.

To exemplify, consider a stochastic process where X_t denotes the task running on a processor at time t. The stochastic process is Markovian if the probability of task τ_j running in the future does not depend on which tasks have run in the past knowing that τ_i is running now.

It can be shown that a continuous time Markov process must have exponentially distributed state holding interval length probabilities. If we construct a stochastic process where X_t denotes the running task at time moment t, then a state transition occurs when a new task is scheduled on the processor. In this case, the state holding times correspond to the task execution times. Therefore, such a process cannot be Markovian if the task execution time probabilities are not exponentially distributed.

The steady state probability vector can relatively simply be computed by solving a linear system of equations in the case of Markov processes (both discrete and continuous ones). This property, as well as the fact that Markov processes are easier to conceptualise, makes them a powerful instrument in the analysis of systems with stochastic behaviour.

Chapter 3

Problem Formulation

In this chapter, we introduce the notations used throughout the thesis and give an exact formulation of the problem. Some relaxations of the assumptions introduced in this chapter and extensions of the problem formulation will be further discussed in Chapter 6.

3.1 Notation

3.1.1 System architecture

Let $PE = \{PE_1, PE_2, \ldots, PE_p\}$ be a set of p *processing elements*. These can be programmable processors of any kind (general purpose, controllers, DSPs, ASIPs, etc.). Let $B = \{B_1, B_2, \ldots, B_l\}$ be a set of l *buses* connecting various processing elements of PE.

Unless explicitly stated, the two types of hardware resources, processing elements and buses, will not be treated differently in the scope of this thesis, and therefore they will be denoted with the general term of *processors*. Let $M = p + l$ and $P = PE \cup B = \{P_1, P_2, \ldots, P_M\}$ be the set of processors.

3.1.2 Functionality

Let $PT = \{t_1, t_2, \ldots, t_n\}$ be a set of n *processing tasks*. Let $CT = \{\chi_1, \chi_2, \ldots, \chi_m\}$ be a set of m *communication tasks*.

Unless explicitly stated, the processing and the communication tasks will not be differently treated in the scope of this thesis, and therefore they will be denoted with the general term of *tasks*. Let $N = n + m$ and $T = PT \cup CT = \{\tau_1, \tau_2, \ldots, \tau_N\}$ denote the set of tasks.

Let $G = \{G_1, G_2, \ldots, G_h\}$ denote h *task graphs.* A task graph $G_i = (V_i, E_i \subset V_i \times V_i)$ is a directed acyclic graph (DAG) whose set of vertices V_i is a non-empty subset of the set of tasks T. The sets V_i, $1 \leq i \leq h$, form a partition of T. There exists a directed edge $(\tau_i, \tau_j) \in E_i$ if and only if the task τ_j is data dependent on the task τ_i. This data dependency imposes that the task τ_j is executed only after the task τ_i has completed execution.

Let $G_i = (V_i, E_i)$ and $\tau_k \in V_i$. Then let ${}^\circ\tau_k = \{\tau_j : (\tau_j, \tau_k) \in E_i\}$ denote the set of predecessor tasks of the task τ_i. Similarly, let $\tau_k^\circ = \{\tau_j : (\tau_k, \tau_j) \in E_i\}$ denote the set of successor tasks of the task τ_k. If ${}^\circ\tau_k = \varnothing$ then task τ_k is a *root.* If $\tau_k^\circ = \varnothing$ then task τ_k is a *leaf.*

Obviously, some consistency rules have to apply. Thus, a communication task has to have exactly one predecessor task and exactly one successor task, and these tasks have to be processing tasks.

Let $\Pi = \{\pi_1, \pi_2, \ldots, \pi_h\}$ denote the set of *task graph periods*, or task graph inter-arrival times. Each π_i time units, a new instantiation of task graph G_i demands execution. In the special case of mono-processor systems, the concept of period will be applied to individual tasks, with certain restrictions (see Section 6.1).

The real-time requirements are expressed in terms of relative deadlines. Let $\Delta = \{\delta_1, \delta_2, \ldots, \delta_h\}$ denote the set of *task graph deadlines.* δ_i is the deadline for task graph $G_i = (V_i, E_i)$. If there is a task $\tau \in V_i$ that has not completed its execution at the moment of the deadline δ_i, then the entire graph G_i missed its deadline.

The deadlines are supposed to coincide with the arrival of the next graph instantiation ($\delta_i = \pi_i$). This restriction will later be relaxed in the case of mono-processor systems [Section 6.2].

If $D_i(t)$ denotes the number of missed deadlines of graph G_i over a time span t and $A_i(t)$ denotes the number of instantiations of graph G_i over the same time span, then $m_i = \lim_{t\to\infty} \frac{D_i(t)}{A_i(t)}$ denotes the *expected deadline miss ratio* of task graph G_i.

3.1.3 Mapping

Let $MapP : PT \to PE$ be a surjective function that maps processing tasks on the processing elements. $MapP(t_i) = P_j$ indicates that processing task t_i is executed on the processing element P_j. Let $MapC : CT \to B$ be a surjective function that maps communication tasks on buses. $MapC(\chi_i) = B_j$ indicates that the communication task χ_i is performed on the bus B_j. For notation simplicity, $Map : T \to P$ is defined, where $Map(\tau_i) = MapP(\tau_i)$ if $\tau_i \in PT$ and $Map(\tau_i) = MapC(\tau_i)$ if $\tau_i \in CT$.

3.1.4 Execution times

Let Ex_i denote an execution time of an instantiation of the task t_i. Let $ET = \{\epsilon_1, \epsilon_2, \ldots, \epsilon_N\}$ denote a set of N *execution time probability density functions (ETPDFs).* ϵ_i is the probability density of the execution time (or communication time) of task (communication) τ_i on the processor (bus) $Map(\tau_i)$. The execution times are assumed to be statistically independent.

3.1.5 Late tasks policy

If a task misses its deadline, the real-time operating system takes a decision based on a designer-supplied *late task policy.* Let $Bounds = \{b_1, b_2, \ldots, b_h\}$ be a set of h integers greater than 0. The late task policy specifies that at most b_i instantiations of the task graph G_i are allowed in the system at any time. If an instantiation of graph G_i demands execution when b_i instantiations already exist in the system, the instantiation with the earliest arrival time is discarded (eliminated) from the system. An alternative to this late task policy will be discussed in Section 6.4

3.1.6 Scheduling

In the common case of more than one task mapped on the same processor, the designer has to decide on a *scheduling policy.* Such a scheduling policy has to be able to unambiguously determine the running task at any time on that processor.

Let an *event* denote a task arrival, departure or discarding. In order to be acceptable in the context described in this thesis, a scheduling policy is assumed to preserve the sorting of tasks according to their execution priority between consecutive events (the priorities are allowed to change in time, but the sorting of tasks according to their priorities is allowed to change only at event times). All practically used priority based scheduling policies, both with static priority assignment (rate monotonic, deadline monotonic) and with dynamic assignment (EDF, LLF) fulfill this requirement. The scheduling policy is restricted to non-preemptive scheduling.

3.2 Problem formulation

This section gives the problem formulation.

3.2.1 Input

The following data is given as an input to the analysis procedure:

- The set of task graphs G,
- The set of processors P,
- The mapping Map,
- The set of task graph periods Π,
- The set of task graph deadlines Δ,
- The set of execution time probability density functions ET,
- The late task policy $Bounds$, and
- The scheduling policy.

3.2.2 Output

The result of the analysis is the set $Missed = \{m_1, m_2, \ldots, m_h\}$ of expected deadline miss ratios for each task graph.

The problem formulation is extended in Section 6.3 by including the expected deadline miss ratios for each task in the results.

3.3 Example

Figure 3.1 depicts a hardware architecture consisting of the set PE of three ($p = 3$) processing elements PE_1 (white), PE_2 (dashed), and PE_3 (solid gray) and the set B of two ($l = 2$) communication buses B_1 (thick

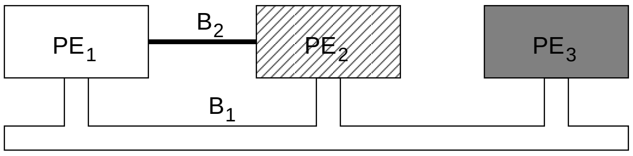


Figure 3.1: System architecture

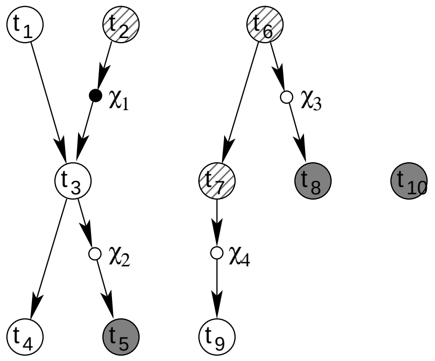


Figure 3.2: Application graphs

hollow line) and B_2 (thin line). The total number of processors $M = p + l = 5$ in this case.

Figure 3.2 depicts an application that runs on the architecture given above. The application consists of the set G of three ($h = 3$) task graphs G_1, G_2, and G_3. The set PT of processing tasks consists of ten ($n = 10$) processing tasks $t_1, t_2, \ldots, t_{10}$. The set of communication tasks CT consists of four ($m = 4$) communication tasks $\chi_1, \chi_2, \ldots, \chi_4$. The total number of tasks $N = n + m = 14$ in this case. According to the affiliation to task graphs, the set T of tasks is partitioned as follows. $T = V_1 \cup V_2 \cup V_3$, $V_1 = \{t_1, t_2, \ldots, t_5, \chi_1, \chi_2\}$, $V_2 = \{t_6, t_7, t_8, t_9, \chi_3, \chi_4\}$, $V_3 = \{t_{10}\}$. The predecessor set of t_3, for instance, is ${}^\circ t_3 = \{t_1, \chi_1\}$ and its successor set is $t_3^\circ = \{t_4, \chi_2\}$. The edge (t_1, t_3) indicates, for example, that an instantiation of the task t_3 may run only after an instantiation of the task t_1 belonging to the same task graph instantiation has successfully completed its execution. The set of task graph periods Π is $\{6, 4, 3\}$. This means that every $\pi_1 = 6$ time units a new instantiation of task graph G_1 will demand execution. Each of the three task graphs has an associated deadline, δ_1, δ_2 and δ_3, and these deadlines are assumed to be equal to the task graph periods π_1, π_2 and π_3 respectively.

The tasks t_1, t_3, t_4, and t_9 (depicted as white circles) are mapped on the processing element PE_1, the tasks t_2, t_6, and t_7 (depicted as dashed circles) are mapped on the processing element PE_2, and the tasks t_5, t_8, and t_{10} (depicted as gray circles) are mapped on the processing element PE_3. The communication between t_2 and t_3 (the communication task χ_1) is mapped on the point-to-point link B_2 whereas the other three communication tasks share the bus B_1.

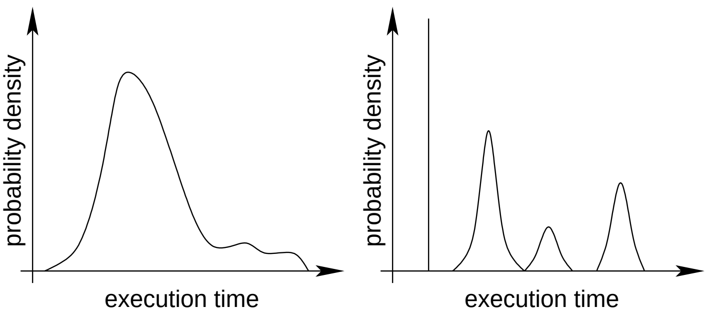


Figure 3.3: Execution time probability density functions example

Figure 3.3 depicts a couple of possible execution/communication time probability density functions (ETPDFs). There is no restriction on the supported type of probability density functions.

The late task policy is specified by the set of integers $Bounds = \{1, 1, 2\}$. It indicates that, as soon as one instantiation of G_1 or G_2 is late, that particular instantiation is discarded from the system (1 indicates that only one instantiation of the graph is allowed in the system at the same time). However, one instantiation of the graph G_3 is tolerated to be late (there may be two simultaneously active instantiations of G_3).

A possible scheduling policy could be fixed priority scheduling, for example. As the task priorities do not change, this policy obviously satisfies the restriction that the sorting of tasks according to their priorities must be unique between consecutive events.

A Gantt diagram illustrating a possible task execution over a span of 20 time units is depicted in Figure 3.4. The different task graphs are depicted in different shades in this figure. Note the two simultaneous instantiations of t_{10} in the time span 9–9.375. Note also the discarding of the task graph G_1 happening at the time moment 12 due to the lateness of task t_5. It follows that the deadline miss ratio of G_1 over the interval $[0, 18)$ is $1/3$ (one instantiations out of three missed its deadline). The deadline miss ratio of G_3 over the same interval is $1/6$, because the instantiation that arrived at time moment 6 missed its deadline. When analysing this system, the *expected* deadline miss ratio of G_1 (the ratio of the number instantiations that missed their deadline and the total number of instantiations over an infinite time interval) is 0.4. The expected deadline miss ratio of G_3 is 0.08 and the one of G_2 is 0.15.

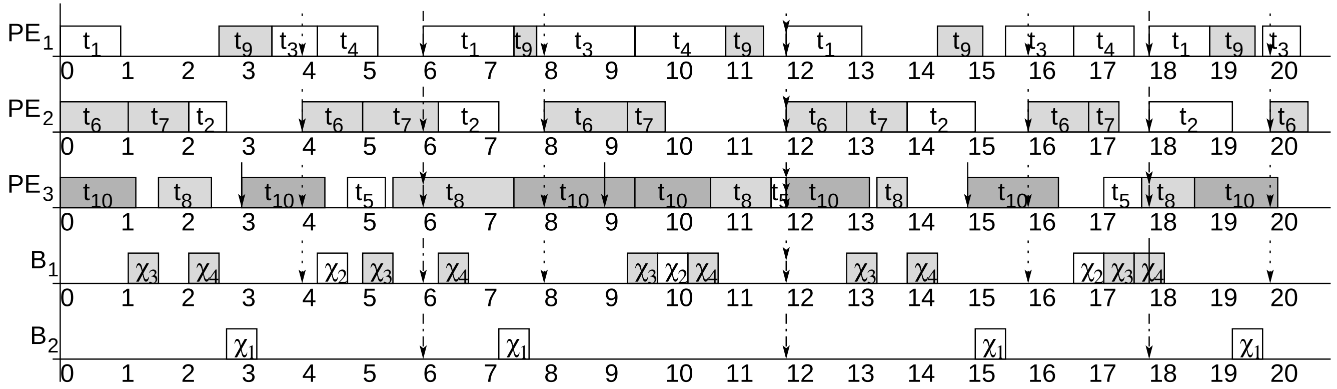


Figure 3.4: Gantt diagram

Chapter 4

An Exact Solution for Schedulability Analysis: The Mono-processor Case

This chapter presents an exact approach for determining the expected deadline miss ratios of task graphs in the case of mono-processor systems. First, it describes the stochastic process underlying the application, and shows how to construct such a stochastic process in order to obtain a semi-Markov process. Next, it introduces the concept of *priority monotonicity intervals (PMIs)* and shows how to significantly reduce the problem complexity by making use of this concept. Then, the construction of the stochastic process and its analysis are illustrated by means of an example. Finally, a more concise formulation of the algorithm is given and experimental results are presented.

4.1 The underlying stochastic process

Let us consider the problem as defined in Section 3.2 and restrict it to mono-processor systems, i.e. $P = PE = \{PE_1\}$; $B = \varnothing$; $CT = \varnothing$; $Map(\tau_i) = PE_1, \forall \tau_i \in T$; $p = 1$; $l = 0$; $m = 0$; and $M = 1$.

The goal of the analysis is to obtain the expected deadline miss ratios of the task graphs. These can be derived from the behaviour of the system. The behaviour is defined as the evolution of the system through a *state space* in time. A *state* of the system is given by the values of a set of variables that characterise the system. Such variables may be the currently running task, the set of ready tasks, the current time and the start time of the current task, etc.

Due to the considered periodic task model, the task arrival times are deterministically known. However, because of the stochastic task execution times, the completion times and implicitly the running task at an arbitrary time instant or the state of the system at that instant cannot be deterministically predicted.

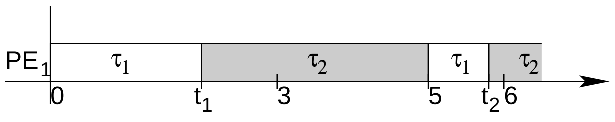
The mathematical abstraction best suited to describe and analyse such a system with random character is the stochastic process.[1] In the sequel, several alternatives for constructing the underlying stochastic process and its state space are illustrated and the most appropriate one from the analysis point of view is finally selected.

The following example is used in order to discuss the construction of the stochastic process. The system consists of one processor and the following application: $G = \{(\{\tau_1\}, \varnothing), (\{\tau_2\}, \varnothing)\}$, $\Pi = \{3, 5\}$, i.e. a set of two independent tasks with corresponding periods 3 and 5. The tasks are scheduled according to an EDF scheduling policy. For simplicity, in this example it is assumed that a late task is immediately discarded ($b_1 = b_2 = 1$). The application is observed over the time interval $[0, 6]$. At time moment 0 both tasks are ready to run and task τ_1 is activated because it has the closest deadline. Consider the following four possible execution traces:

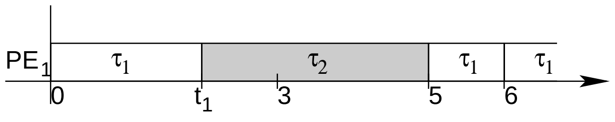
1. Task τ_1 completes its execution at time moment $t_1 < 3$. Task τ_2 is then activated and, because it attempts to run longer than its deadline 5, it is discarded at time moment 5. The instance of task τ_1 that arrived at time moment 3 is activated and it completes its execution at time moment $t_2 < 6$. At its completion time, the instantiation of task τ_2 that arrived at time moment 5 is activated. Figure 4.1(a) illustrates a Gantt diagram corresponding to this scenario.

2. The system behaves exactly like in the previous case until time moment 5. In this scenario, however, the second instantiation of task τ_1 attempts to run longer than its deadline 6 and it is discarded at 6. The new instance of task τ_1 that arrived at time moment 6 is then activated on the processor. This scenario corresponds to the Gantt diagram in Figure 4.1(b).

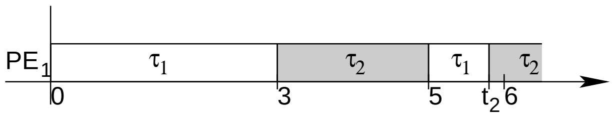
[1]The mathematical concepts used in this thesis are informally introduced in Section 2.3, and more formally in Appendix B.



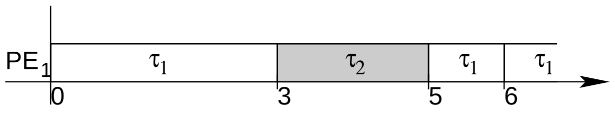
(a) Scenario 1



(b) Scenario 2



(c) Scenario 3



(d) Scenario 4

Figure 4.1: Four possible execution scenarios

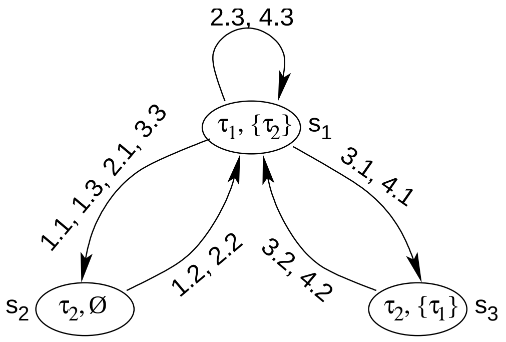


Figure 4.2: Part of the underlying stochastic process

3. Task τ_1 is activated on the processor at time moment 0. As it attempts to run longer than its deadline 3, it is discarded at this time. Task τ_2 is activated at time moment 3, but discarded at time moment 5. The instantiation of task τ_1 that arrived at time moment 3 is activated at 5 and completes its execution at time moment $t_2 < 6$. The instantiation of τ_2 that arrived at 5 is then activated on the processor. The scenario is depicted in the Gantt diagram in Figure 4.1(c).

4. The same execution trace as in the previous scenario until time moment 5. In this scenario, however, the instantiation of the task τ_1 that was activated at 5 attempts to run longer than its deadline 6 and it is discarded at 6. The new instance of task τ_1 that arrived at time moment 6 is then activated on the processor. Figure 4.1(d) illustrates the Gantt diagram corresponding to this scenario.

Let S denote the state space of the system and let a state consist of the currently running task and the multiset (bag) of ready tasks at the start time of the currently running task.[2] Formally, $S = \{(\tau, W) : \tau \in T, W \in \text{ set of all multisets of } T\}$. Because there is an upper bound on the number of concurrently active instantiations of the same task, the set S is finite. Let $\{X_t : t \geq 0\}$ be the stochastic process with state space S underlying the presented application, where X_t indicates the state of the system at time t.

Figure 4.2 depicts a part of this stochastic process corresponding to our example. The ovals in the figure denote the system states, while

[2]The ready tasks form a *multiset* rather than a set because there might be several concurrently active instantiations of the same task in the system, as explained in Chapter 3.

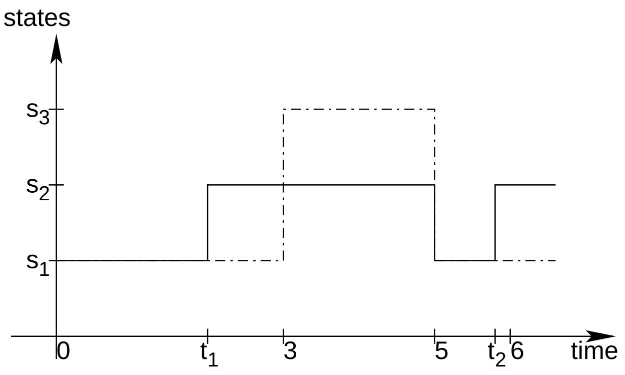


Figure 4.3: Sample paths for scenarios 1 and 4

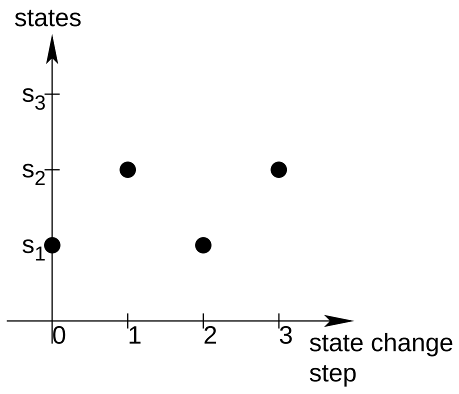
the arcs indicate the possible transitions between states. The states are marked by the identifier near the ovals. The transition labels of form $u.v$ indicate the v^{th} taken transition in the u^{th} scenario. The first transition in the first scenario is briefly explained in the following. At time moment 0, task τ_1 starts running while τ_2 is ready to run. Hence, the system state is $s_1 = (\tau_1, \{\tau_2\})$. In the first scenario, task τ_1 completes execution at time moment t_1, when the ready task τ_2 is started and there are no more ready tasks. Hence, at time moment t_1 the system takes the transition $s_1 \rightarrow s_2$ labelled with 1.1 (the first transition in the first scenario), where $s_2 = (\tau_2, \varnothing)$.

The solid line in Figure 4.3 depicts the sample path corresponding to scenario 1 while the dashed line represents the sample path of scenario 4.

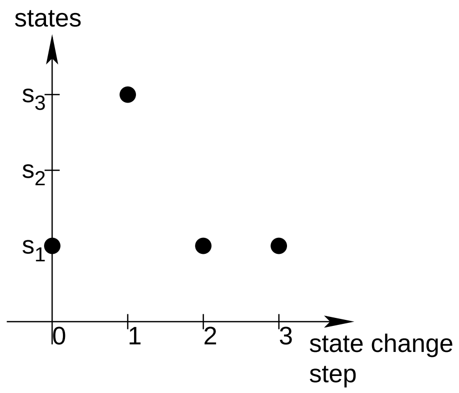
The state holding intervals (the time intervals between two consecutive state transitions) correspond to residual task execution times (how much time it is left for a task to execute). In the general case, the ET-PDFs can be arbitrary and do not exhibit the memorylessness property. Therefore, the constructed stochastic process cannot be Markovian.

In a second attempt to find a more suitable underlying stochastic process, we focus on the discrete time stochastic process embedded in the process presented above. The sample functions for the embedded discrete time stochastic process are depicted in Figure 4.4. Figure 4.4(a) corresponds to scenario 1, while Figure 4.4(b) corresponds to scenario 4. The depicted discrete sample paths are strobes of the piecewise continuous sample paths in Figure 4.3 taken at state change moments.

Task τ_1 misses its deadline when the transitions $s_1 \rightarrow s_3$ or $s_1 \rightarrow s_1$ occur. Therefore, the probabilities of these transitions have to be



(a) Discrete sample function, scenario 1



(b) Discrete sample function, scenario 4

Figure 4.4: Sample functions of the embedded chain

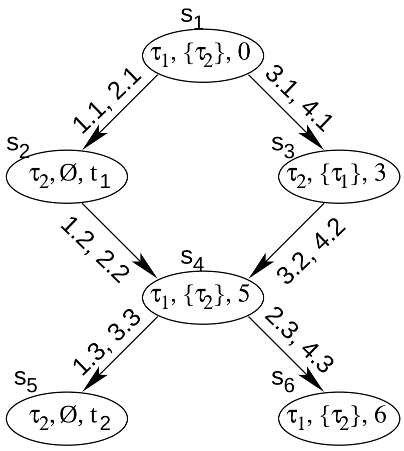
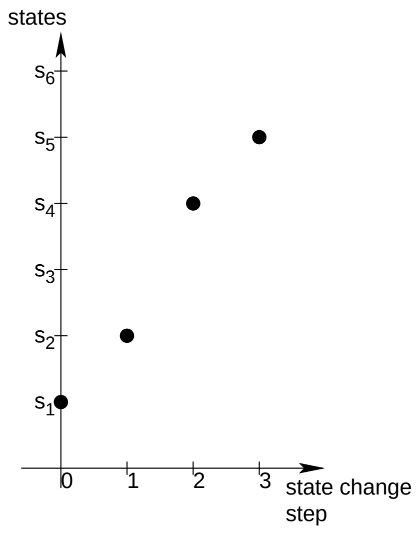


Figure 4.5: Stochastic process with new state space

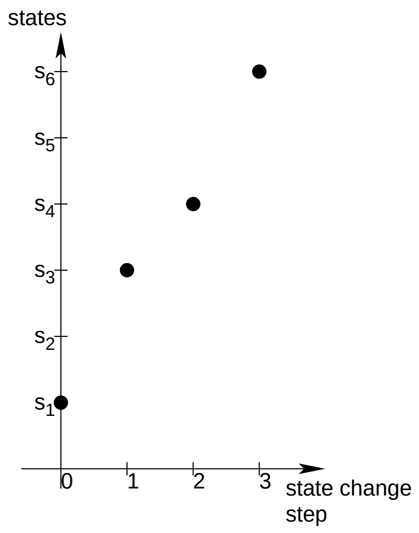
determined. It can be seen from the example that the transition $s_1 \to s_1$ cannot occur as a first step in the scenario, but only if certain previous steps have been performed (when, for example, the previous history is $s_1 \to s_3 \to s_1$). Thus, it is easy to observe that the probability of a transition from a state is dependent not only on that particular state but also on the history of previously taken transitions. Therefore, not only the continuous time process is not Markovian, but neither is the considered embedded discrete time chain.

However, by choosing another state space, it is possible to obtain a Markovian embedded process. One could choose the following state space definition: $S = \{(\tau, W, t) : \tau \in T, W \in \text{ set of all multisets of } T, t \in \mathbb{R}\}$, where τ represents the currently running task and W stands for the multiset of ready tasks at the start time of the running task. The variable t may have two interpretations, leading to different continuous time stochastic processes but having the same underlying embedded discrete time process. Thus, t may represent the current time, or it may stand for the start time of the currently running task. In the sequel, the second interpretation is used.

Figure 4.5 is a graphical representation of the stochastic process corresponding to the four scenarios, when choosing the state space as above. Figure 4.6 depicts the sample functions for scenarios 1 and 4. A state change would occur if the running task finished execution for some reason. The ready tasks can be deduced from the old ready tasks and the task instances arrived during the old tasks execution time. The new run-



(a) Sample function, scenario 1



(b) Sample function, scenario 4

Figure 4.6: Sample functions of the embedded process

ning task can be selected considering the particular scheduling policy. As all information needed for the scheduler to choose the next running task is present in a state, the set of possible next states can be determined regardless of the path to the current state. Moreover, as the task start time appears in the state information, the probability of a next state can be determined regardless of the past states.

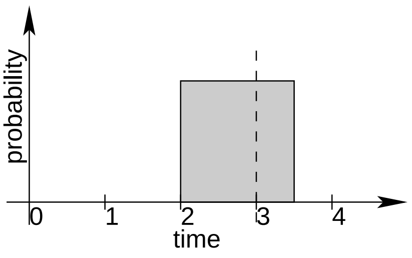
For example, let us consider the two states $s_4 = (\tau_1, \{\tau_2\}, 5)$ (τ_1 is running, τ_2 is ready, and τ_1 has started at time moment 5), and $s_6 = (\tau_1, \{\tau_2\}, 6)$ (τ_1 is running, τ_2 is ready and τ_1 has started at time moment 6) in Figure 4.5. State s_4 is left when the instantiation of τ_1 which arrived at time moment 3 and started executing at time moment 5 completes execution for some reason. Transition $s_4 \rightarrow s_6$ is taken if τ_1 is discarded because it attempts to run longer than its deadline 6. Therefore, the probability of this transition equals the probability of task τ_1 executing for a time interval longer than $6 - 5 = 1$ time unit, i.e. $\mathsf{P}(Ex(\tau_1) > 1)$. Obviously, this probability does not depend on the past states of the system.

Therefore, the embedded discrete time process is Markovian and, implicitly, the continuous time process is a semi-Markov process.

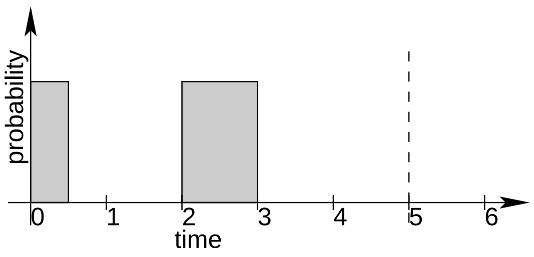
As opposed to the process depicted in Figure 4.2, the time information present in the state space of the process depicted in Figure 4.5 removes any ambiguity related to the exact instantiation which is running in a particular state. In Figure 4.2, the state s_1 corresponds to s_1, s_4 and s_6 in Figure 4.5. In Figure 4.2, transition $s_1 \rightarrow s_1$ corresponds to $s_4 \rightarrow s_6$ in Figure 4.5. However, in the case of the stochastic process in Figure 4.2 it is not clear if the currently running task in state s_1 (τ_1) is the instantiation that has arrived at time moment 0 or the one that arrived at 3. In the first case, the transition $s_1 \rightarrow s_1$ is impossible and therefore has probability 0, while in the second case, the transition *can* be taken and has non-zero probability.

4.2 Construction and analysis of the underlying stochastic process

Unfortunately, by introducing a continuous variable (the time) in the state definition, the resulting continuous time stochastic process and implicitly the embedded discrete time process become continuous (uncountable) space processes which makes their analysis very difficult. In principle, there may be as many next states as many possible execution times the running task has. Even in the case when the task execution



(a) ϵ_1



(b) ϵ_2

Figure 4.7: ETPDFs of tasks τ_1 and τ_2

time probabilities are distributed over a discrete set, the resulting underlying process becomes prohibitively large.

In our approach, we have grouped time moments into equivalence classes and, by doing so, we limited the process size explosion. Thus, practically, a set of equivalent states is represented as a single state in the stochastic process. Let us define LCM as the least common multiple of the task periods. For the simplicity of the exposition, let us first assume that the task instantiations are immediately discarded when they miss their deadlines ($b_i = 1, \forall 1 \leq i \leq h$). Therefore, the time moments $0, LCM, 2 \cdot LCM, \ldots, k \cdot LCM, \ldots$ are *renewal points* of the underlying stochastic process and the analysis can be restricted to the interval $[0, LCM)$.

Let us consider the same application as in the previous section, i.e. two independent tasks with respective periods 3 and 5 ($LCM = 15$). The tasks are scheduled according to an EDF policy. The corresponding task

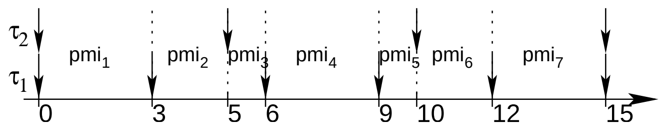


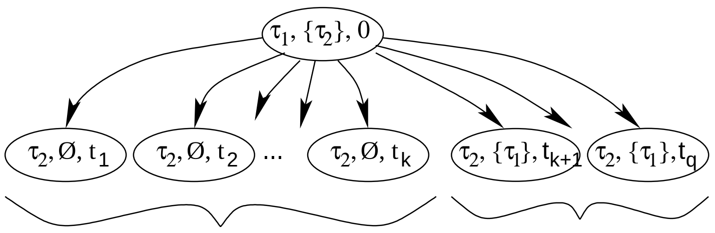
Figure 4.8: Priority monotonicity intervals

execution time probability density functions are depicted in Figure 4.7. Note that ϵ_1 contains execution times larger than the deadline.

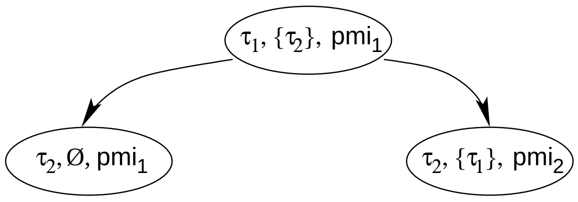
As a first step to the analysis, the interval $[0, LCM)$ is divided in disjunct intervals, the so-called *priority monotonicity intervals (PMI)*. A PMI is delimited by the time moments a task instantiation may arrive or may be discarded. Figure 4.8 depicts the PMIs for the example above.

Next, the stochastic process is constructed and analysed at the same time. Let us assume a state representation like the one introduced in the previous section. Each process state contains the identity of the currently running task, its start time and the set of ready task at the start time of the currently running one. $t_1, t_2, \ldots, t_q$ in Figure 4.9(a) are possible finishing times for the task τ_1 and, implicitly, possible starting times of the waiting instantiation of task τ_2. The number of next states equals the number of possible execution times of the running task in the current state. The resulting process is extremely large (theoretically infinite, practically depending on the discretisation resolution) and, in practice, unsolvable. Therefore, we would like to group as many states as possible in one equivalent state and still preserve the Markovian property.

Consider a state s characterised by (τ_i, W, t): τ_i is the currently running task, it has been started at time t, and W is the multiset of ready tasks. Let us consider the next two states derived from s: s_1 characterised by (τ_j, W_1, t_1) and s_2 with (τ_k, W_2, t_2). Let t_1 and t_2 belong to the same PMI. This means that no task instantiation has arrived or finished in the time interval between t_1 and t_2, no one has missed its deadline, and the relative priorities of the tasks inside the set W have not changed. Thus, $\tau_j = \tau_k =$ the highest priority task in the multiset W; $W_1 = W_2 = W \setminus \{\tau_j\}$. It follows that all states derived from state s that have their time t belonging to the same PMI have an identical currently running task and identical sets of ready tasks. Therefore, instead of considering individual times we consider time intervals, and we group together those states that have their associated start time inside the same PMI. With such a representation, the number of next states of a state s equals the number of PMIs the possible execution time of the task that runs in state s is spanning over.



(a) Individual task completion times


$\tau_1, \{\tau_2\}, pmi_1$
$\tau_2, \emptyset, pmi_1$
$\tau_2, \{\tau_1\}, pmi_2$

(b) Intervals containing task completion times

Figure 4.9: State encoding

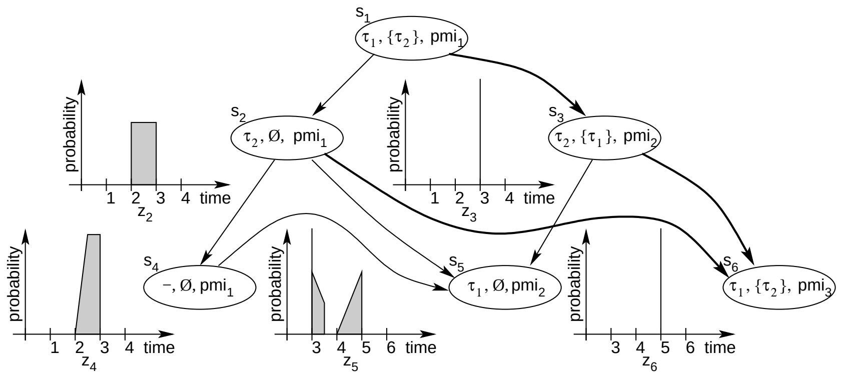


Figure 4.10: Stochastic process example

We propose a representation in which a stochastic process state is a triplet (τ, W, pmi), where τ is the running task, W the multiset of ready tasks, and pmi is the PMI containing the running task start time. In our example, the execution time of task τ_1 (which is in the interval $[2, 3.5]$, as shown in Figure 4.7(a)) is spanning over the PMIs pmi_1— $[0, 3)$—and pmi_2—$[3, 5)$. Thus, there are only two states emerging from the initial state, as shown in Figure 4.9(b).

Let $\mathcal{P}_i$, the set of predecessor states of a state s_i, denote the set of all states that have s_i as a next state. The set of successor states of a state s_i consists of those states that can be reached directly from state s_i. With our proposed stochastic process representation, the time moment a transition to a state s_i occurred is not determined exactly, as the task execution times are known only probabilistically. However, a probability density of this time can be deduced. Let z_i denote this density function. Then z_i can be computed from the functions z_j, where $s_j \in \mathcal{P}_i$, and the ETPDFs of the tasks running in the states $s_j \in \mathcal{P}_i$.

Figure 4.10 depicts a part of the stochastic process constructed for our example. The initial state is $s_1 : (\tau_1, \{\tau_2\}, pmi_1)$. The first field indicates that an instantiation of task τ_1 is running. The second field indicates that an instantiation of task τ_2 is ready to execute. The third field shows the current PMI (pmi_1—$[0, 3)$). If the instantiation of task τ_1 does not complete until time moment 3, then it will be discarded. The state s_1 has two possible next states. The first one is state $s_2 : (\tau_2, \varnothing, pmi_1)$ and corresponds to the case when the τ_1 completes before time moment 3. The second one is state $s_3 : (\tau_2, \{\tau_1\}, pmi_2)$ and corresponds to the case when τ_1 was discarded at time moment 3. State s_2 indicates that an instantiation of task τ_2 is running (it is the instance that was waiting in state s_1), that the PMI is pmi_1—$[0, 3)$—and that no task is waiting. Consider state s_2 to be the new current state. Then the next states could be state $s_4 : (-, \varnothing, pmi_1)$ (task τ_2 completed before time moment 3 and the processor is idle), state $s_5 : (\tau_1, \varnothing, pmi_2)$ (task τ_2 completed at a time moment somewhere between 3 and 5), or state $s_6 : (\tau_1, \{\tau_2\}, pmi_3)$ (the execution of task τ_2 reached over time moment 5, and hence it was discarded at time moment 5). The construction procedure continues until all possible states corresponding to the time interval $[0, LCM)$, i.e. $[0, 15)$ have been visited.

A function z_i is the probability density function of the times when the system takes a transition to state s_i. z_2, z_3, z_4, z_5, and z_6 are shown in Figure 4.10 to the left of their corresponding states $s_2, s_3, \ldots, s_6$ respectively. The transition from state s_4 to state s_5 occurs at a precisely known time instant, time 3, at which a new instantiation of task τ_1 arrives. Therefore, z_5 will contain a scaled Dirac impulse at the beginning of the corresponding PMI. The scaling coefficient equals the probability of being in state s_4 (the integral of z_4, i.e. the shaded surface below the z_4 curve). The probability density function z_5 results from the superposition of $z_2 * \epsilon_2$ (because task τ_2 runs in state s_2) with $z_3 * \epsilon_2$ (because task τ_2 runs in state s_3 too) and with the aforementioned scaled Dirac impulse over pmi_2, i.e. over the time interval $[3, 5)$. With $*$, we denote the convolution of two probability densities, i.e. $(z * \epsilon)(t) = \int_0^\infty z(t-x) \cdot \epsilon(x) dx$.

The embedded process being Markovian, the probabilities of the transitions out of a state s_i are computed exclusively from the information stored in that state s_i. For example, the probability of the transition from state s_2 to state s_5 (see Figure 4.10) is given by the probability that the transition occurs at some time moment in the PMI of state s_5 (the interval $[3, 5)$). This probability is computed by integrating $z_2 * \epsilon_2$ over the interval $[3, 5)$. The probability of a task missing its deadline is easily computed from the transition probabilities of those transitions that correspond to a discarding of a task instantiation (the thick arrows in Figure 4.10, in our case). For example, let us consider the transition $s_2 \to s_6$. The system enters state s_2 at a time whose probability density is given by z_2. The system takes the transition $s_2 \to s_6$ when the attempted completion time of τ_2 (running in s_2) exceeds 5. The completion time is the sum of the starting time of τ_2 (whose probability density is given by z_2) and the execution time of τ_2 (whose probability density is given by ϵ_2). Hence, the probability density of the completion time of τ_2 is given by the convolution of the above mentioned densities. Once this density is computed, the probability of the completion time being larger than 5 is easily computed by integrating the result of the convolution over the interval $[5, \infty)$.

As it can be seen, by using the PMI approach, some process states have more than one incident arc, thus keeping the graph "narrow". This is because, as mentioned, one process state in our representation captures several possible states of a representation considering individual times (see Figure 4.9(a)).

Because the number of states grows rapidly and each state has to store its probability density function, the memory space required to store the whole process can become prohibitively large. Our solution to mastering memory complexity is to perform the stochastic process construction and analysis simultaneously. As each arrow updates the time probability density of the state it leads to, the process has to be constructed in topological order. The result of this procedure is that the process is never stored entirely in memory but rather that a *sliding window of states* is used for analysis. For the example in Figure 4.10, the construction starts with state s_1. After its next states (s_2 and s_3) are created, their corresponding transition probabilities determined and the possible discarding probabilities accounted for, state s_1 can be removed from memory. Next, one of the states s_2 and s_3 is taken as current state, let us consider state s_2. The procedure is repeated, states s_4, s_5 and s_6 are created and state s_2 removed. At this moment, the arcs emerging from states s_3 and s_4 have not yet been created. Consequently, one would think that any of the states s_3, s_4, s_5, and s_6 can be selected for continuation of the analysis. Obviously, this is not the case, as not all the information needed in order to handle states s_5 and s_6 are computed (in particular those coming from s_3 and s_4). Thus, only states s_3 and s_4 are possible alternatives for the continuation of the analysis in topological order. The next section discusses the criteria for selection of the correct state to continue with.

4.3 Memory efficient analysis method

As shown in the example in Section 4.2, only a sliding window of states is simultaneously kept in memory. All states belonging to the sliding window are stored in a priority queue. The key to the process construction in topological order lies in the order in which the states are extracted from this queue. First, observe that it is impossible for an arc to lead from a state with a PMI number u to a state with a PMI number v so that $v < u$ (there are no arcs back in time). Hence, a first criterion for selecting a state from the queue is to select the one with the smallest

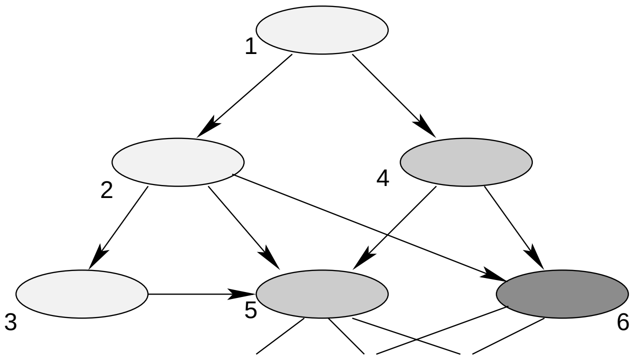


Figure 4.11: State selection order

PMI number. A second criterion determines which state has to be selected out of those with the same PMI number. Note that inside a PMI no new task instantiation can arrive, and that the task ordering according to their priorities is unchanged. Thus, it is impossible that the next state s_k of a current state s_j would be one that contains waiting tasks of higher priority than those waiting in s_j. Hence, the second criterion reads: among states with the same PMI, one should choose the one with the waiting task of highest priority. Figure 4.11 illustrates the algorithm on the example given in Section 4.2 (Figure 4.10). The shades of the states denote their PMI number. The lighter the shade, the smaller the PMI number. The numbers near the states denote the sequence in which the states are extracted from the queue and processed.

4.4 Flexible discarding

The examples considered so far dealt with applications where a late task is immediately discarded ($b_i = 1$, $1 \leq i \leq h$, i.e. at most one active instance of each task graph is allowed at any moment of time).

In this case, all the late tasks are discarded at the time moments $LCM, 2 \cdot LCM, \ldots, k \cdot LCM, \ldots$ because at these moments new instantiations of all tasks arrive. The system behaves at these time moments as if it has just been started. The time moments $k \cdot LCM$, $k \in \mathbb{N}$ are called renewal points. The system states at the renewal points are equivalent to the initial state which is unique and deterministically known. Thus, the behaviour of the system over the intervals $[k \cdot LCM, (k+1) \cdot LCM)$, $k \in \mathbb{N}$, is statistically equivalent to the behaviour over the time interval

$[0, LCM)$. Therefore, in the case when $b_i = 1$, $1 \leq i \leq h$, it is sufficient to analyse the system solely over the time interval $[0, LCM)$.

In order to illustrate the construction of the stochastic process in the case $b_i > 1$, when several instantiations of a task graph G_i may exist at the same time in the system, let us consider an application consisting of two independent tasks, τ_1 and τ_2, with periods 2 and 4 respectively. $LCM = 4$ in this case. The tasks are scheduled according to an RM policy. At most one active instantiation of τ_1 is tolerated in the system at a certain time ($b_1 = 1$) and at most two concurrently active instantiations of τ_2 are tolerated in the system ($b_2 = 2$).

Figure 4.12 depicts a part of the stochastic process underlying this example application. It was constructed using the procedure sketched in Sections 4.2 and 4.3. The state indexes show the order in which the states were analysed (extracted from the priority queue mentioned in Section 4.3).

Let us consider state $s_6 = (\tau_2, \varnothing, [2, 4))$, i.e. the instantiation of τ_2 that arrived at time moment 0 has been started sometimes between the time moments 2 and 4 and there have not been any ready tasks at the start time of τ_2. Let us assume that the finishing time of τ_2 lies past the LCM (4). At time moment 4, a new instantiation of τ_2 arrives and the running instantiation is *not* discarded, as $b_2 = 2$. On one hand, if the finishing time of the running instantiation belongs to the interval $[6, 8)$, the system performs the transition $s_6 \to s_{14}$ (Figure 4.12). If, on the other hand, the running instantiation attempts to run past the time moment 8, then at this time moment a *third* instantiation of τ_2 would concurrently require service from the system and, therefore, the running task (the oldest instantiation of τ_2) is eliminated from the system. The transition $s_6 \to s_{19}$ in the stochastic process in Figure 4.12 corresponds to this latter case. We observe that when a task execution spans beyond the time moment LCM, the resulting state is not unique. The system does not behave as if it has just been restarted at time moment LCM, and, therefore, the intervals $[k \cdot LCM, (k+1) \cdot LCM)$, $k \in \mathbb{N}$, are not statistically equivalent to the interval $[0, LCM)$. Hence, it is not sufficient to analyse the system over the interval $[0, LCM)$ but rather over several consecutive intervals of length LCM.

Let an interval of the form $[k \cdot LCM, (k+1) \cdot LCM)$ be called the *hyperperiod* k and denoted H_k. $H_{k'}$ is a *lower* hyperperiod than H_k ($H_{k'} < H_k$) if $k' < k$. Consecutively, H_k is a *higher* hyperperiod than $H_{k'}$ ($H_k > H_{k'}$) if $k > k'$.

For brevity, we say that a state s belongs to a hyperperiod k (denoted $s \in H_k$) if its PMI field is a subinterval of the hyperperiod k. In our

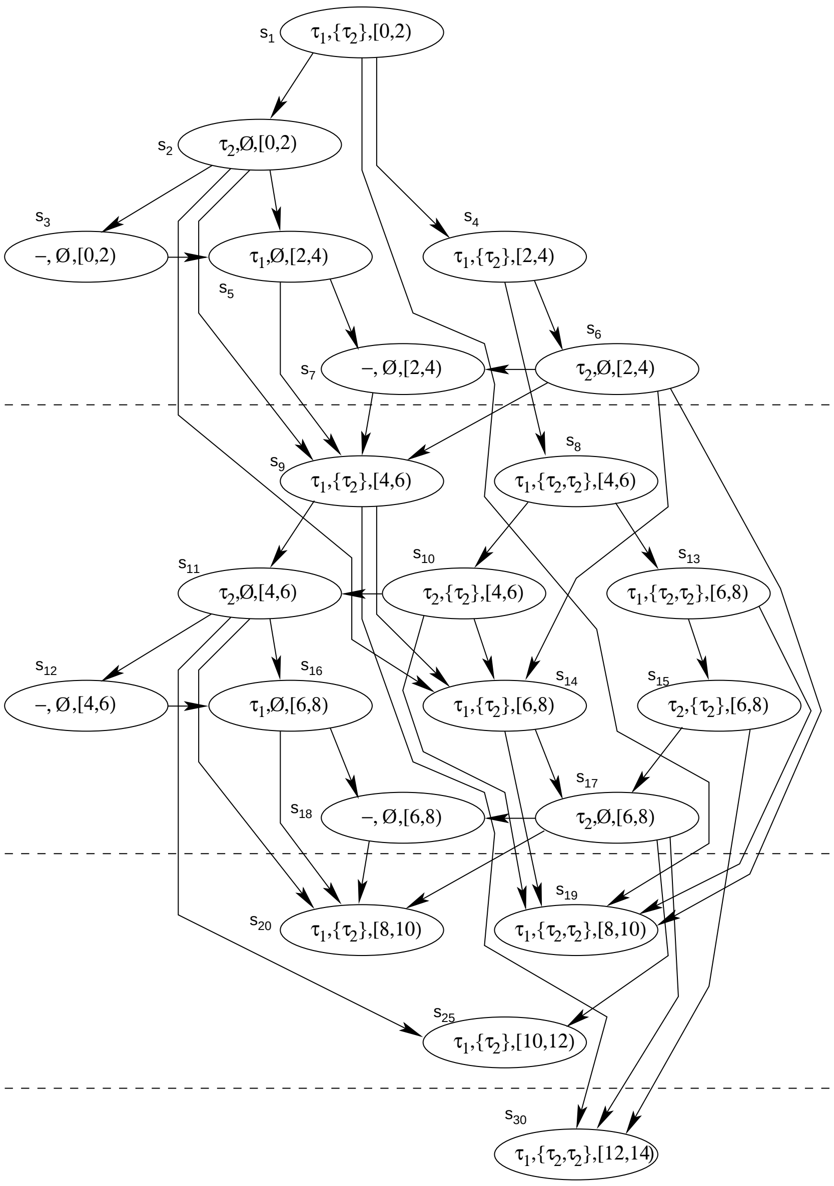


Figure 4.12: Part of the stochastic process underlying the example application

example, three hyperperiods are considered, $H_0 = [0, 4)$, $H_1 = [4, 8)$, and $H_2 = [8, 12)$. In the stochastic process in Figure 4.12, $s_1, s_2, \ldots, s_7 \in H_0$, $s_8, s_9, \ldots, s_{18} \in H_1$, and $s_{19}, s_{20}, s_{25} \in H_2$ (node that not all states have been depicted in Figure 4.12).

In general, let us consider a state s and let $\mathcal{P}_s$ be the set of its predecessor states. Let k denote the *order* of the state s defined as the lowest hyperperiod of the states in $\mathcal{P}_s$ ($k = \min\{j : s' \in H_j, s' \in \mathcal{P}_s\}$). If $s \in H_k$ and s is of order k' and $k' < k$, then s is a *back state.*In our example, s_8, s_9, s_{14}, and s_{19} are back states of order 0, while s_{20}, s_{25} and s_{30} are back states of order 1.

Obviously, there cannot be any transition from a state belonging to a hyperperiod H to a state belonging to a lower hyperperiod than H ($s \to s', s \in H_k, s' \in H_{k'} \Rightarrow H_k \leq H_{k'}$). Consequently, the set $\mathcal{S}$ of all states belonging to a hyperperiod H_k can be constructed from the back states of an order smaller than k. We say that $\mathcal{S}$ is *generated* by the aforementioned back states. For example, the set of all states $s_8, s_9, \ldots, s_{18} \in H_1$ can be derived from the back states s_8, s_9, s_{14}, and s_{19} of order 0. The intuition behind this is that back states are inheriting all the needed information across the border between hyperperiods.

Before continuing our discussion, we have to introduce the notion of *similarity* between states. We say that two states s_i and s_j are similar ($s_i \sim s_j$) if

1. The task which is running in s_i and s_j is the same,
2. The multiset of ready tasks in s_i and s_j is the same,
3. The PMIs in the two states differ only by a multiple of LCM,
4. $z_i = z_j$ (z_i is the probability density function of the times when the system takes a transition to s_i).

Let us consider the construction and analysis of the stochastic process, as described in Sections 4.2 and 4.3. Let us consider the moment x, when the last state belonging to a certain hyperperiod H_k has been eliminated from the sliding window. R_k is the set of back states stored in the sliding window at the moment x. Let the analysis proceed with the states of the hyperperiod H_{k+1} and let us consider the moment y when the last state belonging to H_{k+1} has been eliminated from the sliding window. Let R_{k+1} be the set of back states stored in the sliding window at moment y.

If the sets R_k and R_{k+1} contain pairwise similar states, then it is guaranteed that R_k and R_{k+1} generate identical stochastic processes during the rest of the analysis procedure (as stated, at a certain moment the set of back states univocally determines the rest of the stochastic process). In our example, $R_0 = \{s_8, s_9, s_{14}, s_{19}\}$ and $R_1 = \{s_{19}, s_{20}, s_{25}, s_{30}\}$. If $s_8 \sim s_{19}$, $s_9 \sim s_{20}$, $s_{14} \sim s_{25}$, and $s_{19} \sim s_{30}$ then the analysis process may stop as it reached convergence.

Consequently, the analysis proceeds by considering states of consecutive hyperperiods until the information captured by the back states in the sliding window does not change anymore. Whenever the underlying stochastic process has a steady state, this steady state is guaranteed to be found.

4.5 Construction and analysis algorithm

The analysis is performed in two phases:

1. Divide the interval $[0, LCM)$ in PMIs,
2. Construct the stochastic process in topological order and analyse it.

The concept of PMI (called in their paper "state") was introduced by Zhou *et al.*[ZHS99] in a different context, unrelated to the construction of a stochastic process. Let A_i denote the set of time moments in the interval $[0, LCM)$ when a new instantiation of the task τ_i arrives and let A denote the union of all A_i. Let D_i denote the set of absolute deadlines of the instantiations of task τ_i in the interval $[0, LCM)$, and D be the union of all D_i . Consequently,

$$\begin{aligned} A_i &= \{x : x = k \cdot \pi_i, 0 \le k < LCM/\pi_i\} \\ D_i &= \{x : x = \delta_i + k \cdot \pi_i, 0 \le k < LCM/\pi_i\} \end{aligned}$$

If the deadlines are assumed to equal the periods, then $D_i = A_i$ (the moment 0 is assumed to be the deadline of the instantiation arrived at moment $-\pi_i$.

Let $H = A \cup D$. If H is sorted in ascending order of the time moments, then a priority monotonicity interval is the interval between two consecutive time moments in H. The last PMI is the interval between the greatest element in H and LCM. The only restriction imposed on the scheduling policies accepted by our approach is that inside a PMI the ordering of tasks according to their priorities is not allowed to change. The consequence of this assumption is that the next state can be determined, no matter when the currently running task completes within the

```
divide $[0, LCM)$ in PMIs;
$pmi\_no =$ number of PMIs between 0 and $LCM$;
put first state in the priority queue $pqueue$;
$k = 1$;
$R_{old} = \varnothing$;               // $R_{old}$ is the set of densities $z$
                         // of the back states after iteration $k$
$(R_{new}, Missed) = construct\_and\_analyse()$;      // $Missed$ is the set
                         // of expected deadline miss ratios
do
    $k = k + 1$;
    $R_{old} = R_{new}$;
    $(R_{new}, Missed) = construct\_and\_analyse()$;
while $R_{new} \neq R_{old}$;


construct_and_analyse:
while $\exists s \in pqueue$ such that $s.pmi \leq pmi\_no$ do
    $s_j =$ extract state from $pqueue$;
    $\tau_i = s_j.running$;            // first field of the state
    $\xi = convolute(\epsilon_i, z_j)$;
    $nextstatelist = next\_states(s_j)$; // consider task dependencies!
    for each $s_u \in nextstatelist$ do
        compute the probability of the transition
                                        from $s_j$ to $s_u$ using $\xi$;
        update deadline miss probabilities $Missed$;
        update $z_u$;
        if $s_u \notin pqueue$ then
            put $s_u$ in the $pqueue$;
        end if;
        if $s_u$ is a back state and $s_u \notin R_{new}$ then
            $R_{new} = R_{new} \cup \{s_u\}$;
        end if;
    end for;
    delete state $s_j$;
end while;
return $(R_{new}, Missed)$;
```

Figure 4.13: Construction and analysis algorithm

PMI. All the widely used scheduling policies we are aware of (RM, EDF, FCFS, LLF, etc.) exhibit this property.

The algorithm proceeds as discussed in Sections 4.2, 4.3 and 4.4. An essential point is the construction of the process in topological order, which allows only parts of the states to be stored in memory at any moment. The algorithm for the stochastic process construction is depicted in Figure 4.13.

A global priority queue stores the states in the sliding window. The state priorities are assigned as shown in Section 4.3. The initial state of the stochastic process is put in the queue. The explanation of the algorithm is focused on the `construct_and_analyse` procedure. It extracts one state at the time from the queue. Let $s_j = (\tau_i, W_i, pmi_i)$ be such a state. The probability density of the time when a transition occurred *to* s_j is given by the function z_j. The priority scheme of the priority queue ensures that s_j is extracted from the queue only after *all* the possible transitions to s_j have been considered, and thus z_j contains accurate information. In order to obtain the probability density of the time when task τ_i completes its execution, the density of its starting time (z_j) and the ETPDF of τ_i (ϵ_i) have to be convoluted. Let ξ be the density resulting from the convolution. ξ is used to determine the PMIs the execution of τ_i may span over. For each of those PMIs, based on W_i, on the set of task instantiations that have arrived during the runtime of τ_i, and taking into consideration the precedence relationships between the tasks, the new running task as well as the new multiset of ready tasks are computed resulting in a set of states *nextstatelist*. The probability densities of the times a transition to $s_u \in nextstatelist$ is taken, z_u, are updated based on ξ. The state s_u is then added to the priority queue and s_j removed from memory. This procedure is repeated until no more states in the queue have their PMI field in the range $0, \ldots, pmi\_no$ (until there is no task instantiation that started its execution between 0 and LCM). Once such a situation is reached, partial results, corresponding to an interval of length $[0, LCM)$ are available and the `construct_and_analyse` procedure returns. The `construct_and_analyse` procedure is repeated until the set of back states R does not change anymore.

4.6 Experiments

The most computation intensive part of the analysis is the computation of the convolutions $z_i * \epsilon_j$. In our implementation we used the FFTW library [FJ98] for performing convolutions based on the Fast Fourier Transform. The number of convolutions to be performed equals the

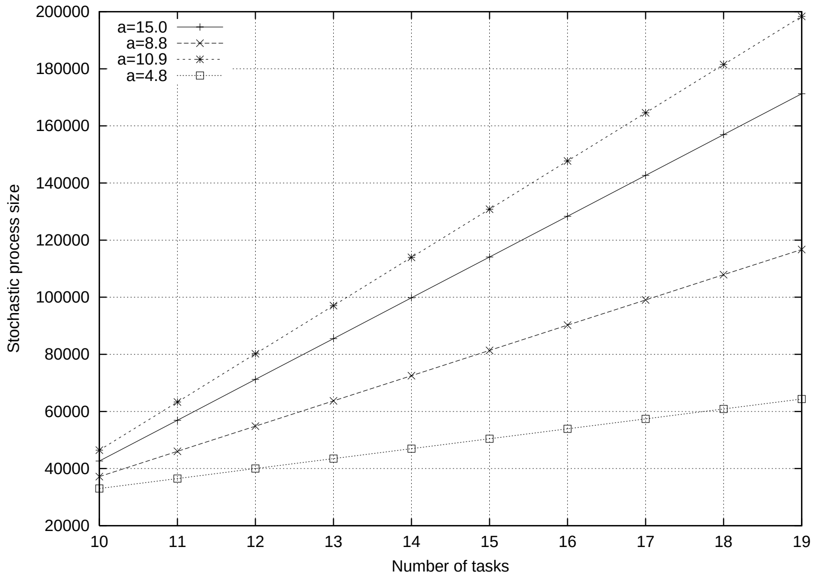


Figure 4.14: Stochastic process size vs. number of tasks

number of states of the stochastic process. The memory required for analysis is determined by the maximum number of states in the sliding window. The main factors on which the stochastic process depends are LCM (the least common multiple of the task periods), the number of PMIs, the number of tasks N, the task dependencies, and the maximum allowed number of concurrently active instantiations of the same task graph.

As the selection of the next running task is unique, given the pending tasks and the time moment, the particular scheduling policy has a reduced impact on the process size. On the other hand, the task dependencies play a significant role, as they strongly influence the set of ready tasks and by this the process size.

In the following, we report on four sets of experiments. The aspects of interest were the stochastic process size, as it determines the analysis execution time, and the maximum size of the sliding window, as it determines the memory space required for the analysis. All experiments were performed on an UltraSPARC 10 at 450 MHz.

In the first set of experiments we analysed the impact of the number of tasks on the process size. We considered task sets of 10 up to 19

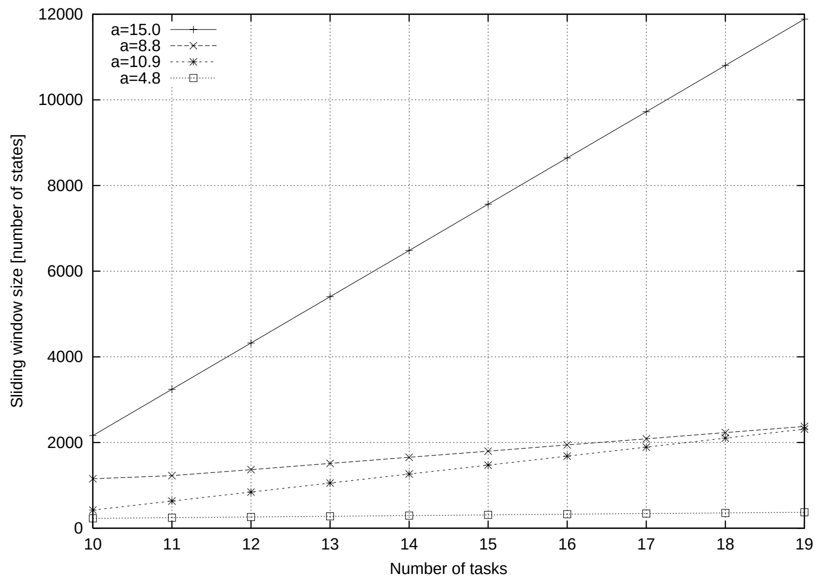


Figure 4.15: Size of the sliding window of states vs. number of tasks

independent tasks. LCM, the least common multiple of the task periods, was 360 for all task sets. We repeated the experiment four times for average values of the task periods $a = 15.0$, 10.9, 8.8, and 4.8 (keeping $LCM = 360$). The results are shown in Figure 4.14. Figure 4.15 depicts the maximum size of the sliding window for the same task sets. As it can be seen from the diagram, the increase, both of the process size and of the sliding window, is linear. The steepness of the curves depends on the task periods (which influence the number of PMIs). It is important to notice the big difference between the process size and the maximum number of states in the sliding window. In the case for 19 tasks, for example, the process size is between 64356 and 198356 while the dimension of the sliding window varies between 373 and 11883 (16 to 172 times smaller). The reduction factor of the sliding window compared to the process size was between 15 and 1914, considering all our experiments.

In the second set of experiments we analysed the impact of the application period LCM (the least common multiple of the task periods) on the process size. We considered 784 sets, each of 20 independent tasks. The task periods were chosen such that LCM takes values in the

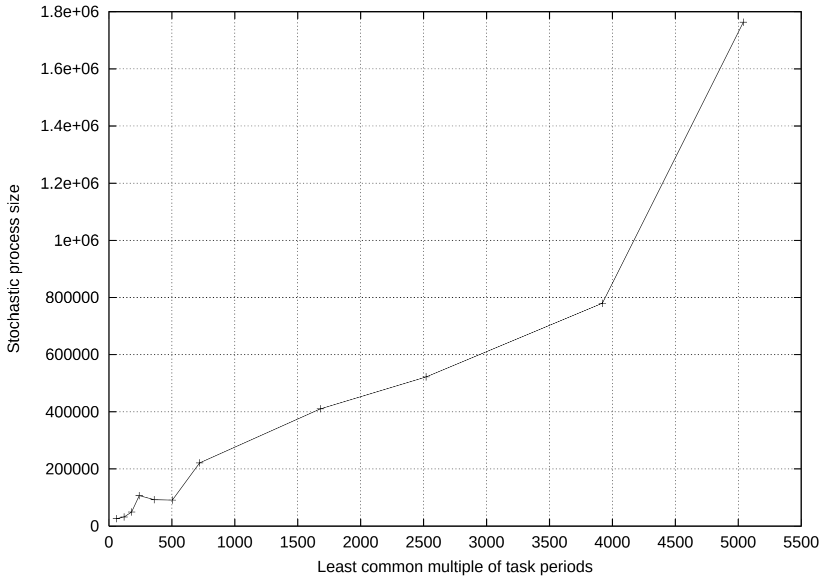


Figure 4.16: Stochastic process size vs. application period LCM

interval $[1, 5040]$. Figure 4.16 shows the variation of the average process size with the application period.

With the third set of experiments we analysed the impact of task dependencies on the process size. A task set of 200 tasks with strong dependencies (28000 arcs) among the tasks was initially created. The application period LCM was 360. Then 9 new task graphs were successively derived from the first one by uniformly removing dependencies between the tasks until we finally got a set of 200 independent tasks. The results are depicted in Figure 4.17 with a logarithmic scale for the y axis. The x axis represents the degree of dependencies among the tasks (0 for independent tasks, 9 for the initial task set with the highest amount of dependencies).

In the fourth set of experiments, the impact of the average number of concurrently active instantiations of the same task graph on the stochastic process size was analysed. 18 sets of task graphs containing between 12 and 27 tasks grouped in 2 to 9 task graphs were randomly generated. Each task set was analysed between 9 and 16 times considering different upper bounds for the maximum allowed number of concurrently active task graph instantiations. These upper bounds ranged from 1 to 3. The

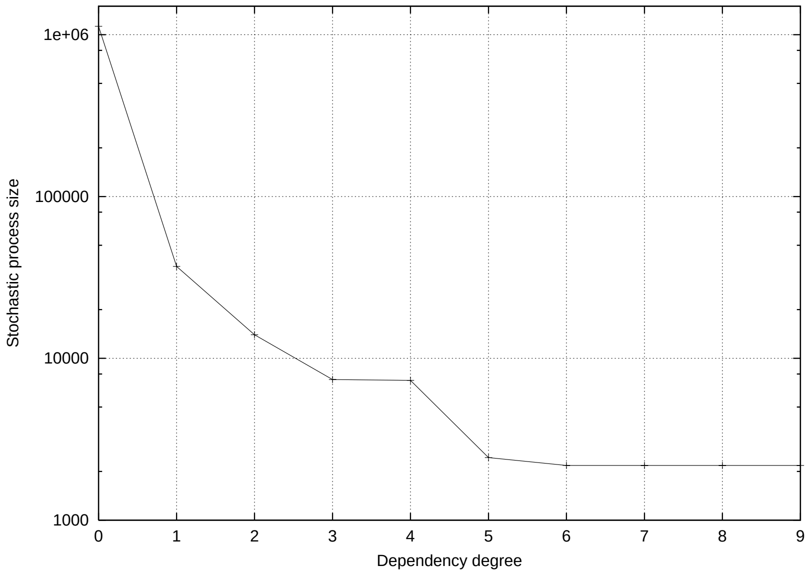


Figure 4.17: Stochastic process size vs. task dependency degree

results were averaged for the same number of tasks. The dependency of the underlying stochastic process size as a function of the average of the maximum allowed number of instantiations of the same task graph that are concurrently active is plotted in Figure 4.18. Note that the y-axis is logarithmic. Different curves correspond to different sizes of the considered task sets. It can be observed that the stochastic process size is approximately linear in the average of the maximum allowed number of concurrently active instantiations of the same task graph.

As mentioned, the execution time for the analysis algorithm strictly depends on the process size. Therefore, we showed all the results in terms of this parameter. For the set of 200 independent tasks used in the third experiment (process size 1126517, Figure 4.17) the analysis time was 745 seconds. In the case of the same 200 tasks with strong dependencies (process size 2178) the analysis took 1.4 seconds.

Finally, we considered an example from the mobile communication area. Figure 4.19 depicts a set of 8 tasks that co-operate in order to decode the digital bursts corresponding to a GSM 900 signalling channel. The incoming bursts are demodulated by the demodulation task, based on the frequency indexes generated by the frequency hopping task. The

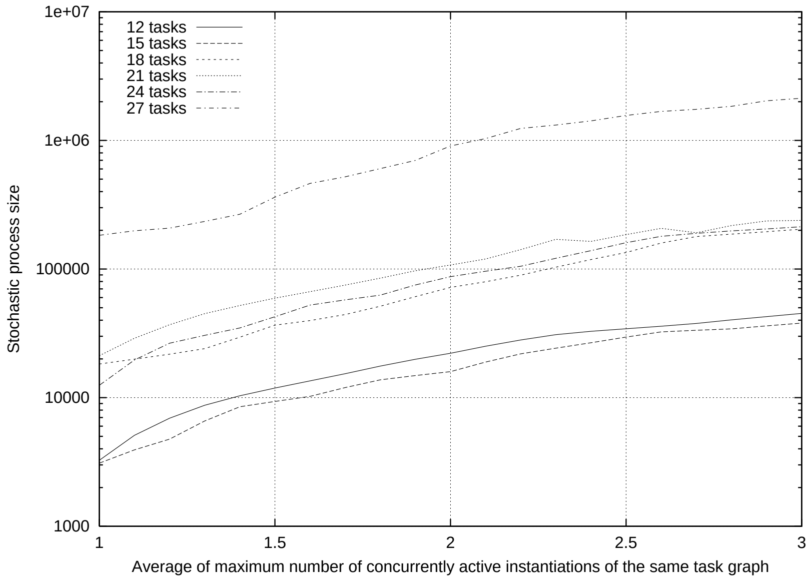


Figure 4.18: Stochastic process size vs. average number of concurrently active instantiations of the same task graph

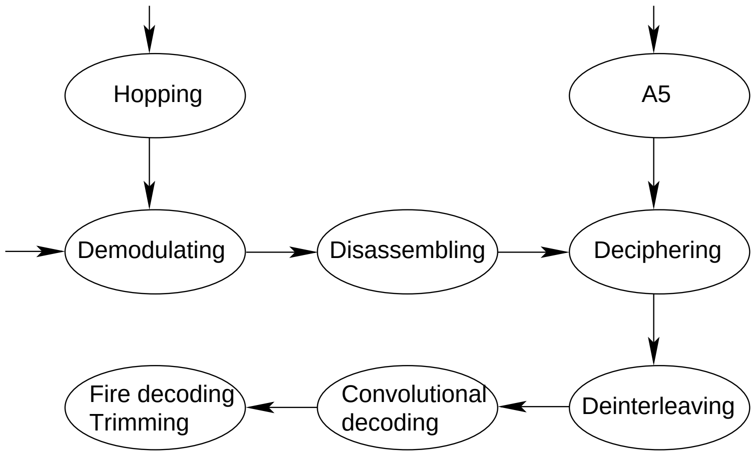


Figure 4.19: Decoding of a GSM dedicated signalling channel

demodulated bursts are disassembled by the disassembling task. The resulting digital blocks are deciphered by the deciphering task based on a key generated by the A5 task. The deciphered block proceeds through bit deinterleaving, convolutional decoding (Viterbi decoding) and the so called fire decoding. The whole application runs on a single DSP processor.

In this example, there are two sources of variation in execution times. The demodulating task has both data and control intensive behaviour, which can cause pipeline hazards on the deeply pipelined DSP it runs on. Its execution time probability density is derived from the input data streams and measurements. Another task will finally implement a deciphering unit. Due to the lack of knowledge about the deciphering algorithm A5 (its specification is not publicly available), the deciphering task execution time is considered to be uniformly distributed between an upper and a lower bound.

When two channels are scheduled on the DSP, the ratio of missed deadlines is 0 (all deadlines are met). Considering three channels assigned to the same processor, the analysis produced a ratio of missed deadlines, which was below the one enforced by the required QoS. It is important to note that using a hard real-time model with WCET, the system with three channels would result as unschedulable on the selected DSP. The underlying stochastic process for the three channels had 130 nodes and its analysis took 0.01 seconds. The small number of nodes is caused by the strong harmony among the task periods, imposed by the GSM standard.

Chapter 5

An Approximate Solution for Schedulability Analysis: The Multi-processor Case

When analysing multi-processor applications, one approach could be to decompose the analysis problem into several subproblems, each of them analysing the tasks mapped on one of the processors. We could attempt to apply the exact approach presented in the previous chapter in order to solve each of the subproblems. Unfortunately, in the case of multi-processors and with the assumption of data dependencies among tasks, this approach cannot be applied. The reason is that the set of ready tasks cannot be determined based solely on the information regarding the tasks mapped on the processor under consideration.

An alternative approach would be to consider all the tasks and to construct the global state space of the underlying stochastic process accordingly. In principle, the exact approach presented in the previous chapter could be applied in this case. However, the number of possible execution traces, and implicitly the stochastic process, explodes due to the parallelism provided by the application platform. As shown, the analysis has to store the probability distributions z_i for each process state in the sliding window of states, leading to large amounts of needed memory and limiting the appropriateness of this approach to very small multi-processor applications. Moreover, the number of convolutions $z_i * \epsilon_j$ would also explode, leading to prohibitive analysis times.

The challenge taken in this chapter is to analyse a multi-processor system with acceptable accuracy without the need to explicitly store and compute the memory consuming distributions z_i and to avoid the computations of the convolutions.

Thus, the generalised task execution time probability distributions are approximated by weighted sums of convoluted exponential distributions, leading to a large continuous time Markov chain (as opposed to semi-Markov processes in the previously presented approach). Such a Markov chain is much larger, but, as the state holding times probability distributions are exponential, there is no need to explicitly store their distributions, leading to a much more efficient use of the analysis memory. Moreover, by construction, the Markov chain exhibits regularities in its structure. These regularities are exploited during the analysis such that the infinitesimal generator of the chain is constructed on-the-fly, saving additional amounts of memory. In addition, the solution of the continuous time Markov chain does not imply any computation of convolutions. As a result, multi-processor applications of realistic size may be analysed with sufficient accuracy. Moreover, by controlling the precision of the approximation of the ETPDFs, the designer may trade analysis resources for accuracy.

5.1 Approach outline

In order to extract the desired performance metrics, the underlying stochastic process corresponding to the application has to be constructed and analysed. When considering arbitrary execution time probability distribution functions, the resulting process is a 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 [KS96], the process would be a continuous time Markov chain which would be easier to solve.

The outline of our approach is depicted in Figure 5.1. As a first step, we generate a model of the application as a Generalised Stochastic Petri Net (GSPN). We use this term in a broader sense than the one defined by Balbo [BCFR87], allowing arbitrary probability distributions for the firing delays of the timed transitions. This step is detailed in the next section.

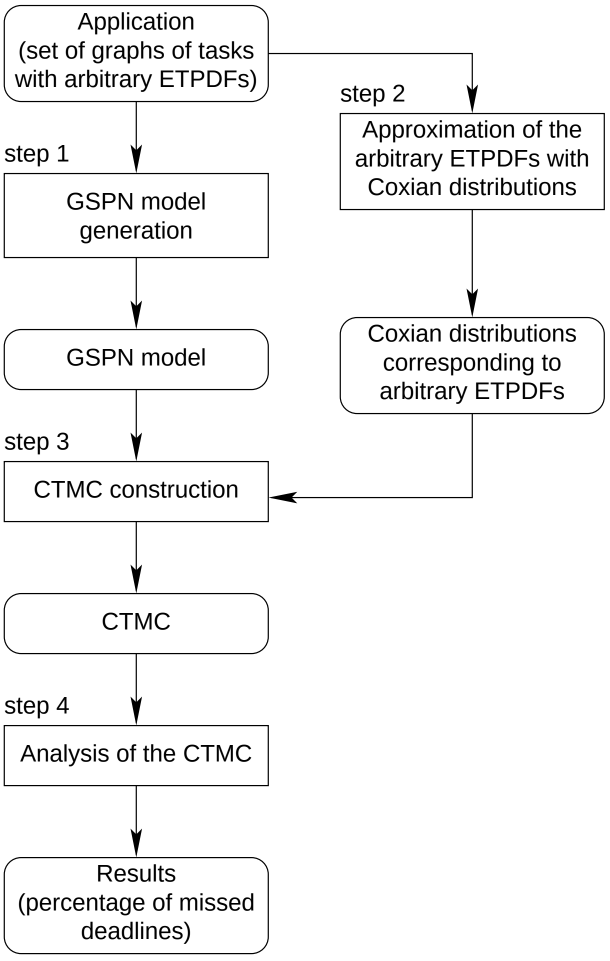


Figure 5.1: Approach outline

The second step implies the approximation of the arbitrary real-world ETPDFs with Coxian distributions, i.e. weighted sums of convoluted exponential distributions. Some details regarding Coxian distributions and the approximation process follow in Section 5.3.

In the third step, the tangible reachability graph of the GSPN is obtained. Its nodes correspond to states in a semi-Markov process (SMP). 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 SMP are substituted by transitions with exponentially distributed firing interval probabilities from the Coxian distributions. What results is a CTMC, however much larger than the SMP. The construction procedure of the CTMC is detailed in Section 5.4.

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

5.2 Intermediate model generation

As a first step, starting from the task graph model given by the designer, an intermediate model based on Generalised Stochastic Petri Nets (GSPN) [BCFR87] 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 [BCFR87], 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 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.

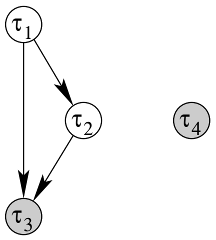


Figure 5.2: Task graphs

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 semi-Markov process.

We illustrate the construction of the GSPN based on an example. Let us consider the task graphs in Figure 5.2. Tasks τ_1, τ_2 and τ_3 form graph G_1 while G_2 consists of task τ_4. τ_1 and τ_2 are mapped on processor P_1 and τ_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 5.3. 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 τ_i is running. The probability distribution of the firing delay of transition e_i is equal to the ETPDF of task τ_i. As soon as e_i fires, it means that the instantiation of τ_i completed execution and leaves the system. The task priorities are modelled by prioritising the immediate transitions j_i.

The periodic arrival of graph instantiations is modelled by means of the transition Gen with the deterministic delay $Tick$. Gen fires every $Tick$ time units, where $Tick$ is the greatest common divisor of the graph periods. As soon as Gen has fired $\pi_i/Tick$ times, the transition v_i fires and a new instantiation of task graph G_i demands execution. (In our example, we considered $\pi_1/Tick = 3$ and $\pi_2/Tick = 2$.)

The place Bnd_i is initially marked with b_i tokens, meaning that at most b_i concurrent instantiations of G_i are allowed in the system. Whenever a new instantiation arrives (v_i fires), if Bnd_i does not contain b_i tokens, it means that there is at least one instantiation that is still running. In such case, because the task graph deadlines are equal to the task graph periods, by assumption, it follows that the running instantiation missed its deadline. If Bnd_i contains no tokens at all when v_i fires,

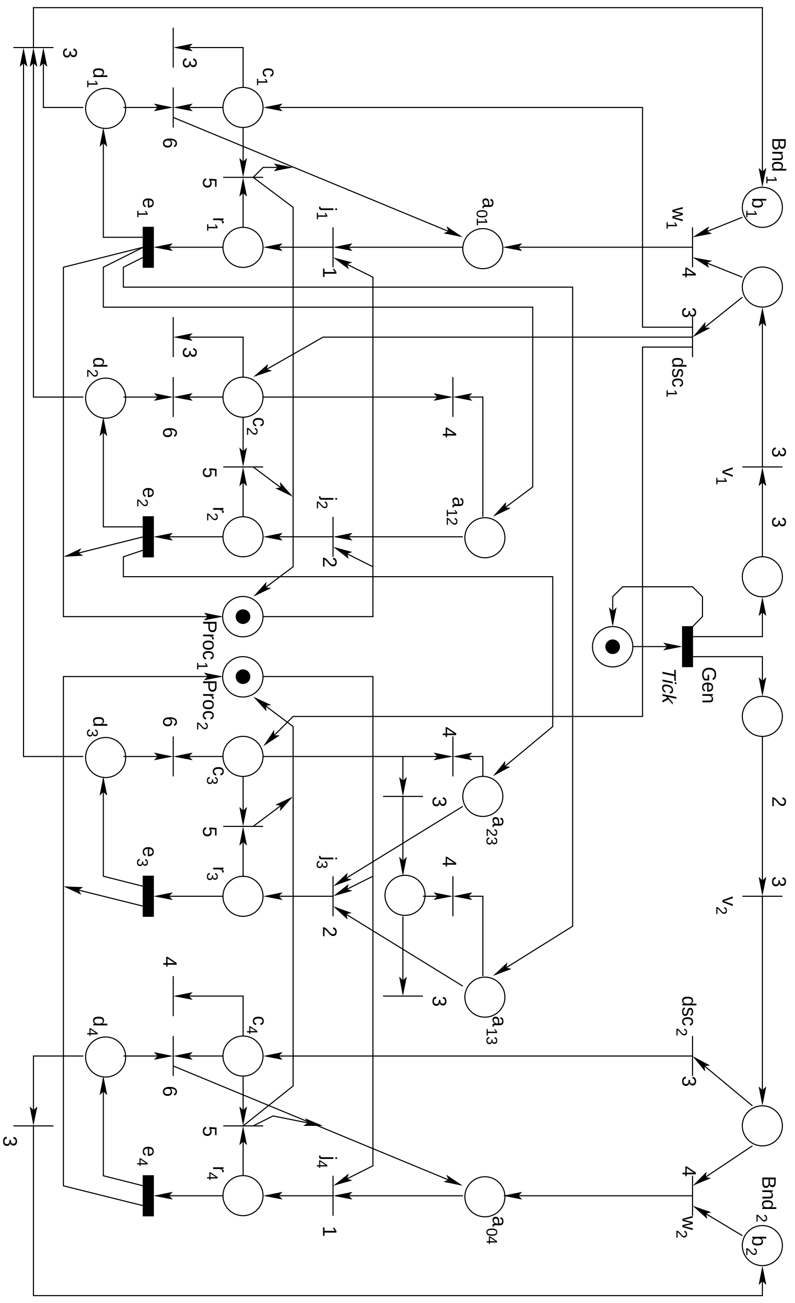


Figure 5.3: GSPN example

then the maximum number of instantiations of G_i are already present in the system and, therefore, the oldest one will be discarded. This is modelled by firing the immediate transition dsc_i and marking the places c_j, where one such place c_j corresponds to each task in G_i. The token in c_j will attempt to remove from the system an already completed task (a token from d_j), or a running task (a token from r_j), or a ready task (a token from a_{kj}), in this order. The transitions w_i have a higher priority than the transitions dsc_i, in order to ensure that a instantiation of G_i is discarded only when Bnd_i contains no tokens (there already are b_i concurrently active instantiations of G_i in the system). The structure of the GSPN is such that a newly arrived instantiation is always accepted in the system.

In our example, the mutual exclusion of the execution of tasks mapped on the same processor is modelled by means of the places $Proc_1$ and $Proc_2$. The data dependencies among the tasks are modelled by the arcs $e_2 \to a_{23}$, $e_1 \to a_{13}$ and $e_1 \to a_{12}$. Once a task graph instantiation leaves the system (the places d_j are marked), a token is added to Bnd_i.

The underlying SMP is extracted from the Petri Net by building its tangible reachability graph. The SMP 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 5.3 and Section 5.4.

5.3 Coxian distribution approximation

Coxian distributions were introduced by Cox [Cox55] 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 5.4 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 5.4(a). This transition can be replaced by the Petri Net in Figure 5.4(b), where hollow rectangles

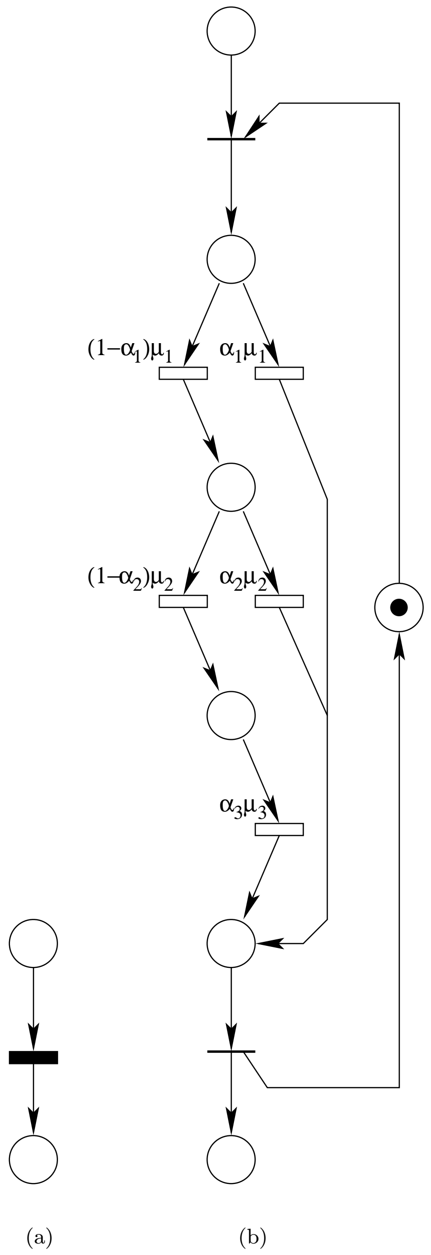


Figure 5.4: Coxian approximation with three stages

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 and a certain number of stages r, find $\mu_i, 1 \leq i \leq r$, and $\alpha_i, 1 \leq i \leq r-1$ ($\alpha_r = 1$), such that the quality of approximation of the given distribution by the Coxian distribution with r stages is maximised. This is usually done in the complex space by minimising the distance between the Fourier transform $X(j\omega)$ of the Coxian distribution and the computed Fourier transform of the distribution to be approximated. The minimisation is a typical interpolation problem and can be solved by various numerical methods [PTVF92]. We use a simulated annealing approach that minimises 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 generalised transitions of the type depicted in Figure 5.4(a) with subnets of the type depicted in Figure 5.4(b) the SMP 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.

5.4 CTMC construction and analysis

Let S be the set of states of the SMP underlying the Petri Net. This SMP corresponds to the TRG of the Petri Net model. Let $\mathcal{M} = [m_{ij}]$ be a square matrix of size $|S| \times |S|$ where $m_{ij} = 1$ if there exists a transition from the state s_i to the state s_j in the SMP and $m_{ij} = 0$ otherwise. 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 (equivalently, the set of tasks that are running in the states belonging to that 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 three independent tasks, each of them mapped on a different processor. In this case, 8 clusters can be

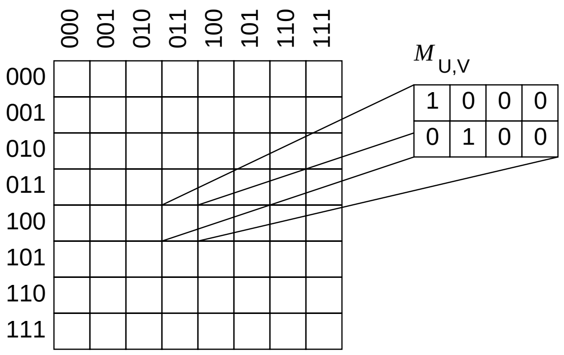


Figure 5.5: The matrix corresponding to a SMP

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. The cluster labelled with 101, for example, contains states in which the tasks τ_1 and τ_3 are running.

Figure 5.5 depicts the matrix $\mathcal{M}$ corresponding to the SMP of this example application. The rows and columns in the figure do not correspond to individual rows and columns in $\mathcal{M}$. Each row and column in Figure 5.5 corresponds to one *cluster* of states. The row labelled with 100, for example, as well as the column labelled with the same binary number, indicate that the task τ_1 is running in the states belonging to the cluster labelled with 100. Each cell in the figure does not correspond to a matrix element but to a submatrix $\mathcal{M}_{l_i,l_j}$, where $\mathcal{M}_{l_i,l_j}$ is the incidence matrix corresponding to the clusters labelled with l_i and l_j (an element of $\mathcal{M}_{l_i,l_j}$ is 1 if there is a transition from the state corresponding to its row to the state corresponding to its column, and it is 0 otherwise). The submatrix $\mathcal{M}_{U,V}$ at the intersection of the row labelled with $U = 100$ and the column labelled with $V = 011$ is detailed in the figure. The cluster labelled with $U = 100$ contains 2 states, while the cluster labelled with $V = 011$ contains 4 states. As shown in the figure, when a transition from the first state of the cluster labelled with U occurs, the first state of the cluster labelled with V is reached. This corresponds to the case when τ_1 completes execution (τ_1 is the only running task in the states belonging to the cluster labelled with U) and τ_2 and τ_3 are subsequently started (τ_2 and τ_3 are the running tasks in the states belonging to the cluster labelled with V).

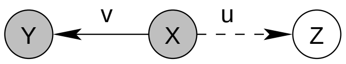


Figure 5.6: Part of a SMP

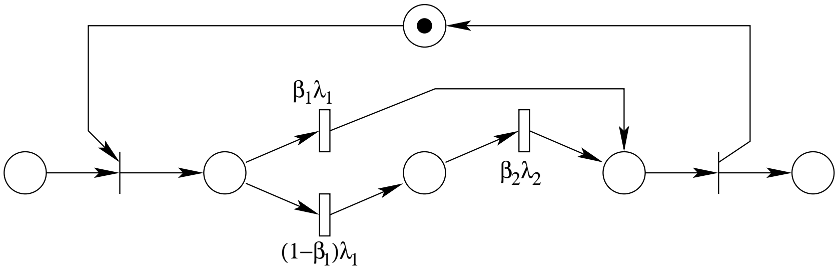


Figure 5.7: Coxian approximation with two stages

Once we have the matrix $\mathcal{M}$ corresponding to the underlying SMP, 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 (S is the set of states in the SMP), resulting an expanded state space S', the state space of the approximating CTMC. We have to construct a matrix $\mathcal{Q}$ of size $|S'| \times |S'|$, the so called infinitesimal generator of the approximating CTMC. The construction of $\mathcal{Q}$ is done cell-wise: for each submatrix of $\mathcal{M}$, a corresponding submatrix of $\mathcal{Q}$ will be generated. Furthermore, null submatrices of $\mathcal{M}$ will result in null submatrices of $\mathcal{Q}$. A cell $\mathcal{Q}_{U,V}$ of $\mathcal{Q}$ will be of size $G \times H$, where

$$G = |U| \cdot \prod_{i \in EnU} r_i$$

$$H = |V| \cdot \prod_{i \in EnV} r_i$$

and U and V are clusters of states, $EnU = \{k : \text{transition } e_k \text{ corresponding to the execution of task } \tau_k \text{ is enabled in } U\}$, $EnV = \{k : \text{transition } e_k \text{ corresponding to the execution of task } \tau_k \text{ is enabled in } V\}$, and r_k is the number of stages we use in the Coxian approximation of the ETPDF of task τ_k.

We will illustrate the construction of a cell in $\mathcal{Q}$ from a cell in $\mathcal{M}$ 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 SMP depicted in

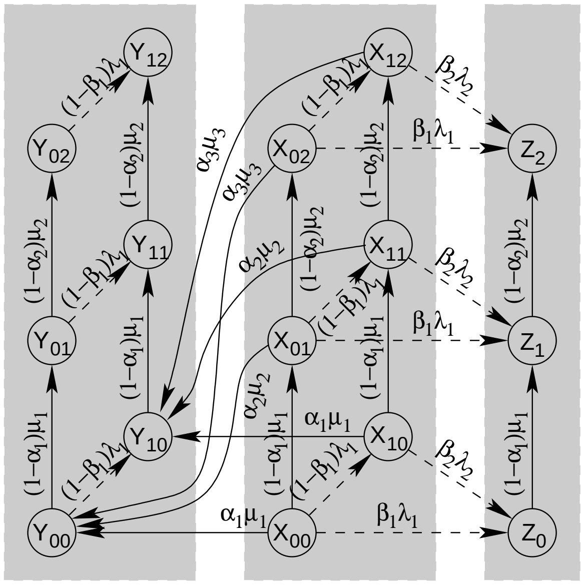


Figure 5.8: Expanded Markov chain

Figure 5.6. Two tasks, τ_u and τ_v, are running in the states X and Y. These two states belong to the same cluster, labelled with 11. Only task τ_v is running in state Z. State Z belongs to cluster labelled with 10. If task τ_v finishes running in state X, a transition to state Y occurs in the SMP. This corresponds to the situation when a new instantiation of τ_v becomes active immediately after the completion of a previous one. When task τ_u finishes running in state X, a transition to state Z occurs in the SMP. This corresponds to the situation when a new instantiation of τ_u is not immediately activated after the completion of a previous one. Consider that the probability distribution of the execution time of task τ_v is approximated with the three stage Coxian distribution depicted in Figure 5.4(b) and that of τ_u is approximated with the two stage Coxian distribution depicted in Figure 5.7. The resulting CTMC corresponding to the part of the SMP in Figure 5.6 is depicted in Figure 5.8. The edges between the states are labelled with the average firing rates of the transitions of the Coxian distributions. Dashed edges denote state changes in the CTMC caused by firing of transitions belonging to the Coxian approximation of the ETPDF of τ_u. Solid edges denote state changes in the CTMC caused by firing of transitions belonging to the

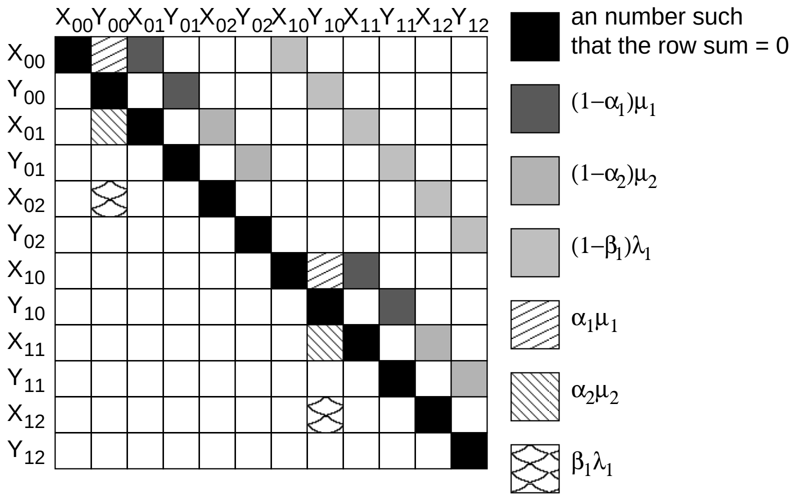

X00 Y00 X01 Y01 X02 Y02 X10 Y10 X11 Y11 X12 Y12
X00
Y00
X01
Y01
X02
Y02
X10
Y10
X11
Y11
X12
Y12
an number such that the row sum = 0
(1−α1)μ1
(1−α2)μ2
(1−β1)λ1
α1μ1
α2μ2
β1λ1

Figure 5.9: The cell $\mathcal{Q}_{(11),(11)}$ corresponding to the example in Figure 5.8

Coxian approximation of the ETPDF of τ_v. The state Y_{12}, for example, corresponds to the situation when the SMP in Figure 5.6 would be in state Y and the first two of the three stages of the Coxian distribution approximating the ETPDF of τ_v (Figure 5.4(b)) and the first stage out of the two of the Coxian distribution approximating the ETPDF of τ_u (Figure 5.7) have fired.

Let us construct the cell $\mathcal{Q}_{(11),(11)}$ on the main diagonal of $\mathcal{Q}$, situated at the intersection of the row and column corresponding to the cluster labelled with 11. The cell is depicted in Figure 5.9. The matrix $\mathcal{Q}_{(11),(11)}$ contains the average transition rates between the states X_{ij} and Y_{ij}, $0 \leq i \leq 1$, $0 \leq j \leq 2$, of the CTMC in Figure 5.8 (only states X and Y belong to the cluster labelled with 11). The observed regularity in the structure of stochastic process in Figure 5.8 is reflected in the expression of $\mathcal{Q}_{(11),(11)}$ as shown in Figure 5.9. Because $\mathcal{Q}$ 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 5.8. The expression of $\mathcal{Q}_{(11),(11)}$ is given below:

$$\mathcal{Q}_{(11),(11)} = (\mathcal{A}_u \oplus \mathcal{A}_v) \otimes I_{|11|} + I_{r_u} \otimes \mathcal{B}_v \otimes e_{r_v} \otimes \mathcal{D}_v$$

where

$$\mathcal{A}_u = \begin{bmatrix} -\lambda_1 & (1-\beta_1)\lambda_1 \\ 0 & -\lambda_2 \end{bmatrix}$$

$$\mathcal{B}_v = \begin{bmatrix} \alpha_1\mu_1 \\ \alpha_2\mu_2 \\ \alpha_3\mu_3 \end{bmatrix}$$

$$\mathcal{D}_v = \begin{bmatrix} 0 & 1 \\ 0 & 0 \end{bmatrix}$$

and

$$\mathcal{A}_v = \begin{bmatrix} -\mu_1 & (1-\alpha_1)\mu_1 & 0 \\ 0 & -\mu_2 & (1-\alpha_2)\mu_2 \\ 0 & 0 & -\mu_3 \end{bmatrix}$$

$$e_{r_v} = \begin{bmatrix} 1 & 0 & 0 \end{bmatrix}$$

$|11|$ denotes the size of the cluster labelled with 11. I_i is the identity matrix of size $i \times i$, r_i indicates the number of stages of the Coxian distribution that approximates the ETPDF of task τ_i. $\oplus$ and $\otimes$ are the Kronecker sum and product of matrices, respectively.

In general, a matrix $\mathcal{A}_k = [a_{ij}]$ is an $r_k \times r_k$ 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} characterise the i^{th} stage of the Coxian distribution approximating a transition t_k.

A matrix $\mathcal{B}_k = [b_{ij}]$ is an $r_k \times 1$ matrix and $b_{i1} = \alpha_{ki} \cdot \mu_{ki}$. 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 $\mathcal{D}_k = [d_{ij}]$ corresponding to a cell U, V is a $|U| \times |V|$ matrix defined as follows:

$$d_{ij} = \begin{cases} 1 & \text{if an edge labelled with } k \text{ links} \\ & \text{the } i^{th} \text{ state of } U \text{ with the } j^{th} \text{ state of } V \\ 0 & \text{otherwise} \end{cases}$$

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

$$\mathcal{Q}_{U,U} = \left(\bigoplus_{i\in U} \mathcal{A}_i\right) \otimes I_{|U|} + \sum_{i\in U} \left(\bigotimes_{\substack{j\in U\\ j>i}} I_{r_j}\right) \otimes \mathcal{B}_i \otimes e_{r_i} \otimes \left(\bigotimes_{\substack{j\in U\\ j<i}} I_{r_j}\right) \otimes \mathcal{D}_i$$

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:

$$\mathcal{Q}_{U,V} = \sum_{i \in U} \left(\bigotimes_{j \in U \cup V} \mathcal{F}_{ij} \right) \otimes \mathcal{D}_i \tag{5.1}$$

The matrices $\mathcal{F}$ are given by the following expression:

$$\mathcal{F}_{ij} = \begin{cases} v_{r_j} & \text{if } j \in U \wedge j \notin V \wedge j \neq i \\ I_{r_j} & \text{if } j \in U \wedge j \in V \wedge j \neq i \\ \mathcal{B}_i & \text{if } j \notin V \wedge j = i \\ \mathcal{B}_i \otimes e_{r_i} & \text{if } j \in V \wedge j = i \\ e_{r_j} & \text{if } j \notin U \end{cases} \tag{5.2}$$

where $v_{r_k} = [v_{i1}]$ is a $r_k \times 1$ matrix, $v_{i1} = 1$, $1 \leq i \leq r_k$.

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

$$\pi \cdot \mathcal{Q} = 0$$

where π is the steady-state probability vector and $\mathcal{Q}$ the infinitesimal generator of the CTMC.

Let us consider the edge $X \to Z$ in the SMP in Figure 5.6. The edges $X_{00} \to Z_0$, $X_{10} \to Z_0$, $X_{01} \to Z_1$, $X_{11} \to Z_1$, $X_{02} \to Z_2$, and $X_{12} \to Z_2$ in the CTMC in Figure 5.8, that approximates the SMP in Figure 5.6, correspond to the edge $X \to Z$ in the SMP. The expected transition rate of $X \to Z$ can be approximated by means of the expected transition rates of the corresponding edges in the CTMC and is given by the expression

$$(\pi_{X_{00}} + \pi_{X_{01}} + \pi_{X_{02}}) \cdot \beta_1 \lambda_1 + (\pi_{X_{10}} + \pi_{X_{11}} + \pi_{X_{12}}) \cdot \beta_2 \lambda_2,$$

where β_1, β_2, λ_1, and λ_2 are characterising the Coxian distribution that approximates the probability distribution of the delay of the transition $X \to Z$ (in this case, the ETPDF of τ_v). $\pi_{X_{ij}}$ is the probability of the CTMC being in state X_{ij} after the steady state is reached. The probabilities $\pi_{X_{ij}}$ are obtained as the result of the numerical solution of the CTMC.

As explained in Section 5.2, if there already exists at least one active instantiation of a task graph G_i in the system when a new instantiation of G_i arrives, then the most recent instantiation of G_i present in the system misses its deadline (which is equal to the period). In the Petri Net model (see Figure 5.3), this event corresponds to the sequence of firings of the timed transition Gen, followed by v_i and either the immediate transition w_i if the number of tokens in Bnd_i is less than b_i but larger than 0, or the immediate transition dsc_i if Bnd_i is unmarked. Each of these two situations corresponds to an edge in the stochastic process. Thus, the expected deadline miss rate of G_i can be obtained by computing the expected transition rates (as shown above for $X \rightarrow Z$) for the edges corresponding to a deadline miss. The expected deadline miss ratio for the task graph G_i is then computed by multiplying the obtained expected deadline miss rate by the period of task graph G_i.

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

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

where $E = \{i$: transition e_i corresponding to the execution of task τ_i is enabled in the dominant cluster$\}$ and $|\mathcal{M}|$ denotes the number of non-zero elements of $\mathcal{M}$, the matrix corresponding to the SMP. In the context of multiprocessor scheduling, E may have at most M elements (M=number of processors). The above formula shows that the increase in the size of the CTMC, relative to the initial SMP, is mainly dictated by the number of processors and the number of stages for the approximation of the probability distributions (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 SMP also depends on the number of tasks. The SMP is reflected in the matrix $\mathcal{M}$ and, thus, has an influence on the dimension of the resulted CTMC.

The dimension of matrix $\mathcal{Q}$, 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 $\mathcal{Q}_{U,U}$ and $\mathcal{Q}_{U,V}$, the matrix $\mathcal{Q}$ is completely specified by means of the matrices $\mathcal{A}_i$, $\mathcal{B}_i$, and $\mathcal{D}_i$, 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 $\mathcal{A}_i$, $\mathcal{B}_i$, and $\mathcal{D}_i$ 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 (number of states) of the SMP. It is easy to observe that π is as large as a row (column) of $\mathcal{Q}$. 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 factorisations that exploit the particular structure of $\mathcal{Q}$.

5.5 Experiments

We performed four sets of experiments. All were run on an AMD 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

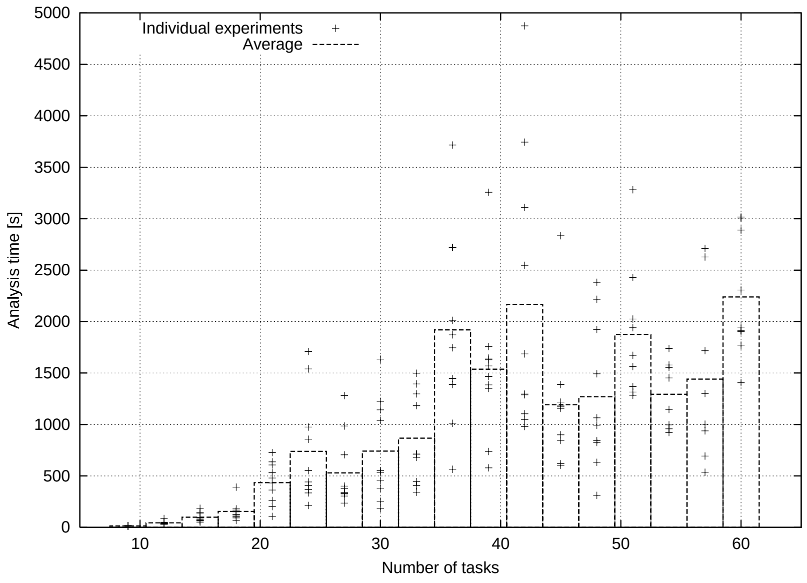


Figure 5.10: Analysis time vs. number of tasks

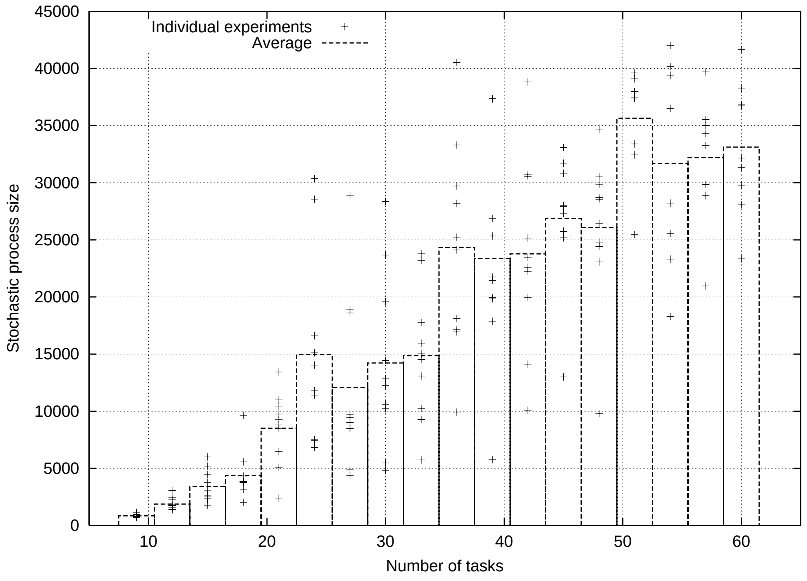


Figure 5.11: Stochastic process size vs. number of tasks

underlying architecture consists of two processors. The dependency between the needed analysis time and the number of tasks is depicted in Figure 5.10. 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 5.10. The dependency of the stochastic process size as a function of the number of tasks is plotted in Figure 5.11.

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 5.12.

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

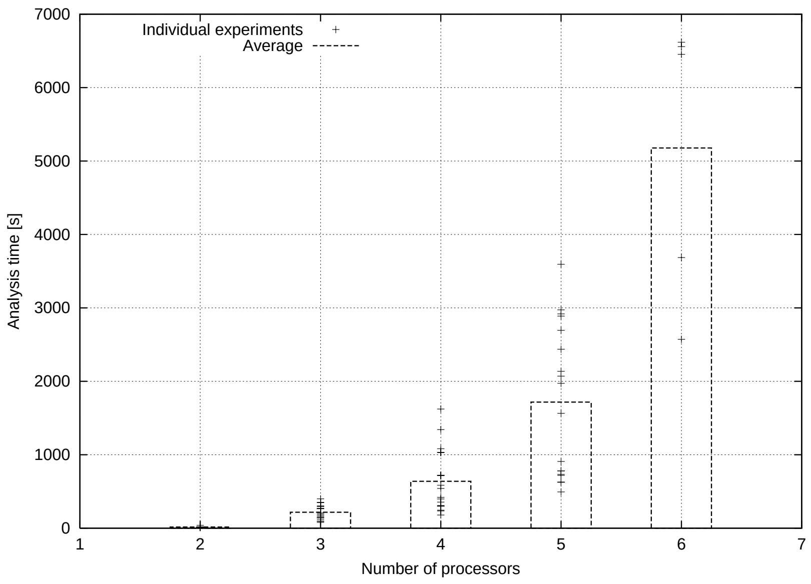


Figure 5.12: Analysis time vs. number of processors

over the 98 sets of graphs and the results are plotted in Figure 5.13. Recall that $|S|$ is the size of the SMP 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 SMP. As shown in Section 5.4, in the worst case, the growth factor is

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

As can be seen from Figure 5.13, the real growth factor is smaller than the theoretical upper bound. It is important to emphasise that the matrix $\mathcal{Q}$ corresponding to the CTMC needs not to be stored, but only a vector with the length corresponding to a column of $\mathcal{Q}$. The growth of the vector length with the number of Coxian stages used for approximation can be easily derived from Figure 5.13. 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 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

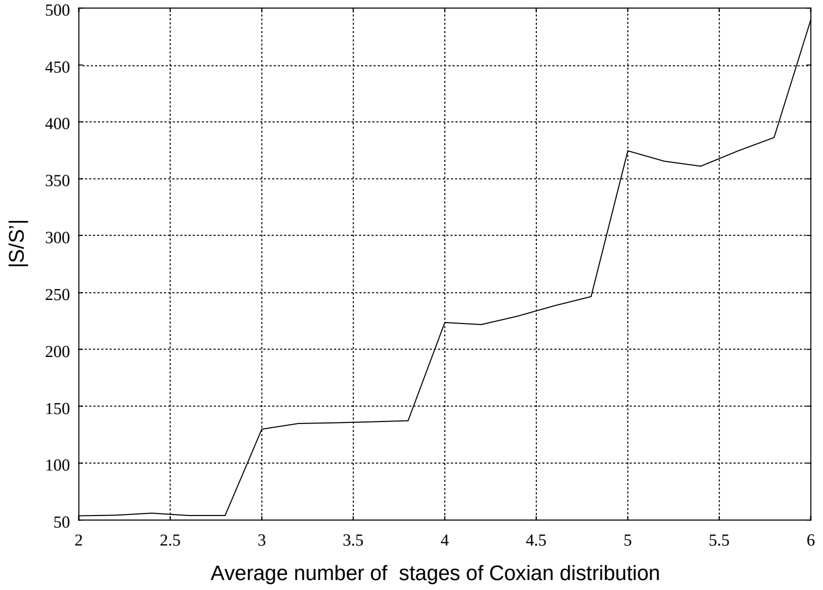


Figure 5.13: Increase in stochastic process size with number of stages for approximating the arbitrary ETPDFs

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

Table 5.1: Accuracy vs. number of stages

Chapter 4. As mentioned there, that approach is an exact one based on solving the underlying SMP. However, because of complexity reasons, it can handle only mono-processor systems. Therefore, we applied the approach presented in this paper to a mono-processor 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 5.1. The generalised ET-PDF used in this experiment were created by drawing Bezier curves that interpolated randomly generated control points. It can be observed that good quality results can already be obtained with a relatively small number of stages.

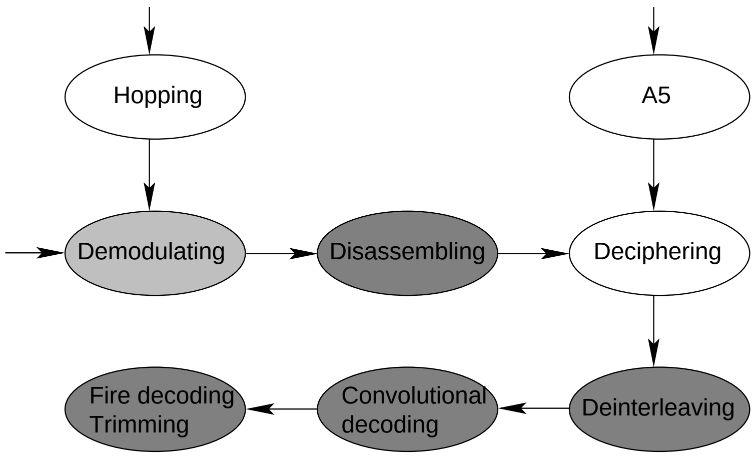


Figure 5.14: Decoding of a GSM dedicated signalling channel

Finally, we considered the same example from the mobile communication area introduced in Section 4.6, this time considering an architecture consisting of three processors. Figure 5.14 depicts the application task graph. The shades of the ovals indicate the mapping of the tasks. The demodulation task is mapped on a dedicated processor, the hopping, deciphering and A5 tasks are mapped on a general purpose processor, and the disassembling, deinterleaving, convolutional decoding and the so called fire decoding tasks are mapped on a DSP processor.

In the case of 8 tasks, the analysis reported an acceptable miss deadline ratio after an analysis time of 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.

Chapter 6

Extensions

The discussions in Chapter 4 and 5 were based on the assumptions and the problem formulation introduced in Chapter 3. One of the main concerns has been to master the complexity generated by the relatively unrestricted nature of those assumptions. In this chapter, we discuss some possible further relaxations of the assumptions introduced in Chapter 3 and their consequences on the complexity of the solution.

The following extensions of the assumptions were considered:

1. The tasks belonging to the same task graph may have different periods.

2. The relative deadline of a task may be shorter than the task period.

3. When there are already b_i concurrently active instantiations of a task graph G_i in the system at the arrival time of a new instantiation, instead of discarding the oldest instantiation, the newly arrived one is rejected.

Furthermore, the problem formulation has been extended by adding the expected deadline miss ratio per *task* (not only per task graph) to the output of the analysis. This feature provides more insight to the designer relative to the causes of deadline misses.

Each of these extensions is treated in a separate section.

6.1 Task periods

As presented in Chapter 3, all the tasks belonging to a task graph have the same period. This assumption has been relaxed as follows. Each task $\tau_i \in G_j$ has its own period π_{ij}, with the restriction that π_{ij} is a common multiple of all periods of the tasks in $^\circ\tau_i$ (π_{kj} divides π_{ij}, where $\tau_k \in {}^\circ\tau_i$). In this case, π_j will denote the period of the task graph G_j and π_j is equal to the least common multiple of all π_{ij}, where π_{ij} is the period of τ_i and $\tau_i \in V_j$.

6.1.1 The effect on the complexity of the exact solution

In almost all of the cases, the relaxation concerning task periods increases the analysis complexity. This is because the number of PMIs is increased, leading to an increase of the underlying stochastic process state space.

In order to assess this increase, the following experiment was carried out. 126 sets of task graphs were randomly generated. Each set contained between 12 and 27 tasks grouped in 3 to 9 task graphs. Two test cases were derived from each set of task graphs. In the first, the task periods were assigned as described in this section and the whole task graph had a period equal to the LCM of the individual task periods. In the second test case, all the tasks belonging to the same graph were assigned the period of the corresponding task graph as resulted in the first test case. By doing so, it was ensured that in both test cases, the LCM of task periods in both test cases are equal, in order not to influence the result. The number of states of the underlying stochastic process was measured for both test cases. Table 6.1 summarises the results that were averaged for task sets with the same cardinality. As expected, it can be seen that introducing different periods to tasks belonging to the same task graph increases the underlying stochastic process.

Tasks	Stochastic process size		Increase
	Identical task periods	Individual task periods	
12	922.14	1440.42	56.20%
15	2385.85	3153.47	32.17%
18	2034.00	4059.42	99.57%
21	14590.66	17012.80	16.60%
24	19840.19	35362.85	78.23%
27	42486.28	64800.19	52.52%

Table 6.1: Task vs. task graph periods

6.1.2 The effect on the complexity of the approximate solution

Two issues have to be considered: the modifications performed on the GSPN that models the application and the impact on the analysis process.

Consider the tasks τ_1, τ_2 and τ_3 in Figure 5.2 and let their periods be $\pi_1 = 2$, $\pi_2 = 4$ and $\pi_3 = 8$. τ_2 executes once whenever τ_1 executes twice, and τ_3 executes once whenever τ_1 executes four times and τ_2 executes twice. This can be modelled by a GSPN similar to the one in Figure 5.3 by setting the multiplicity of the arc $a_{12} \rightarrow j_2$ to π_2/π_1 (2), the one of arc $a_{13} \rightarrow j_3$ to π_3/π_1 (4) and the one of arc $a_{23} \rightarrow j_3$ to π_3/π_2 (2).

Observe that, under the original assumptions, in the unmodified GSPN depicted in Figure 5.3, there is no ambiguity in the situation when a place a_{ki} is marked with more than one token. This situation would occur only because of the multiple concurrently active instantiations of a graph G. Under the extended assumptions, multiple tokens may appear in a place a_{ki} because several instantiations of task τ_k of the *same* instantiation of G have completed, or because several instantiations of task τ_k of *different* instantiations of G have completed. Therefore, either we assume $b_i = 1$, $\forall 1 \leq i \leq h$, in the case of the extended assumptions about task periods, or we significantly change the GSPN, which also increases its complexity, such that it distinguishes between tokens modelling different graph instantiations.

In the case of discarding, under the original assumptions, at most *one* token was removed from a place a_{ki}. Under the extended assumptions, there might exist q tokens in a_{ki}, $0 \leq q < \pi_i/\pi_k$. Therefore, a marking dependent arc multiplicity has to be introduced in order to remove exactly q tokens from a_{ki}.

If a task graph has several tasks with no predecessors and these tasks have different periods, a new scheme for modelling the task arrivals has to be designed, which is different from that implemented with the transitions v_1 and v_2 in Figure 5.3. Moreover, the firing delay $Tick$ of the timed transition Gen is given by the greatest common divisor of the *task* periods and not of the task *graph* periods.

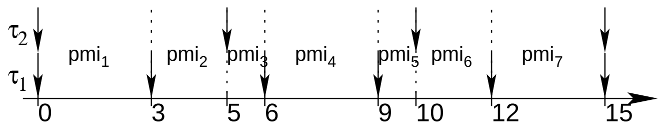
The main effect of introducing individual task periods on the GSPN is an increase in the number of tokens circulating in the GSPN at a certain time. This leads to a significant increase in the tangible reachability graph of the GSPN and implicitly in the stochastic process size. Therefore, only small applications can be analysed under the assumption of individual task periods.

6.2 Deadlines shorter than periods

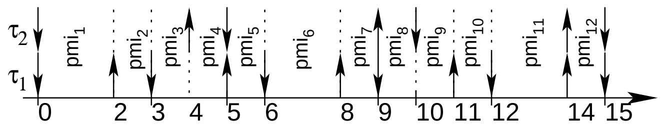
The restriction that the task deadlines are equal to the corresponding periods can be relaxed too. When letting the relative deadlines of tasks to be shorter than the task periods, two conflicting effects on the analysis complexity are observed. On one hand, if a task misses its deadline and if the late task policy dictates that the task graph has to be discarded, fewer behaviours are possible in the interval from the discarding and the graph deadline. This would reduce the stochastic process size. On the other hand, this earlier discarding introduces new contexts for the behaviours of the rest of tasks. This would increase the stochastic process size.

6.2.1 The effect on the complexity of the exact solution

In almost all of the cases, letting $\delta_i \leq \pi_i$ leads to a significant increase in the number of PMIs as seen in Figure 6.1. The considered application consists of two independent tasks with periods 3 and 5 respectively. Figure 6.1(a) depicts the resulting PMIs if the relative task deadlines are equal to the respective task periods, while Figure 6.1(b) depicts the resulting PMIs if the relative task deadlines are 2 and 4 respectively.



(a) PMIs if $\delta_i = \pi_i$



(b) PMIs if $\delta_i \leq \pi_i$

Figure 6.1: PMIs if deadlines are less than the periods

Tasks	Stochastic process size		Increase
	$\delta_i = \pi_i$	$\delta_i < \pi_i$	
12	1440.42	2361.38	63.93%
15	3153.47	3851.90	22.14%
18	4059.42	5794.33	42.73%
21	11636.29	24024.35	106.46%
24	35142.60	48964.80	39.33%
27	35044.30	40218.60	14.76%

Table 6.2: Deadlines equal to periods vs. shorter deadlines

In order to assess the effect of this relaxation on the analysis complexity, the following experiment was carried out. 378 sets of task graphs were randomly generated, each set containing between 12 and 27 tasks grouped in 3 to 9 graphs. Two test cases were created for each task set. In the first, the deadlines are equal to the periods while in the second test case random deadlines, less than the periods, were generated. Table 6.2 summarises the results that were averaged for task sets with the same cardinality.

6.2.2 The effect on the complexity of the approximate solution

Let us consider the GSPN in Figure 5.3. The firing delay *Tick* of the timed transition *Gen* has to be modified in order to fire when a deadline arrived. Therefore, *Tick* equals the greatest common divisor of the task periods *and* of the relative deadlines. The GSPN has to be modified such that the places c_j are marked not when a new instantiation of a corresponding task graph arrives but when the deadline of the corresponding task graph arrives.

The additional places that have to be introduced lead to an increase in the tangible reachability graph and implicitly in the stochastic process size and analysis complexity.

6.3 Expected deadline miss ratios per task

The problem formulation can be extended by requiring the analysis to provide expected deadline miss ratios per *task* and not only per task *graph*.

Both in the case of the exact solution and in the case of the approximate solution, the sizes of the resulting stochastic processes are not influenced by this extension. The only difference is that there are more transitions in the stochastic process that correspond to deadline misses. The additional time required to compute the expected firing rates of these transitions is negligible as compared to the time required to find the steady state solution of the stochastic process.

6.4 Task rejection versus discarding

As formulated in Section 3.1.5, when there are b_i concurrently active instantiations of task graph G_i in the system, and a new instantiation of G_i demands service, the oldest instantiation of G_i is eliminated from the system. Sometimes, this behaviour is not desired, as the oldest instantiation might have been very close to finishing, and by discarding it, the invested resources (time, memory, bandwidth, etc.) are wasted.

Therefore, both our exact solution approach presented in Chapter 4 and our approximate solution approach presented in Chapter 5 have been extended to support a late task policy in which, instead of discarding the oldest instantiation of G_i, the newly arrived instantiation is denied service (rejected) by the system.

The following sections describe how this late policy is supported in both our approaches and what is the impact of this extension on the analysis complexity.

6.4.1 The effect on the complexity of the exact solution

In principle, the rejection policy is easily supported by only changing the `next_states` procedure in the algorithm presented in Section 4.5. However, this has a strong impact on the analysis complexity as shown in Table 6.3. The significant increase in the stochastic process size (about two orders of magnitude), can be explained considering the following example. Let s be the stochastic process state under analysis, let τ_j belonging to task graph G_i be the task running in s and let us consider that there are b_i concurrently active instantiations of G_i in the system. The execution time of τ_j may be very large, spanning over many PMIs. In the case of discarding, it was guaranteed that τ_j will stop running after at most $b_i \cdot \pi_i$ time units, because at that time moment it would be eliminated from the system. Therefore, when considering the discarding

Tasks	Avg. stoch. proc. size		Rel. incr.
	Disc.	Rej.	
12	2223.52	95780.23	42.07
15	7541.00	924548.19	121.60
18	4864.60	364146.60	73.85
21	18425.43	1855073.00	99.68
24	14876.16	1207253.83	80.15
27	55609.54	5340827.45	95.04

Table 6.3: Exact solution: Discarding compared to rejection

policy, the number of next states of a state s is upper bounded. When considering the rejection policy, this is not the case.

101 task sets of 12 to 27 tasks grouped in 2 to 9 task graphs were randomly generated. For each task set two analysis were performed, one considering the discarding policy and the other considering the rejection policy. The results were averaged for task sets with the same cardinality and shown in Table 6.3.

6.4.2 The effect on the complexity of the approximate solution

The rejection policy is supported by the approximate solution approach by changing the modelling of the application at the level of the intermediate representation, i.e. at the level of the GSPN.

The GSPN modelling the application in Figure 5.2, when considering the rejection policy, is depicted in Figure 6.2.

If there are b_i concurrently active instantiations of a task graph G_i in the system, then the place Bnd_i contains no tokens. If a new instantiation of G_i arrives in such a situation (v_i fires), then dsc_i will fire, "throwing away" the newly arrived instantiation. Although GSPNs like the one in Figure 6.2, modelling applications with rejection policy, seem simpler than the GSPN in Figure 5.3, modelling applications with discarding policies, the resulting tangible reachability graphs (and implicitly underlying semi-Markov processes) are larger, mainly because of the same reason discussed in the previous section.

In order to assess the impact of the rejection policy on the analysis complexity compared to the discarding policy, the following experiments were carried out. 109 task sets of 12 to 27 tasks grouped in 2 to 9 task graphs were randomly generated. For each task set two analysis were performed, one considering the discarding policy and the other

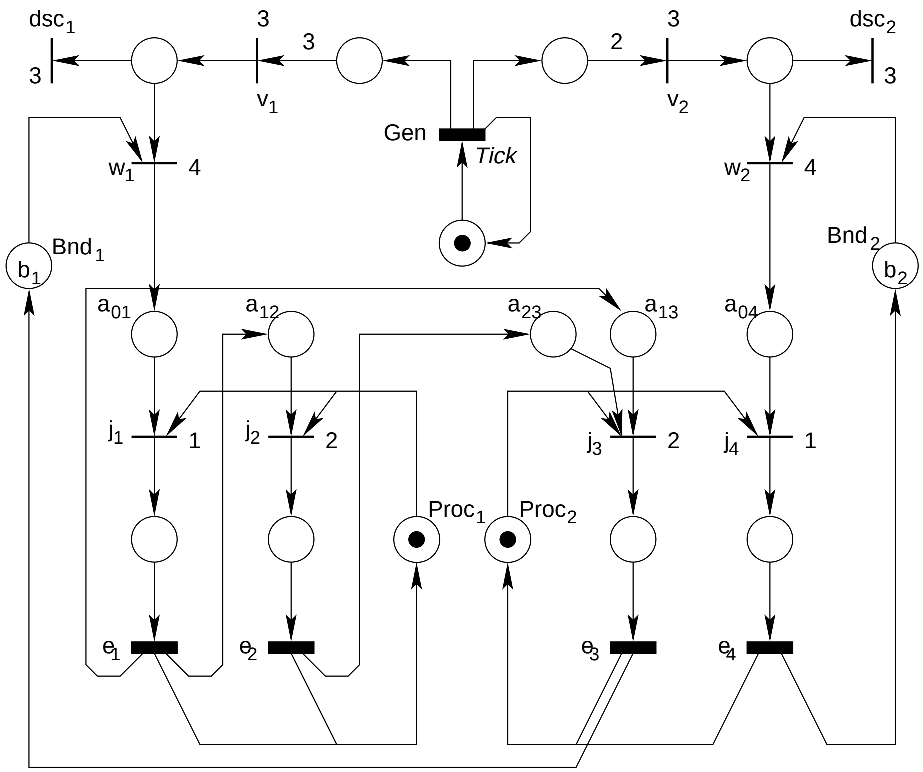


Figure 6.2: GSPN modelling the task graphs in Figure 5.2 in the case of the rejection policy

Tasks	Avg. SMP size		Rel. incr.
	Disc.	Rej.	
12	8437.85	18291.23	1.16
15	27815.28	90092.47	2.23
18	24089.19	194300.66	7.06
21	158859.21	816296.36	4.13
24	163593.31	845778.31	4.17
27	223088.90	1182925.81	4.30

Table 6.4: Approximate solution: Discarding compared to rejection

considering the rejection policy. The results were averaged for task sets with the same cardinality and shown in Table 6.4.

Even if the size of the SMP is larger in the case of the rejection policy than in the case of the discarding policy, the following property can be exploited in order to speed up the analysis of the stochastic process. It can be observed from Figure 6.2 that the firing of any timed transition cannot lead to the disabling of any other timed transition. This is the case because neither pre-emption nor discarding is allowed. Note that, as opposed to rejection, as soon as discarding is allowed (Figure 5.3), by firing the timed transition $Tick$ followed by the firing of the immediate transitions v_i, dsc_i, and the transition between the places c_i and r_i, a token from r_i is removed and the timed transition e_i is not anymore enabled.

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 τ_i.) This means that no transition in W_1 can be disabled when firing U, except possibly U itself ($W_1 \backslash \{U\} \subseteq W_2$). In other words, if the cardinality of $W_1 \backslash W_2$ 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 $|W_1 \backslash W_2| > 1$. This is used in order to determine the null submatrices of the matrix $\mathcal{M}$ (see page 69), and implicitly of the infinitesimal generator that will be constructed.

Consider an example application with four independent tasks, each of them mapped on a different processor. In this case, 16 clusters can be formed, each corresponding to a possible combination of simultaneously running tasks. Figure 6.3 depicts the matrix $\mathcal{M}$ corresponding to

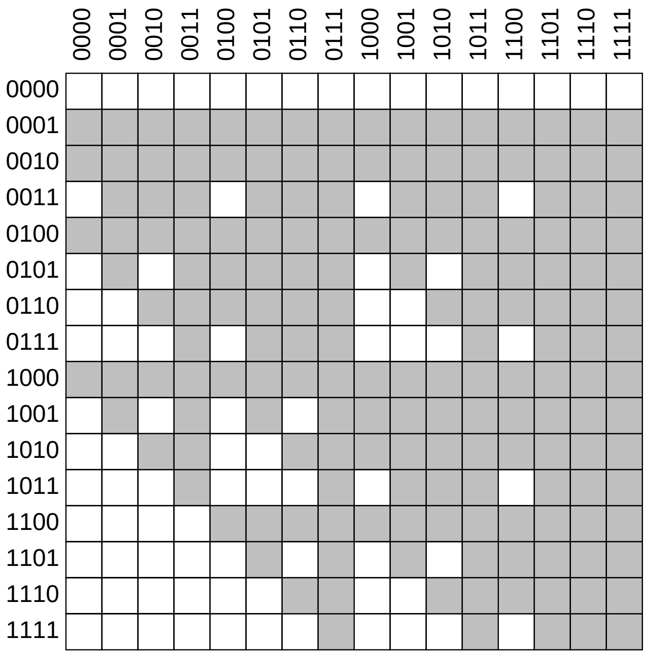


Figure 6.3: The matrix $\mathcal{M}$ in the case of the rejection policy

the SMP of this example application. As in Figure 5.9, the rows and columns in the figure do not correspond to individual rows and columns in $\mathcal{M}$ but to clusters of states. The shaded cells of $\mathcal{M}$ 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 τ_1, τ_2, and τ_4 are running, triggers a transition to a state where only the task τ_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 τ_1 completes execution, the stochastic process may transit to a state in which no task is running.

Because of the property described above, the expression for computing a submatrix $Q_{U,V}$ of the infinitesimal generator of the CTMC that approximates the SMP, slightly differs from the one given in Equation 5.1. The new expression of $Q_{U,V}$ is given below:

$$\mathcal{Q}_{U,V} = \sum_{i \in U} \left(\bigotimes_{j \in U \cup V} \mathcal{F}_{ij} \right) \otimes \mathcal{D}_i$$

$$\mathcal{F}_{ij} = \begin{cases} I_{r_j} & \text{if } j \in U \wedge j \in V \wedge j \neq i \\ \mathcal{B}_i & \text{if } j \notin V \wedge j = i \\ \mathcal{B}_i \otimes e_{r_i} & \text{if } j \in V \wedge j = i \\ e_{r_j} & \text{if } j \notin U \end{cases}$$

Note that the alternative $j \in U \wedge j \notin V \wedge j \neq i$ appearing in the expression of $\mathcal{F}_{ij}$ in Equation 5.2 is impossible in the case of rejection, reducing the number of non-null submatrices of $\mathcal{Q}$. Summing up, in the case of a rejection policy, the size of the CTMC (and implicitly of the matrix $\mathcal{Q}$) is larger than in the case of discarding. This, however, is compensated, to a certain extent, by the fact that the number of null elements in matrix $\mathcal{Q}$ is larger in the case of rejection, which limits the increase of the analysis time.

Chapter 7

Conclusions and Future Work

7.1 Conclusions

The design process of real-time embedded systems is an extremely complex task, requiring a disciplined and systematic methodology as well as the support of design space exploration and performance estimation tools.

It is of capital importance that the predicted performance of a not yet implemented system is an accurate estimate of the performance to be achieved by the implemented system. In order to obtain such accurate results and still not commit to an implementation, the performance estimation tool has to be fed with an architecture alternative, a mapping alternative, and a scheduling alternative.

This thesis presented two approaches for the performance estimation of real-time systems with stochastic execution times in the context mentioned above.

The first approach delivers exact results and it is applicable, but theoretically not limited, to mono-processor systems. In this case, applications under a fairly relaxed set of assumptions are considered. The innovative way of concurrently constructing and analysing the stochastic process underlying the application, as well as the economic method of parsing the stochastic process allows for the consideration of larger, real-world applications.

The second approach may trade result accuracy for analysis resources (time and memory space) and it is applicable to multi-processor systems as well. It relies on a designer controlled approximation of real-world execution time probability distribution functions by means of Coxian distributions and on the efficient way of solving the resulting Markov chain without the need to explicitly store the generator matrix.

Experiments were carried out for both approaches in order to assess their applicability.

7.2 Future work

Future work could focus on three possible directions:

1. Better support for design space exploration,
2. More efficient extraction of the performance indicators,
3. Extending the assumptions regarding the applications amenable to analysis.

Along the first line, further performance indicators, besides the expected deadline miss ratios, have to be found, such that they better support the design exploration phase. Such information could be useful for deciding on different scheduling policies, task mappings and architectures. If significantly different implementations of the same application need to be compared, the accuracy of the results is not very important. Rather, the results should provide crude guidelines for further exploration of one of the evaluated implementation. Therefore, the trade-off analysis complexity versus result accuracy has to be further investigated in more detail.

Along the second line, symmetries at the application level, as well as at the intermediate representation level, could be exploited in order to further reduce the analysis complexity and still obtain the desired performance indicators of interest.

Along the third line, the class of applications amenable to analysis could encompass applications with shared resources, and applications where the task mapping is not unique, but rather a set of alternative mappings is concurrently evaluated.

Appendix A

Notation Summary

Notation	Meaning
$\mathcal{A}$, $\mathcal{B}$, $\mathcal{F}$, $\mathcal{Q}$	matrices
α_i	the probability of a token leaving a subnet modelling a Coxian distribution after the i^{th} stage
α_{ij}	the probability of a token leaving a subnet modelling the i^{th} Coxian distribution after the j^{th} stage
B	the set of buses
B_i	a bus
$Bounds$	the set of bounds
b_i	the upper bound of concurrently active instances of the task graph G_i
CT	the set of communication tasks
$CTMC$	continuous time Markov chain
χ_i	a communication task
Δ	the set of task graph deadlines
δ_i	the deadline of the task graph G_i
E_i	the set of edges in graph G_i
ET	the set of ETPDFs
$ETPDF$	execution time probability distribution function
ϵ_i	the ETPDF of task τ_i
Ex_i	the execution time of the task τ_i
G	the set of task graphs
G_i	a task graph
$GSPN$	Generalised Stochastic Petri Net

Notation	Meaning
h	the number of task graphs
l	the number of buses
M	the number of processors
$Missed$	the set of expected miss deadline ratios of the task graphs
m_i	the expected deadline miss ratio of the task graph G_i
m	the number of communication tasks
$MapP$	the mapping function for processing tasks
$MapC$	the mapping function for communication tasks
Map	the mapping function
μ_i	the average firing rate of the i^{th} stage of a Coxian distrib.
μ_{ij}	the average firing rate of the j^{th} stage of the Coxian approximation of a the i^{th} generalised transition
n	the number of processing tasks
N	the number of tasks
p	the number of processing processors
P	the set of processors
P_i	a processor
PE	the set of processing elements
PE_i	a processing element
PT	the set of processing tasks
Π	the set of task graph periods
π_i	the period of the task graph G_i
r	the number of stages of a Coxian approximation
r_i	the number of stages of the Coxian approximation of the i^{th} generalised transition
S	stochastic process state space
SMP	semi-Markov process
T	the set of tasks
TRG	tangible reachability graph
t_i	a processing task
τ_i	a task
$^\circ\tau_i$	the set of predecessor tasks of the task τ_i
τ_i°	the set of successor tasks of the task τ_i
V_i	the set of vertices of graph G_i

Appendix B

Elements of Probability Theory and Stochastic Processes

Definition 1 *Let* Ω *be a set and* $\mathcal{F} \subseteq 2^{\Omega}$. $\mathcal{F}$ *is a* σ-algebra *if*

1. $\varnothing \in \mathcal{F}$
2. $A \in \mathcal{F} \Rightarrow \overline{A} \in \mathcal{F}$
3. $\forall i \in I \subseteq \mathbb{N}, A_i \in \mathcal{F}, A_i \subseteq A_j, \forall i < j \Rightarrow \bigcup_{i \in I} A_i \in \mathcal{F}$

Definition 2 *The tuple* $(\Omega, \mathcal{F})$ *is a* measurable space *if* $\mathcal{F}$ *is a* σ*-algebra.*

Definition 3 *A set function* $\lambda : \mathcal{F} \to \mathbb{R}$ *is countably* additive *if* $A_i \in \mathcal{F}, i \in I \subseteq \mathbb{N}, A_k \cap A_l = \varnothing, \forall k, l \in I, k \neq l, \Rightarrow \lambda(\bigcup_{i \in I} A_i) = \sum_{i \in I} \lambda(A_i)$.

Definition 4 *The countably additive set function* $\mathsf{P} : \mathcal{F} \to \mathbb{R}$ *is a* probability measure *if it is positively defined,* $(\Omega, \mathcal{F})$ *is a measurable space and* $\mathsf{P}(\Omega) = 1$.

Definition 5 *The tuple* $(\Omega, \mathcal{F}, \mathsf{P})$ *is a* probability space *if* $(\Omega, \mathcal{F})$ *is a measurable space and* P *is a probability measure defined on* $\mathcal{F}$.

Definition 6 *Let* $(\Omega, \mathcal{F})$ *and* $(S, \mathcal{S})$ *be two measurable spaces and* $X : \Omega \to S$. *If* $\forall A \in \mathcal{S}, \{\omega : X(\omega) \in A\} \in \mathcal{F}$ *then* X *is a* measurable function

Definition 7 *Let $(\Omega, \mathcal{F}, \mathsf{P})$ be a probability space, $(S, \mathcal{S})$ a measurable space and $X : \Omega \to S$ a measurable function. Then X is a* random variable.

Definition 8 *Let $(\Omega, \mathcal{F}, \mathsf{P})$ be a probability space, $(S, \mathcal{S})$ a measurable space and $X : \Omega \to S$ a random variable. The probability measure $P : \mathcal{S} \to \mathbb{R}$, $P(A) = \mathsf{P}(X^{-1}A) = \mathsf{P}\{\omega : X(\omega) \in A\}, \forall A \in \mathcal{S}$ is the* distribution *of X.*

Theorem 1 *Let $(\mathbb{R}, \mathcal{B})$ be a real measurable space, where $\mathcal{B}$ is a Borel algebra, i.e. the σ-algebra of open sets (open intervals in this case). For every monotone increasing right-continuous real function $F : \mathbb{R} \to \mathbb{R}$ with $\lim_{t\to\infty} F(t) = 1$, there exists a unique probability measure P_F which is the extension on Borel sets of the unique measure function $\mathsf{P}^*_F((a, b]) = F(b) - F(a)$. Conversely, for every probability measure $\mathcal{P}$ on $(\mathbb{R}, \mathcal{B})$ there exists a unique monotone increasing right-continuous real function $F_\mathsf{P} : \mathbb{R} \to \mathbb{R}$, $\lim_{t\to\infty} F(t) = 1$ such that $F_\mathsf{P}(b) - F_\mathsf{P}(a) = \mathsf{P}((a, b])$.*

Definition 9 *Let $(\Omega, \mathcal{F}, \mathsf{P})$ be a probability space, $(\mathbb{R}, \mathcal{B})$ a real measurable space, where $\mathcal{B}$ are the Borel sets of the line, let $X : \Omega \to \mathbb{R}$ be a random variable and let $F : \mathbb{R} \to \mathbb{R}$ be the monotone increasing right-continuous real function corresponding to the distribution of X. If F is continuous, then X is a* continuous *random variable. Conversely, if $\sum_{t\in D} F(t) = 1$, where D is the set of discontinuity points of F, then X is a* discrete *random variable.*

Definition 10 *Let $(\Omega, \mathcal{F}, \mathsf{P})$ be a probability space. The set $\{X_t : t \in I\}$ of random variables with values in some arbitrary real measurable space $(S, \mathcal{S})$, set is a* stochastic process. *If I is discrete, then $\{X_t\}$ is a* discrete time *stochastic process. Otherwise, $\{X_t\}$ is a* continuous time *stochastic process. If S is finite, the stochastic process is a* discrete state *stochastic process or a* chain.

Definition 11 *Let $(\Omega, \mathcal{F}, \mathsf{P})$ be a probability space and the family of random variables on Ω $\{X_t : t \in I\}$ be a stochastic process. $x_\omega : I \to \mathbb{R}$, $x_\omega(t) = X_t(\omega)$ is a* sample path *or a* sample function *of the stochastic process.*

Definition 12 *Let $\{X_t : t \in I\}$ be a stochastic process. The intervals during which the sample paths are constant are called the* state holding times.

Definition 13 *Let $\{X_t : t \in I \subseteq \mathbb{R}\}$ be a stochastic process with state space S. If $\mathsf{P}(X_{t+u} = j | X_t = i, X_s, 0 \leq s < t, u > 0) = \mathsf{P}(X_{t+u} =$*

$j|X_t = i, u > 0), \forall t \in I, \forall i, j \in S, \forall u > 0$, *then the process exhibits the* Markovian property *and is a* Markov *process.*

Definition 14 *Let* $F : [0, \infty) \to \mathbb{R}$ *be the distribution of a real valued random variable* X. *If* F *is of the form* $F(t) = 1 - e^{-\lambda t}$ *then* X *is* exponentially distributed *or is an* exponential random variable.

Definition 15 *Let* X *be a real valued random variable and let* P *be its distribution. If* $\mathsf{P}(X \leq x+u|X > x) = \mathsf{P}(X \leq u)$, *then* P *is a* memoryless *distribution.*

Theorem 2 *The exponential distributions are the only continuous memoryless distributions.*

Theorem 3 *A Markovian continuous time stochastic process has exponentially distributed state holding times.*

Definition 16 *Let* $\{X_t : t \in I\}$ *be a continuous time stochastic process. Let* $t_1 < t_2 < \cdots < t_k < \ldots$ *be the points where the process changes state and let* I' *be their set. If the discrete time stochastic process* $\{X_n : t_n \in I'\}$ *is Markovian, then* $\{X_t\}$ *is a* semi-Markov *process and the* $\{X_n\}$ *is its* embedded Markov *process.*

Bibliography

[AB98] A. Atlas and A. Bestavros. Statistical rate monotonic scheduling. In *Proceedings of the* 19th *IEEE Real-Time Systems Symposium*, pages 123–132, 1998.

[AB99] L. Abeni and G. Butazzo. QoS guarantee using probabilistic deadlines. In *Proceedings of the* 11th *Euromicro Conference on Real-Time Systems*, pages 242–249, 1999.

[ABD$^+$91] N. C. Audsley, A. Burns, R. I. Davis, K. W. Tindell, and A. J. Wellings. Hard real-time scheduling: The deadline monotonic approach. In *Proceedings of the* 8th *IEEE Workshop on Real-Time Operating Systems and Software*, pages 133–137, 1991.

[ABR$^+$93] N. Audsley, A. Burns, M. Richardson, K. Tindell, and A. Wellings. Applying new scheduling theory to static priority pre-emptive scheduling. *Software Engineering Journal*, 8(5):284–292, 1993.

[ABRW93] N. C Audsley, A. Burns, M. F. Richardson, and A. J. Wellings. Incorporating unbounded algorithms into predictable real-time systems. *Computer Systems Science and Engineering*, 8(3):80–89, 1993.

[ACZD94] G. Agrawal, B. Chen, W. Zhao, and S. Davari. Guaranteeing synchronous message deadlines with the timed token medium access control protocol. *IEEE Transactions on Computers*, 43(3), March 1994.

[Aud91] N. C. Audsley. Optimal priority assignment and feasibility of static priority tasks with arbitrary start times. Technical Report YCS 164, Department of Computer Science, University of York, December 1991.

[BBB01] E. Bini, G. Butazzo, and G. Butazzo. A hyperbolic bound for the rate monotonic algorithm. In *Proceedings of the* 13^{th} *Euromicro Conference on Real-Time Systems*, pages 59–66, 2001.

[BCFR87] G. Balbo, G. Chiola, G. Franceschinis, and G. M. Roet. On the efficient construction of the tangible reachability graph of generalized stochastic petri nets. In *Proceedings of the* 2^{nd} *Workshop on Petri Nets and Performance Models*, pages 85–92, 1987.

[Bla76] J. Blazewicz. Scheduling dependent tasks with different arrival times to meet deadlines. In E. Gelenbe and H. Bellner, editors, *Modeling and Performance Evaluation of Computer Systems.* North-Holland, Amsterdam, 1976.

[BRH90] S. K. Baruah, L. E. Rosier, and R. R Howell. Algorithms and complexity concerning the preemptive scheduling of periodic, real-time tasks on one processor. *Journal of Real-Time Systems*, 2(4):301–324, 1990.

[Bur79] D. Y. Burman. *An Analytic Approach to Diffusion Approximation in Queueing.* PhD thesis, New York University, February 1979.

[But97] Giorgio C. Butazzo. *Hard Real-Time Computing Systems.* Kluwer Academic, 1997.

[BW94] A. Burns and A. Wellings. *Real-Time Systems and Their Programming Languages.* Addison Wesley, 1994.

[Cox55] D. R. Cox. A use of complex probabilities in the theory of stochastic processes. In *Proceedings of the Cambridge Philosophical Society*, pages 313–319, 1955.

[CSB90] H. Chetto, M. Silly, and T. Bouchentouf. Dynamic scheduling of real-time tasks under precedence constraints. *Journal of Real-Time Systems*, 2:325–346, 1990.

[DF01] R. Dobrin and G. Fohler. Implementing off-line message scheduling on Controller Area Network (CAN). In *Proceedings of the* 8^{th} *IEEE International Conference on Emerging Technologies and Factory Automation*, volume 1, pages 241–245, 2001.

[dJG00] G. de Veciana, M. Jacome, and J.-H. Guo. Assessing probabilistic timing constraints on system performance. *Design Automation for Embedded Systems*, 5(1):61–81, February 2000.

[DLS01] B. Doytchinov, J. P. Lehoczky, and S. Shreve. Real-time queues in heavy traffic with earliest-deadline-first queue discipline. *Annals of Applied Probability*, 11:332–378, 2001.

[Doo94] J.L. Doob. *Measure Theory.* Springer, 1994.

[DW93] J. G. Dai and Y. Wang. Nonexistence of Brownian models for certain multiclass queueing networks. *Queueing Systems*, 13:41–46, 1993.

[EDPP00] P. Eles, A. Doboli, P. Pop, and Z. Peng. Scheduling with bus access optimization for distributed embedded systems. *IEEE Transactions on Very Large Scale Integrated (VLSI) Systems*, 8(5):472–491, October 2000.

[EHS97] H. Ermedahl, H. Hansson, and M. Sjödin. Response-time guarantees in ATM networks. In *Proceedings of the Real-Time Systems Symposium*, pages 274–284, 1997.

[Ele02] P. Eles. System design and methodology, 2002. http://www.ida.liu.se/~TDTS30/.

[Ern98] R. Ernst. Codesign of embedded systems: Status and trends. *IEEE Design and Test of Computers*, pages 45–54, April-June 1998.

[FJ98] M. Frigo and S. G. Johnson. FFTW: An adaptive software architecture for the FFT. In *Proceedings of the IEEE International Conference on Acoustics, Speech and Signal Processing*, volume 3, pages 1381–1384, 1998.

[GI99] A. Goel and P. Indyk. Stochastic load balancing and related problems. In *IEEE Symposium on Foundations of Computer Science*, pages 579–586, 1999.

[GJ75] M. R. Garey and D. S. Johnson. Complexity results for multiprocessor scheduling under resource constraints. *SIAM Journal on Computing*, 4(4):397–411, 1975.

[GJ79] M. R. Garey and D. S. Johnson. *Computers and Intractability.* Freeman, 1979.

[GKL91] M. González Harbour, M. H. Klein, and J. P. Lehoczky. Fixed priority scheduling of periodic rasks with varying execution priority. In *Proceedings of the IEEE Real Time Systems Symposium*, pages 116–128, 1991.

[GKL94] M. González Harbour, M. H. Klein, and J. P. Lehoczky. Timing analysis for fixed-priority scheduling of hard real-time systems. *IEEE Transactions on Software Engineering*, 20(1):13–28, January 1994.

[HN93] J. M. Harrison and V. Nguyen. Brownian models of multiclass queueing networks: Current status and open problems. *Queueing Systems*, 13:5–40, 1993.

[HST97] H. Hansson, M. Sjödin, and K. Tindell. Guaranteeing real-time traffic through an ATM network. In *Proceedings of the 30th Hawaii International Conference on System Sciences*, volume 5, pages 44–53, 1997.

[HZS01] X. S. Hu, T. Zhou, and E. H.-M. Sha. Estimating probabilistic timing performance for real-time embedded systems. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 9(6):833–844, December 2001.

[JMEP00] R. Jigorea, S. Manolache, P. Eles, and Z. Peng. Modelling of real-time embedded systems in an object oriented design environment with UML. In *Proceedings of the 3rd IEEE International Symposium on Object-Oriented Real-Time Distributed Computing (ISORC00)*, pages 210–213, March 2000.

[JP86] M. Joseph and P. Pandya. Finding response times in a real-time system. *The Computer Journal*, 29(5):390–395, 1986.

[KFG$^+$92] H. Kopetz, G. Fohler, G. Grünsteidl, H. Kantz, G. Pospischil, P. Puschner, J. Reisinger, R. Schlatterbeck, W. Schutz, A. Vrchoticky, and R. Zainlinger. The programmer's view of MARS. In *Proceedings of the Real-Time Systems Symposium*, pages 223–226, 1992.

[KM98] A. Kalavade and P. Moghé. A tool for performance estimation of networked embedded end-systems. In *Proceedings of the 35th Design Automation Conference*, pages 257–262, 1998.

[Kop97] H. Kopetz. *Real-Time Systems*. Kluwer Academic, 1997.

[KRT00] J. Kleinberg, Y. Rabani, and E. Tardos. Allocating bandwidth for bursty connections. *SIAM Journal on Computing*, 30(1):191–217, 2000.

[KS96] J. Kim and K. G. Shin. Execution time analysis of communicating tasks in distributed systems. *IEEE Transactions on Computers*, 45(5):572–579, May 1996.

[KS97] C. M. Krishna and K. G. Shin. *Real-Time Systems*. McGraw-Hill, 1997.

[Leh96] J. P. Lehoczky. Real-time queueing theory. In *Proceedings of the* 18th *Real-Time Systems Symposium*, pages 186–195, December 1996.

[Leh97] J. P. Lehoczky. Real-time queueing network theory. In *Proceedings of the* 19th *Real-Time Systems Symposium*, pages 58–67, December 1997.

[LL73] C. L. Liu and J. W. Layland. Scheduling algorithms for multiprogramming in a hard-real-time environment. *Journal of the ACM*, 20(1):47–61, January 1973.

[LSD89] J. Lehoczky, L. Sha, and Y. Ding. The rate monotonic scheduling algorithm: Exact characterization and average case behaviour. In *Proceedings of the* 11th *Real-Time Systems Symposium*, pages 166–171, 1989.

[LW82] J. Y. T. Leung and J. Whitehead. On the complexity of fixed-priority scheduling of periodic, real-time tasks. *Performance Evaluation*, 2(4):237–250, 1982.

[MA95] Y. Manabe and S. Aoyagi. A feasibility decision algorithm for rate monotonic scheduling of real-time tasks. In *Proceedings of the Real-Time Technology and Applications Symposium*, pages 212–218, 1995.

[MEP01] S. Manolache, P. Eles, and Z. Peng. Memory and time-efficient schedulability analysis of task sets with stochastic execution time. In *Proceedings of the* 13th *Euromicro Conference on Real Time Systems*, pages 19–26, June 2001.

[MEP02] S. Manolache, P. Eles, and Z. Peng. Schedulability analysis of multiprocessor real-time applications with stochastic task execution times. In *Proceedings of the* 20th *International Conference on Computer Aided Design*, November 2002.

[PEP99] P. Pop, P. Eles, and Z. Peng. Scheduling with optimized communication for time-triggered embedded systems. In *Proceedings of the* 7th *International Workshop on Hardware-Software Co-Design*, pages 178–182, May 1999.

[PEP00] P. Pop, P. Eles, and Z. Peng. Bus access optimization for distributed embedded systems based on schedulability analysis. In *Proceedings of Conference on Design, Automation and Test in Europe*, pages 567–575, 2000.

[PEP02] T. Pop, P. Eles, and Z. Peng. Holistic scheduling and analysis of mixed time/event triggered distributed embedded systems. In *Proceedings of the* 10th *International Symposium on Hardware-Software Co-Design*, pages 187–192, May 2002.

[PG98] J. C. Palencia Gutiérrez and M. González Harbour. Schedulability analysis for tasks with static and dynamic offsets. In *Proceedings of the* 19th *IEEE Real Time Systems Symposium*, pages 26–37, December 1998.

[PKH01] E. L. Plambeck, S. Kumar, and J. M. Harrison. A multiclass queue in heavy traffic with throughput time constraints: Asymptotically optimal dynamic controls. *Queueing Systems*, 39(1):23–54, September 2001.

[Ple92] P. Pleinevaux. An improved hard real-time scheduling for the IEEE 802.5. *Journal of Real-Time Systems*, 4(2):99–112, 1992.

[PTVF92] W. H. Press, S. A. Teukolsky, W. T. Vetterling, and B. P. Flannery. *Numerical Recipes in C.* Cambridge University Press, 1992.

[SGL97] J. Sun, M. K. Gardner, and J. W. S. Liu. Bounding completion times of jobs with arbitrary release times, variable execution times, and resource sharing. *IEEE Transactions on Software Engineering*, 23(10):604–615, October 1997.

[SL95] J. Sun and J. W. S. Liu. Bounding the end-to-end response time in multiprocessor real-time systems. In *Proceedings of the Workshop on Parallel and Distributed Real-Time Systems*, pages 91–98, April 1995.

[SM89] J. K. Strosnider and T. E. Marchok. Responsive, deterministic IEEE 802.5 token ring scheduling. *Journal of Real-Time Systems*, 1(2):133–158, 1989.

[Spu96a] M. Spuri. Analysis of deadline scheduled real-time systems. Technical Report 2772, INRIA Rocquencourt, Le Chesnay Cedex, France, January 1996.

[Spu96b] M. Spuri. Holistic analysis for deadline scheduled real-time distributed systems. Technical Report 2873, INRIA Rocquencourt, Le Chesnay Cedex, France, April 1996.

[SS94] M. Spuri and J. A. Stankovic. How to integrate precedence constraints and shared resources in real-time scheduling. *IEEE Transactions on Computers*, 43(12):1407–1412, December 1994.

[SSDB94] J. A. Stankovic, M. Spuri, M. Di Natale, and G. C. Butazzo. Implications of classical scheduling results for real-time systems. Technical Report UM-CS-94-089, Computer Science Department, University of Massachusetts, 1994.

[Sun97] J. Sun. *Fixed-Priority End-to-End Scheduling in Distributed Real-Time Systems.* PhD thesis, University of Illinois at Urbana-Champaign, 1997.

[TC94] K. Tindell and J. Clark. Holistic schedulability analysis for distributed real-time systems. *Euromicro Jurnal on Microprocessing and Microprogramming (Special Issue on Parallel Embedded Real-Time Systems)*, 40:117–134, 1994.

[TDS+95] T.-S. Tia, Z. Deng, M. Shankar, M. Storch, J. Sun, L.-C. Wu, and J. W. S. Liu. Probabilistic performance guarantee for real-time tasks with varying computation times. In *Proceedings of the IEEE Real-Time Technology and Applications Symposium*, pages 164–173, May 1995.

[THW94] K. W. Tindell, H. Hansson, and A. J. Wellings. Analysing real-time communications: Controller Area Network (CAN). In *Proceedings of the Real-Time Systems Symposium*, pages 259–263, 1994.

[Tin94] K. Tindell. Adding time-offsets to schedulability analysis. Technical Report YCS 221, Department of Computer Science, University of York, January 1994.

[Ull75] J. D. Ullman. NP-complete scheduling problems. *Journal of Computer and System Sciences*, 10(3):384–393, June 1975.

[Wil98] R. J. Williams. Diffusion approximations for open multiclass queueing networks: Sufficient conditions involving state space collapse. *Queueing Systems*, 30:27–88, 1998.

[ZHS99] T. Zhou, X. (S.) Hu, and E. H.-M. Sha. A probabilistic performace metric for real-time system design. In *Proceedings of the 7th International Workshop on Hardware-Software Co-Design*, pages 90–94, 1999.